Effective Integrations of Constraint Programming, Integer Programming and Local Search for Two Combinatorial Optimisation Problems

Fang He, BSc, MSc

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

April 2012

Contents

List of Figures

List of Figures

List of Tables

Abstract

This thesis focuses on the construction of effective and efficient hybrid methods based on the integrations of Constraint Programming (CP), Integer Programming (IP) and local search (LS) to tackle two combinatorial optimisation problems from different application areas: the nurse rostering problems and the portfolio selection problems. The principle of designing hybrid methods in this thesis can be described as: for the combinatorial problems to be solved, the properties of the problems are investigated firstly and the problems are decomposed accordingly in certain ways; then the suitable solution techniques are integrated to solve the problem based on the properties of substructures/subproblems by taking the advantage of each technique.

For the over-constrained nurse rostering problems with a large set of complex constraints, the problems are first decomposed by constraint. That is, only certain selected set of constraints is considered to generate feasible solutions at the first stage. Then the rest of constraints are tackled by a second stage local search method. Therefore, the hybrid methods based on this constraint decomposition can be represented by a two-stage framework "feasible solution + improvement". Two integration methods are proposed and investigated under this framework. In the first integration method, namely a ***hybrid CP with Variable Neighourhood Search (VNS)*** approach, the generation of feasible initial solutions relies on the CP while the improvement of initial solutions is gained by a simple VNS in the second stage. In the second integration method, namely a ***constraint-directed local search***, the local search is enhanced by using the information of constraints. The experimental results demonstrate the effectiveness of these hybrid approaches.

Based on another decomposition method, Dantzig-Wolfe decomposition, in the third integration method, a ***CP based column generation***, integrates the feasibility reasoning of CP with the relaxation and optimality reasoning of Linear Programming. The experimental results demonstrate the effectiveness of the methods as well as the knowledge of the quality of the solution.

Abstract

For the portfolio selection problems, two integration methods, which integrate Branch-and-Bound algorithm with heuristic search, are proposed and investigated. In ***layered Branch-and-Bound*** algorithm, the problem is decomposed into the subsets of variables which are considered at certain layers in the search tree according to their different features. Node selection heuristics, and branching rules, etc. are tailored to the individual layers, which speed up the search to the optimal solution in a given time limit. In ***local search branching Branch-and-Bound*** algorithm, the idea of local search is applied as the branching rule of Branch-and-Bound. The local search branching is applied to generate a sequence of subproblems. The procedure for solving these subproblems is accelerated by means of the solution information reusing. This close integration between local search and Branch-and-Bound improves the efficiency of the Branch-and-Bound algorithm according to the experimental results.

The hybrid approaches benefit from each component which is selected according to the properties of the decomposed problems. The effectiveness and efficiency of all the hybrid approaches to the two application problems developed in this thesis are demonstrated. The idea of designing appropriate components in hybrid approach concerning properties of subproblems is a promising methodology with extensive potential applications in other real-world combinatorial optimisation problems.

Acknowledgements

I would like to acknowledge so many people who helped me accomplish this thesis. First of all, I would like to thank my supervisor, Dr Rong Qu, for her constant guidance, support and encouragement throughout this research, and for giving me a wonderful opportunity to study at the University of Nottingham.

I would like to thank every staff and researcher in the Automated Scheduling, Optimisation and Planning (ASAP) research group for their help and assistance. I would like to thank Dr. Andrew Parkes for the discussions that directly or indirectly contribute to my work.

I would also like to thank Mick Pont, the deputy manager of NAG development division and David Sayers, the principal technical consultant of NAG, because they gave me the opportunity to work on the portfolio selection project, and to validate on this project where many of the ideas in this thesis are presented.

I would like to thank my husband, Lin. He not only gave me consistent help and support during my PhD study, but also made the last few years a truly wonderful time with his love and company.

I would like to thank my parents and brother for believing in me and supporting me in all that I have done. Without their love and support, I could not have made it this far.

Finally, to the new member of our family, Emily my love, who was born at the same time as this thesis, Mummy loves you forever!

Chapter 1 Introduction

This thesis focuses on the construction of effective and efficient hybrid methods to tackle two combinatorial optimisation problems from different application areas. These two problems both come from real-world applications where essential and complex features of problems are present. The first one is the nurse rostering problem which is a type of o personnel scheduling problem and the second one is the portfolio selection problem in the financial domain.

Although these two problems come from different application domains, they share some common features: (1) they are complex due to the presence of the large set of constraints which represent the real-world restrictions, e.g. logical restriction, and resources restriction, etc. For example, in the nurse rostering problem, large set of logical restrictions regulate the working patterns, such as an early shift should not be assigned after a late shift, etc. In the portfolio selection problem, resources restriction regulates that the amount of assets can be invested in a portfolio should be within the budget, etc. (2) they tend to be large. For example, in the nurse rostering problem, usually, a variety of shifts needs to be assigned along the scheduling horizons. The number of nurses that need to be scheduled is usually large. In the portfolio selection problem, hundreds of assets in the finance market need to be optimised.

Due to these two main features of the problems, in general, solving these problems is computationally challenging. To tackle these complex and large-scale problems efficiently, we construct hybrid solution methods in this thesis. These hybrid methods integrate and take the advantage of different techniques from different disciplines.

1.1 Hybrid methods to Nurse Rostering Problem

The Nurse Rostering Problem (NRP) consists of assigning a certain set of tasks to a certain set of nurses, subject to a large set of constraints which are related to the working regulations and personnel preferences. On the one hand, this problem

represents an important administration activity in modern hospitals, thus solving the problem efficiently is important for both practitioners and administrators in hospitals, and has a positive impact on nurses' working conditions, which is firmly related to the quality of the healthcare [1]. On the other hand, most of NRPs in real-world are NP-hard [2]. In computational complexity theory, NP stands for *nondeterministic polynomial*. NP is the class of the problems that can be solved in *polynomial* time by a *nondeterministic* algorithm. NP-hard is the class of the problems that "at least as hard as any problem in NP". Solving the complex NRPs which are NP-hard efficiently can make great contributions to the research community.

Usually, a variety of shifts needs to be assigned along the scheduling horizons. The number of nurses that need to be scheduled is usually large. Most importantly, the number of constraints in the problems is large. These constraints which are related with working regulations and personnel preferences make the problem over-constrained. These are key features of NRPs.

1.1.1 Constraint Programming method to NRPs

Constraint Programming, originated from Artificial Intelligence, is a solution method to the combinatorial optimisation problems from different applications [3]. It models the problem as a set of *variables* which take values in their finite *domains* and are linked by a set of *constraints* that can be mathematical or symbolic. *Global constraints* which capture the interesting substructure of a problem have fundamental modelling capability in a Constraint Programming system. What's more, constraint *propagation* algorithms which are embedded in each global constraint can reduce the search space by removing the value assignments that are proven to be infeasible, which is called *feasibility reasoning*. The effectiveness of global constraints has been shown for solving practical problems such as rostering and scheduling, etc. in literature [4, 5].

Due to the key features of NRPs we introduced above, Constraint Programming techniques are chosen as main techniques in this thesis to solve NRPs for following reasons:

- Constraint Programming has the strength of modelling the problem with *global constraints* [5]. Global constraints serve declaratively as building blocks of the problem modelling. It has been shown in the literature that global constraints can model the complex constrained NRPs well [1].
- *Propagation algorithms* of the global constraints have been shown to be efficient to find feasible solutions to the problem, i.e. propagation algorithms enable the powerful feasibility reasoning of Constraint Programming. This feasibility reasoning of Constraint Programming is well recognized in literature and we will take the advantage of it in the construction of the hybrid methods in this thesis.
- Most importantly, the solution approach of Constraint Programming which consists of modelling, propagation and searching in the solution procedure makes it easy to be integrated with other techniques. This paves the way of hybridizing other techniques to solve NRPs.

1.1.2 Hybrid methods to NRPs

In literature, solution methods to NRPs based on Constraint Programming have been widely investigated where NRPs are usually modelled as Constraint Satisfaction Problems [1, 6, 7]. The feasibility reasoning of Constraint Programming plays an important role in these methods.

However, the real-world large-scale NRPs we tackle in this thesis are over-constrained. That is, nurses have conflicting preferences. A feasible solution does not exist that *satisfies* all the preferences when we model the problem as a Constraint Satisfaction Problem. Hence, Constraint Satisfaction Problem model cannot be applied directly, because no solutions can be found. However, we still want to find some solutions, preferably one that minimizes the total number of conflicts. In this thesis, *soft*

constraints are applied to model the conflicting preferences of nurses. We seek an (optimal) feasible solution that minimizes the violation of soft constraints.

Constraint Programming is powerful with respect to feasibility reasoning due to the efficient propagation algorithms associated with constraints [3, 4]. But it is less efficient on *optimality reasoning* without the dedicated *cost propagation* [4]. This motivates us to hybridize other techniques to tackle the optimality reasoning in over-constrained NRPs in this thesis.

Based on the above description of features of the complex and large-scale NRPs and features of Constraint Programming techniques, we investigate two different ways of *decomposition* of NRPs. After the problem is decomposed, the hybrid methods are designed where the suitable solution techniques are integrated to solve the problem according to the decompositions.

The first way to decompose NRPs is designed according to the constraint of the problem. That is, we are first concerned with certain selected set of constraints only. First, a feasible solution is generated by Constraint Programming techniques with respect to this set of constraint first. Then the rest of constraints are tackled by a second stage *local search* method. Therefore, this hybrid method can be represented by a two-stage framework "feasible solution + improvement".

Due to their efficiency, a great variety of *local search* and *meta-heuristics* methods have been applied to complex and large-scale NRPs [1, 6, 7]. A local search algorithm typically starts from an *initial* solution (an assignment of values to all the decision variables) and iteratively moves to their *neighbour*ing solutions, defined by neighbourhood operator(s), with the hope of improving the quality of the solution measured by a function *f*. Many advanced heuristic approaches, called meta-heuristics (or extensions of local search), have been developed to prevent simple local search methods from getting trapped at local optima [6]. In this thesis, we investigate the integration of Constraint Programming with local search methods under the two-stage

framework "feasible solution + improvement". With this two-stage framework, two hybrid methods are proposed and investigated. In the first hybrid method, the feasible solution is constructed by applying the feasible reasoning of Constraint Programming at the first stage. A simple Variable Neighourhood Search method is applied at the second stage to improve the feasible solution in limited computational time by taking the advantage of efficient optimality reasoning of local search. The effort is focus on the construction of feasible solution by Constraint Programming techniques in this hybrid method.

Another hybrid method is investigated under the same "feasible solution + improvement" framework. The local search in the second stage is enhanced by using the information of constraints.

Local search and meta-heuristics are efficient for *heuristically* improving solutions. However, optimality cannot be guaranteed or proven. We desire to know the quality of solutions, e.g. how far obtained solution is away from the optimal one. Therefore, we propose another decomposition method and corresponded hybrid solution method to NRPs.

The second decomposition method comes from Operational Research. As classic methods for solving combinatorial optimisation problems, Operational Research methods have been used for a long time. They are based on the mathematical representation of the problem, which is typically modeled as an Integer Program. The Integer Programming method, by relaxing some constraints (e.g. in the form of Linear Program), defines a new problem that usually can be optimally solved. The value of the optimal solution to the relaxed problem represents an optimistic estimation of the optimal solution to the original problem, and it can be used to reduce the search space. This shows the strength of optimality reasoning. Our integration of feasibility reasoning in Constraint Programming and optimality reasoning in the form of Linear Program relaxation is well motivated. Both of them can be used to solve combinatorial optimisation problems and they have complementary strengths. OR techniques are

expert at relaxation techniques and optimality analysis. Constraint programming is distinguished by its inference techniques and modeling power[5]. Rather than choose between these methods, we will integrate them to tackle the problems in this thesis.

NRPs can be modelled as Integer Programs. For these large-scale Integer Programs, one classic decomposition method, Dantzig-Wolfe decomposition, can be applied to decompose them. The column generation is an efficient algorithm for solving the decomposed problem efficiently. We apply a ***Constraint Programming based column generation*** method to NRPs. The complex NRPs are decomposed and modeled based on the column generation scheme, where the master problem is formulated as an Integer Program and the pricing subproblem is modeled and solved in Constraint Programming paradigm.

1.2 Hybrid methods to Portfolio Selection Problem

The second problem we investigate in this thesis is the Portfolio Selection Problem (PSP). PSPs arise in the financial domain. The problem is primarily concerned with finding a combination of assets that satisfies an investor's needs the best. These needs can be basically expressed as minimizing the *risk* and guaranteeing a given level of *returns* [7].

The basic problem of PSP can be modelled and solved by Linear Programming or Quadratic Programming. However, in reality the investors usually have to include side constraints which reflect the restrictions of the market into the basic model of the problems and the problems quickly become too computationally expensive for exact methods.

The key feature of the problems is the discrete optimisation feature due to the presence of these side constraints. Branch-and-Bound method is a classic method to solve the problem and seeks the optimal solution to the problem while the computational time is not restricted. However, the investors highly desire to seek good quality solutions, not

necessarily the optimal one, in a very limited computational time. At the same time, the investors desire to have the knowledge of the quality of this solution. That is, the quality of the solution should be measurable, e.g. it can be measured by the gap between this solution and optimal solution (or lower bound of the optimal solution).

As recognized by many researchers, the efficiency of Branch-and-Bound highly relies on the branching rule heuristic and the node selection heuristic, etc. Therefore, in this thesis, we aim to investigate how to integrate heuristic search into Branch-and-Bound algorithm in the forms of node selection heuristics, and branching rules, etc. to obtain measurable quality solutions in a given time limit.

Two integration methods, which integrate Branch-and-Bound algorithm with heuristic search, are proposed and investigated in this thesis. In ***layered Branch-and-Bound*** algorithm, the problem is decomposed into the subsets of variables which are considered at certain layers in the search tree according to their different features. Node selection heuristics, and branching rules, etc. are tailored to the individual layers, which speed up the search to the optimal solution in a given time limit. In ***local search branching Branch-and-Bound*** algorithm, the idea of local search is applied as the branching rule of Branch-and-Bound. The local search branching is applied to generate a sequence of subproblems. The procedure for solving these subproblems is accelerated by means of the solution information reusing. This close integration between local search and Branch-and-Bound improves the efficiency of Branch-and-Bound algorithm according to the experimental results.

1.3 Scope and aim

The aim of this thesis is to investigate how to efficiently integrate Constraint Programming, Operational Research techniques and heuristic search methods to solve the two combinatorial optimisation problems from real-world applications—the Nurse Rostering Problem and the Portfolio Selection Problem, taking the advantage and the strengths of each well developed component. The design of the integration methods

needs to consider the strength and weakness of each component, as well as the interface/interaction between them. Therefore, the design procedure of the hybrid methods in this thesis starts from the observations of the features of each problem. The problem is firstly decomposed into subproblems in certain ways, and then the suitable methods are chosen to solve the subproblems. At last the solution to the original problem is obtained by merging the solutions to the subproblems.

1.4 Structure of the thesis

This thesis investigates the construction of hybrid methods to tackle two combinatorial optimisation problems from different application domains. This thesis begins with an overview to combinatorial optimisation in chapter 2. Chapter 3 presents an introduction to two application problems we investigate in this thesis.

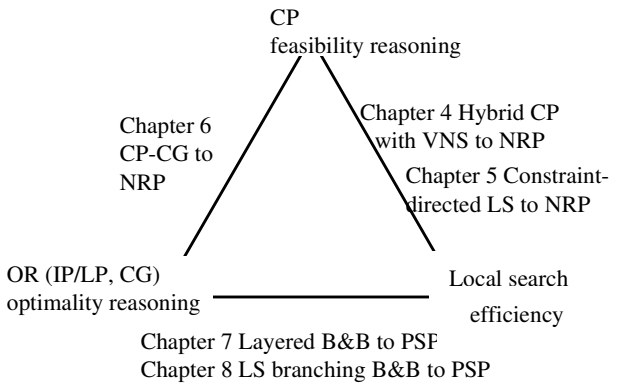


Fig. 1.1 Structure of the thesis

The structure of main body of this thesis, consisting of chapter 4 to chapter 8, is showed in Fig. 1.1. The three vertices of the triangle represent the three techniques which are investigated in this thesis. The three edges represent the integration of each two techniques which are investigated in the corresponding chapters denoted on the edges.

The right edge of the triangle in Fig. 1.1 represents the first decomposition of the NRPs and the corresponding hybrid method we investigate. This decomposition is based on the constraint. That is, certain set of constraints are considered only to generate feasible

solution. Then the rest of constraints are tackled by a second stage local search to obtain the improvement solution. Therefore, this hybrid method can be represented by a two-stage framework "feasible solution + improvement". Both chapter 4 and chapter 5 are based on this two-stage framework.

In chapter 4 we propose a ***hybrid Constraint Programming (CP) with Variable Neighourhood Search (VNS)*** solution approach to the NRPs in CP paradigm.

The model is built with primitive and global constraints. The solving of the model mainly depends on the feasibility reasoning of CP. Therefore, we first tested a small instance of NRPs. For larger-scale instances, the decomposition approach which is based on constraint is applied. The feasible solution subject to a subset of constraints only is firstly generated by solving the corresponding CSP model. Then the complete feasible solution is constructed using an iterative forward search method. The further improvement of the feasible solution is gained by using a second stage local search method.

The aim of this chapter is to investigate three fundamental and important elements in CP: (1) how to model the problem as a Constraint Satisfaction Problem (CSP), (2) design the search heuristics (variable/value ordering heuristic) and (3) design the search strategy. The experimental results show the strength of our CP approach on feasibility reasoning, as well as the weakness on optimality reasoning.

In chapter 5, we integrate Constraint Programming with local search to implement a ***constraint-directed local search*** to NRPs. The local search is enhanced by using the information of constraints (i.e. violation of a constraint and the set of variables violating the constraint) to detect and re-optimise the fragment (set of variables) that need to be improved. With this information we can define the neighbourhood of local search more generally, and the search can be guided by the cost function of these constraints. Different fragment selection strategies and the search effort that need to re-optimise the corresponding fragment are investigated. The proposed approach benefits from both the

feasibility reasoning of CP and efficiency of local search. The experimental results show the proposed approach is simple yet efficient to large and constrained NRPs.

In chapter 6, a ***Constraint Programming based Column Generation (CP-CG)*** approach is proposed to solve the NRPs. The complex NRPs are decomposed and modeled based on the column generation scheme, where the master problem is formulated as an Integer Program and the pricing subproblem is modeled and solved in CP paradigm. It integrates the relaxation and optimality reasoning of Linear Programming with the powerful expressiveness and feasibility reasoning of Constraint Programming to model and solve the complex constrained NRPs. To increase the efficiency of column generation procedure, we propose two strategies in solving the CP pricing subproblem. A Depth Bounded Discrepancy Search is employed to obtain diverse columns. A cost threshold which is adaptively tightened based on the information collected during the search is used to generate columns of good quality. These strategies show the contribution to a faster convergence in the CP-CG approach.

In the next two chapters, we investigate the integration approaches to another application problem - PSPs. The basic form of the problem, modeled as a Mixed Integer Quadratic Program, is usually solved by Branch-and-Bound algorithms. Therefore, the integration methods to the extended form of the problems are mainly based on Branch-and-Bound algorithm, integrated with heuristic/local search methods as node selection heuristic, and branching rules, etc. The integration of heuristic/local search methods into Branch-and-Bound can potentially improve the efficiency of Branch-and-Bound.

In chapter 7, we study the PSPs based on the extended classical Markowitz's mean-variance model. We consider several real-world trading constraints simultaneously in a single model. These trading constraints are modelled using integer variables (binary variable and general integer variable) that lead to a Mixed Integer Quadratic Program. The PSP with these real-world trading constraints thus has variables with different features. In this chapter we propose a multi-level Branch-and-Bound algorithm, named as ***layered Branch-and-Bound***. The problem is decomposed into the subsets of

variables, and layered to several levels in the tree according to their different features. The search is firstly performed on the top layer to produce partial solutions which define the interesting neighbourhoods of complete solutions in the search space. It then goes down to the deeper layers to obtain the complete solutions. Two branching heuristics and one node selection heuristic are tailored to the individual layers in the tree, which speed up the search to the optimal solution in given limited time.

In chapter 8, we study the PSP with non-convex transaction cost based on the extended mean-variance model using a new hybrid approach which integrates local search into Branch-and-Bound algorithm. The PSP is modelled as a Mixed Integer Quadratic Program and solved heuristically by solving a sequence of subproblems. The problem is decomposed into subproblems using variable fixing. The variables to be fixed are the core variables of the problem which are selected according to the property of the problem. The newly proposed local search branching strategy is performed on this set of core variables to generate a sequence of subproblems. Then intense Branch-and-Bound search dives into these restricted subproblems which are easier to solve compared with the original one. Due to the inherent similar structures of the subproblems, the solution information reusing accelerates the Branch-and-Bound solving procedure in solving the subproblems. The upper bound identified at early stage can prune more nodes in the tree to speed up the search to the optimal solution.

Chapter 9 concludes the thesis. The contributions of the research are summarised and the possible future research directions are discussed.

Chapter 2 Preliminaries: an overview of optimisation techniques

2.1 Introduction

This chapter introduces the fundamental concepts and solution approaches to the combinatorial optimisation problems.

Firstly, we give the definitions of the general optimisation problem and the general combinatorial optimisation problem. Then we define the scope of the research in this thesis by listing several important classes of combinatorial optimisation problems. These particular classes of combinatorial optimisation problems capture the basic structures of the two application problems that will be extensively investigated in this thesis. Hereafter, the term "combinatorial optimisation problems" refer to this list of particular problems. Secondly, we summarize the solution approaches to the combinatorial optimisation problems by generally categorising the techniques into two groups: exact solution approaches and heuristic solution approaches. Thirdly, we examine the Constraint Programming techniques to solve the combinatorial optimisation problems. The important concepts and techniques in Constraint Programming are introduced. Fourthly, we present another exact solution approach--Integer Programming and the related techniques from Operational Research. Fifthly, we present an introduction to heuristics and local search/meta-heuristics. Then we present decomposition methods and the corresponding solution approaches. These methods include domain independent, general decomposition methods as well as ones related with the application problems we will tackle in this these. Finally, we review the current mainstream integration approaches based on Constraint Programming, Integer Programming and local search. We do not intend to give an exclusive review of the integration methods. We focus on the integration methods related with the two application problems we will tackle in this thesis.

2.2 Combinatorial optimisation problems

The optimisation problem

In mathematics and computer sciences, an **optimisation problem**, or a **mathematic program**, is the problem of finding the best solution from all the feasible solutions[8]. More formally, an optimisation (minimization) problem can be stated as:

$$\min\{f(\mathbf{x}) / \mathbf{x} \in F \subset \mathbf{R}^n\} \tag{2-1}$$

where $\mathbf{x} \in \mathbf{R}^n$ is the vector of problem variables, $\mathbf{R}$ denotes the real number, $\mathbf{R}^n$ denotes an n-dimensional vector space over $\mathbf{R}$, F is the feasible region (the set of all feasible solutions), and f: $F \to \mathbf{R}$ is the objective function. Every $\mathbf{x} \in F$ is called a feasible solution to (2-1). If there is a $\mathbf{x}^* \in F$ satisfying:

$$f(\mathbf{x}^*) \le f(\mathbf{x}), \forall \mathbf{x} \in F$$

then $\mathbf{x}^*$ is called the (global) optimal solution and $f(\mathbf{x}^*)$ is called the (global) minimum with regard to (2-1).

Equivalently, an optimisation problem can be stated as follows where $\mathbf{x} \in F$ is explicitly expressed by constraint (2-2) and (2-3):

$$\min f(\mathbf{x})$$

$$s.t.\ \ g_i(\mathbf{x}) \ge 0; i = 1...n \tag{2-2}$$

$$h_j(\mathbf{x}) = 0;\ j = 1...m \tag{2-3}$$

where g_i and h_j are the functions $\mathbf{R}^n \to \mathbf{R}$, and (2-2) (2-3) represent the *constraints* of the optimisation problem.

The combinatorial optimisation problem

When an optimisation problem has a ***finite*** number of feasible solutions, the problem is called **combinatorial optimisation problem [8]**. Several important classes of combinatorial optimisation problems will be extensively investigated in this thesis.

These problems capture the basic structures of the two application problems we will tackle in this thesis. They are listed as follows:

- Linear Program: a combinatorial optimisation problem is a Linear Program if the objective function f in (2-1) and constraints g_i, h_j in (2-2) and (2-3) are the linear functions.
- Finite domain optimisation problem: a combinatorial optimisation problem is a finite domain optimisation problem if the domain of variable **x** is a finite set: $x_i \in [a_i, b_i], i = 1...n$. In this thesis, the Constraint Satisfaction Problem and Constraint Optimisation Problem in Constraint Programming paradigm are finite domain optimisation problems.
- Integer Program: If the unknown variables are all required to be integers, then the problem is called an Integer Program or Integer Linear Program.
- Quadratic Program: If the objective function f is a quadratic function and constraints g_i, h_j are linear functions, the problem is a quadratic program. When some or all of the variables are required to be integers, the problem is called Mixed Integer Quadratic Program or Integer Quadratic Program. In this thesis, we only focus on the convex quadratic objective function.

Linear Program is a special class of combinatorial optimisation problem and can be solved efficiently using the standard well-known algorithms (e.g. the Simplex algorithm) [8].

The Quadratic Program with convex quadratic objective function (for minimisation problem) can also be efficiently solved by a group of algorithms such as the extension of Simplex algorithm, etc [8].

We obtain an Integer Program (IP) when the *integrality* constraint is added on the variables of Linear Program. To solve the IP, We can relax the integrality constraint to obtain the Linear Program relaxation of the problem. If the solution to the Linear Program relaxation is integer, the Integer Program is solved. This happens with certain

network problems [8]. In general, however, when the *integrality* constraint is added, the problems are much harder, especially in many practical situations where variables are bounded. Therefore, in this thesis we focus on the *integrality* (*discrete*) feature of the problem. With regard to the convex quadratic term, since the algorithms solving Linear Program can be extended and are well studied to solve the linear system with convex quadratic objective function, the algorithms and solution techniques we investigated and proposed for Integer Program and finite domain optimisation problem can also be applied to the Mixed Integer Quadratic Program.

2.3 Summary of solution approaches to combinatorial optimisation problems

To summarise the solution approaches to the combinatorial optimisation problems, we can generally categorise the techniques into two groups: the exact solution approaches and the heuristic solution approaches. The exact solution approaches refer to the techniques that can obtain the optimal solution and prove its optimality. In the scope of this thesis, we refer solution methods from Constraint Programming as exact methods. Integer Programming techniques (e.g. Branch-and-Bound algorithm etc.) which are one branch of Operational Research are also referred as exact methods.

In general, the solution approach to the combinatorial optimisation problems requires the exploration of the search space represented by all the possible combinations of the assignments of values to the variables. The search space is explored to find the *feasible* solutions and the *optimal* solutions. Some parts of the search space can be pruned (no need to be visited). This can be done based on the feasibility or optimality. The feasibility prune (also called *constraint propagation*) is based on the feasibility reasoning which removes the assignments of values to variables that do not lead to any feasible solutions [3]. Actually, this is the main strength of Constraint Programming.

Integer Programming techniques can also be seen as exact solution approaches [8]. Branch-and-Bound algorithm is a classic method to solve the Integer Program [8]. It

systematically enumerates all candidate solutions through a tree search. The main idea of Branch-and-Bound is to avoid visiting a large subset of unpromising candidates during the tree search by using estimated upper and lower bounds of the objective function being optimised.

Actually, the Branch-and-Bound framework has been extensively used in both Constraint Programming and Operational Research; while in general, Constraint Programming emphasizes more on feasibility reasoning and Operational Research on optimality reasoning (based on the relaxation). This Branch-and-Bound framework will be described in more detail in both Constraint Programming and Integer Programming in the following sections with regard to their emphases in each domain.

The search space can be explored completely or incompletely depending on the requirement of the proof of optimality (the proof of optimality is when an optimal solution has been found, we need to prove that there is no solution that can be obtained better than that). An exact method explores the search space completely so that the optimal solution can be found with the proof of optimality. An incomplete method explores the search space (usually guided by some heuristics) to find good solutions but without the guarantee of optimality. They are used to *heuristically* and *efficiently* solve large combinatorial optimisation problems. Local search belongs to this class of methods.

Local search approaches solve the combinatorial optimisation problems from a very different angle from the systematic Branch-and-Bound applied by Constraint Programming and Integer Programming. It explores the search space by moving from solutions to neighbourhood solutions in the hope improving the value of the objective function [6]. Theoretically, it cannot guarantee the quality of the solution, but it is particularly appropriate to find good solutions for large-scale problems with reasonable time constraints.

In the following sections, we will give an introduction to each solution method as well as basic notations, concepts and techniques.

2.4 Constraint Programming

In this thesis, the term of **Constraint Programming** (CP) refers to the techniques that are used to represent and solve the Constraint Satisfaction Problem and Constraint Optimisation Problem arising from Artificial Intelligence. This section gives a brief introduction and basic notation of CP. A large part of this section is written based on the books [3] and [4].

Definition 1 (Variable and domain): Let x be a variable. The **domain** of x is a set of values that can be assigned to x. A single value is assigned to a variable. In this thesis we only consider the variables with finite domains.

Example 1: $x_1\ x_2\ x_3$ are variables and their respective domains are D_1= {2, 3}, D_2= {1, 2, 3, 4}, D_3= {1, 2}.

Definition 2 (Constraint): Consider a finite sequence of variables $\boldsymbol{X} = x_1, x_2 \ldots x_n$ where $n > 0$, with respective domains $\boldsymbol{D} = D_1, D_2 \ldots D_n$ such that $x_i \in D_i$ for all i. A **constraint** C on $\boldsymbol{X}$ is defined as a subset of the Cartesian product of the domains of the variables in $\boldsymbol{X}$, i.e. $C \subseteq D_1 \times D_2 \times \ldots \times D_n$. A constraint C is called a *unary constraint* if it is defined on one variable. A constraint C is called a *binary constraint* if it is defined on two variables. If C is defined on more than two variables, we call it a *global constraint*.

Example 2: On variables $x_1\ x_2\ x_3$ we impose the binary constraint $x_1 + x_2 \leq 4$ and the global constraint AllDifferent(x_1, x_2, x_3). The latter states that the variables $x_1\ x_2\ x_3$ should be pair wise different.

Definition 3 (Constraint Satisfaction Problem): A Constraint Satisfaction Problem (CSP), is defined by a finite sequence of variables $\boldsymbol{X} = x_1, x_2 \ldots x_n$ with respective

domains $\boldsymbol{D} = D_1, D_2 \ldots D_n$, together with a finite set of constraints $\boldsymbol{C}$, each on a subsequence of $\boldsymbol{X}$. So a CSP can be denoted as $P=(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$.

Example 3(CSP): Based on the variables given in Example 1 and constraints given in Example 2, we denote the resulting CSP as:

$$x_1 \in \{2,3\}, x_2 \in \{1,2,3,4\}, x_3 \in \{1,2\}$$
$$x_1 + x_2 \leq 4,$$
$$AllDifferent(x_1, x_2, x_3)$$

Definition 4 (Satisfaction): For a CSP $P=(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$ with $\boldsymbol{X} = x_1, x_2 \ldots x_n$, $\boldsymbol{D} = D_1, D_2 \ldots D_n$, and set of constraint $\boldsymbol{C}$, a tuple $(d_1, d_2, \ldots d_n) \in D_1 \times D_2 \times \ldots \times D_n$ satisfies a constraint C on the variables $x_{i_1}, x_{i_2}, \ldots x_{i_m}$ if $(d_{i_1}, d_{i_2}, \ldots d_{i_m}) \in C$. A tuple $(d_1, d_2 \ldots d_n) \in D_1 \times D_2 \times \ldots \times D_n$ is a solution to a CSP if it satisfies every constraint C in $P=(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$.

Example 4: For the CSP given in Example 3, a tuple (3, 1, 2) satisfies both of the constraints. It is a solution to the CSP.

Definition 5 (Constraint Optimisation Problem): Often we want to find a solution to a CSP that is optimal with respect to a certain criteria. A **Constraint Optimisation Problem** (COP) is a CSP$(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C})$ where $\boldsymbol{D} = D_1, D_2 \ldots D_n$, together with an objective function $f: D_1 \times D_2 \times \ldots \times D_n \rightarrow \mathbf{R}$ to be optimised. An optimal solution to a constraint optimisation problem is a solution to P that is optimal with respect to f. The objective function value is often represented by a variable z, together with maximizing z or minimizing z for maximization or a minimization problem, respectively.

In CP, the goal is to find a solution (or all solutions) to a given CSP, or an optimal solution (or all optimal solutions) to a given COP. The solution process interleaves *constraint propagation* or *propagation* in short, and *search*.

In summary, we can list a basic structure of CP program describing an optimisation problem as the following:

- definition of variables and domains;

- posting of the constraints among variables;
- definition of the objective function and its link with the problem decision variables;
- definition of the search strategy (which is interplayed with the constraint propagation).

2.4.1 Propagation

Constraint propagation removes (some) inconsistent values from their corresponding domains, based on the considerations on the individual constraints. By doing so, the search space can be significantly reduced. Hence, constraint propagation is essential to make constraint programming solvers efficient.

Let C be a constraint on the variables $x_1, x_2 \ldots x_n$ with respective domains $\boldsymbol{D} = D_1, D_2 \ldots D_n$, a *propagation algorithm* for C removes the values from $\boldsymbol{D} = D_1, D_2 \ldots D_n$ that do not participate in a solution to C.

Let $P=(\boldsymbol{X, D, C})$ be a CSP. We transform P into a smaller CSP P' by repeatedly applying the propagation algorithms for all constraints in $\boldsymbol{C}$ until there is no more domain reduction. This process is called **constraint propagation**. When the process terminates, we say that each constraint, and the CSP, is locally consistent and that we have achieved a notion of local consistency on the constraints and the CSP. The term *local consistency* reflects that we do not obtain a globally consistent CSP, but a CSP in which each constraint is locally consistent.

We begin by introducing the *node consistency* that deals with the unary constraints. Then we introduce the most popular notion of local consistency, the *arc consistency* that deals with the binary constraints. Then we discuss its natural generalisation to arbitrary constraints, called the *hyper-arc consistency* (also called *generalised arc consistency GAC*). Other consistency and a thorough description of the process of constraint propagation are given by [3].

Definition 6 (node consistent): We call a CSP node consistent if for every variable *x,* every unary constraint on *x* coincides with the domain of *x.*

Example 5 (node consistent): Consider a CSP with constraints defined in the form (*C,* $x_1 \geq 0 \ldots x_n \geq 0$; $x_1 \in N ... x_n \in N$),where *C* does not contain any unary constraints and *N* denotes the set of natural numbers. This CSP is node consistent, since for each variable its unique unary constraint is satisfied by all the values in the variable domain.

Definition 7 (arc consistent): Consider a binary constraint *C* on the variables *x* and *y* with a domain of D_x and D_y respectively, that is $C \subseteq D_x \times D_y$. We call *C* arc consistent if

$-\forall a \in D_x, \exists b \in D_y, (a,b) \in C,$

$-\forall b \in D_y, \exists a \in D_x, (a,b) \in C,$

We call a CSP arc consistent if all its binary constraints are arc consistent.

Example 6 (arc consistent): Consider a CSP with two variables *x* and *y* over the domain D_x= [5…10] and D_y= [3…7], and only one constraint, $x<y$. This CSP is not arc consistent. For instance, take the value 8 on the domain of *x*, there is no *y* in domain such that $8<y$.

Definition 8 (generalised arc consistent): The notion of arc consistency can be generalised in a natural way to arbitrary constraints. A constraint *C* is generalised arc consistent if for every involved domain, each element of it participates in a solution to *C*. We call a CSP generalised arc consistent if all its constraints are generalised arc consistent.

Example 7 (generalised arc consistent): Consider again the CSP in Example 3, i.e.

$$x_1 \in \{2,3\}, x_2 \in \{1,2,3,4\}, x_3 \in \{1,2\},$$

$$x_1 + x_2 \leq 4, \tag{2-4}$$

$$AllDifferent(x_1, x_2, x_3) \tag{2-5}$$

We apply constraint propagation until both constraints are generalised arc consistent:

$$\begin{array}{lllll} x_1 \in \{2,3\} & x_1 \in \{2,3\} & x_1 \in \{3\} & x_1 \in \{3\} & x_1 \in \{3\} \\ x_2 \in \{1,2,3,4\} \xrightarrow{(2-4)} & x_2 \in \{1,2\} \xrightarrow{(2-5)} & x_2 \in \{1,2\} \xrightarrow{(2-4)} & x_2 \in \{1\} \xrightarrow{(2-5)} & x_2 \in \{1\} \\ x_3 \in \{1,2\} & x_3 \in \{1,2\} & x_3 \in \{1,2\} & x_3 \in \{1,2\} & x_3 \in \{2\} \end{array}$$

The two constraints are examined sequentially, as indicated above by the arcs. We first examine constraint (2-4), and deduce that values 3 and 4 in D_2 do not appear in a solution to it. Then we examine the constraint (2-5) and remove value 2 from D_1. This is because x_2 and x_3 saturate values 1 and 2. Next we need to re-examine constraint (2-4) and remove value 2 from D_2. Then we consider constraint (2-5) again and remove value 1 from D_3. The resulting CSP is generalised arc consistent. In fact, we have found a solution to the CSP.

Constraint propagation algorithms are the algorithms that achieve local consistency. In the literature, it is also called *filtering algorithm*. There are general constraint propagation algorithms to achieve the node consistency and arc consistency, etc. (see book [3]). Arc consistency is the basic propagation mechanism that is used in almost all solvers. The most well-known algorithm for arc consistency is AC3[9], proposed by Mackworth for the binary constraint network and was extended to GAC in arbitrary constraints in [10].

The main component of GAC3 (illustrated in Fig. 2.1) is the revision of an arc, that is, the update of a domain with respect to a constraint. Updating a domain D_i of variable x_i with respect to a constraint C means removing every value in its domain which is not consistent with C. The function Revise(x_i, C) takes each value d_i in D_i in turn (line 2), and looks for a support on C for d_i (line 3). If such a support is not found, d_i is removed from the domain D_i and the fact that D_i has been changed is flagged (line 4-5). The function returns true if the domain has been reduced, false otherwise (line 6).

Algorithm: AC3/GAC3

Function **Revise** (in x_i: variable; C: constraint)**: Boolean:**
Begin
1: CHANGE=false;
2: **for each** $d_i \in D(x_i)$ do
3: **if** value d_i does not have a support on constraint C **then**
4: remove d_i from $D(x_i)$
5: CHANGE=true
6 : return CHANGE
Function AC3/GAC3(in X): **Boolean**:
Begin
/*initialization*/
7: $Q \leftarrow \{(x_i, C) \mid C \in \mathrm{C}, x_i \in X(C)\}$;
/* propagation*/
8: **while** $Q \neq \varnothing$ do
9: select and remove (x_i, C) from Q
10: **if Revise** (x_i, C) then
11: **if** $D(x_i) = \varnothing$ **then** return false:
12: **else** $Q \leftarrow Q \cup \{(x_j, C') \mid C' \in \mathrm{C} \wedge C' \neq C \wedge x_i, x_j \in X(C') \wedge j \neq i\}$
13: return true
end

Fig. 2.1 AC3/GAC3 [4]

The main algorithm is a simple loop that revises the arcs until no change occurs, to ensure that all domains are consistent with all constraints. To avoid too many useless calls to Revise function (as this is the case in every basic AC algorithm such as AC1or AC2 [9]), the algorithm maintains a list of Q of all the pairs (x_i, C) for which we are not guaranteed that D_i is arc consistent on C. In line 7, Q is filled with all possible pairs (x_i, C) such that $x_i \in X(C)$. Then the main loop (line 8) picks the pairs (x_i, C) in Q one by one (line 9) and calls Revise (x_i, C) (line 10). If D_i is wiped out, the algorithm returns false (line 11). Otherwise, if D_i is modified, it can be the case that a value for another variable x_j has lost its support on a constraint C' involving both x_i and x_j. Hence, all pairs (x_j, C') such that $x_i, x_j \in X(C')$ must be again recorded in Q (line 12). When Q is empty, the algorithm returns true (line 13) as we are guaranteed that all arcs have been revised and all remaining values of all variables are consistent with all constraints.

AC4 is proposed by Mohr and Henderson to improve the time complexity [11] [12]. AC 6 and AC 2001 are other improvements of the arc consistency algorithms. Here we only present the main techniques to enforce arc consistency. Other techniques exist to reduce the cost of arc consistency. They are usually based on one of the arc consistency algorithms presented above to improve its performance. For more details about the algorithms we refer to [4].

There are also specific constraint propagation algorithms tailored for certain global constraints. Actually, this is the field that attracts research on the integration of CP with OR techniques. The integration of specialized graph algorithms, such as matching and network flow algorithms [13] for reducing variable domains in global AllDifferent and cardinality constraint are some of such successful examples.

Constraint propagation is usually applied each time when a domain has been changed. Consequently, the propagation algorithm that we apply to make a CSP locally consistent should be as efficient as possible. However, a propagation algorithm does not need to remove all such values, as this may lead to an exponential running time due to the nature of some constraints (see the complexity analysis of arc consistency in [4]).

2.4.2 Search

After constraint propagation, we usually encounter three kinds of scenarios:

- the problem is inconsistent, which means no feasible solution exist;
- there is only one value in each variable's domain, which means we found the solution;
- there is more than one value in each variable's domain, which means we have to start to search for the solution.

The solution process of CP uses a search tree, which is a particular rooted tree. The vertices of search trees are often referred to as nodes. The arcs of search trees are often

referred to as branches. Further, if (u, v) is an arc of a search tree, we say that v is a direct descendant of u and u is the parent of v.

Definition 9 (Search tree): Let P be a CSP. A search tree for P is a rooted tree such that:

- its nodes are CSPs,
- its root is P,
- if $P_1 \ldots P_m$ where $m>0$ are all direct descendants of P_0, then the union of the solution sets of $P_1 \ldots P_m$ is equal to the solution set of P_0.

We say that a node P of a search tree is at depth d if the length of the path from the root to P is d.

Definition 9 is a very general notion. In CP, a search tree is dynamically built by *splitting* a CSP into smaller CSPs, until we reach a *failed* or a *solved* CSP. A CSP is split into smaller CSPs either by splitting a constraint (for example a disjunction) or by splitting the domain of a variable. For more information about splitting we refer readers to [3]. In this thesis we only apply the latter.

At each node in the search tree we apply constraint propagation to the corresponding CSP. As a result, we may detect that the CSP is inconsistent, or we may reduce some domains of the CSP. In both cases fewer nodes need to be generated and traversed, so the propagation can speed up the solution process. An example of a search tree in which we refer explicitly to constraint propagation and splitting is depicted in Fig. 2.2.

In Fig. 2.2, the constraint propagation (as introduced in section 2.4.2) and splitting are applied in an alternated fashion. Such a general definition allows to model arbitrary forms of constraint propagation and splitting [3]. The most common form of domain splitting consists of *labeling* of the domain of a variable. Informally, it consists of taking a variable, say x, and splitting its domain into singleton sets. Each such singleton set, say $\{a\}$, corresponding to a CSP in which the domain of the variable x is replaced by $\{a\}$. Another domain splitting consists of *enumerating* a domain of a variable,

known as *binary tree*. In the implementation of the algorithms, binary tree is applied in the solver system we applied.

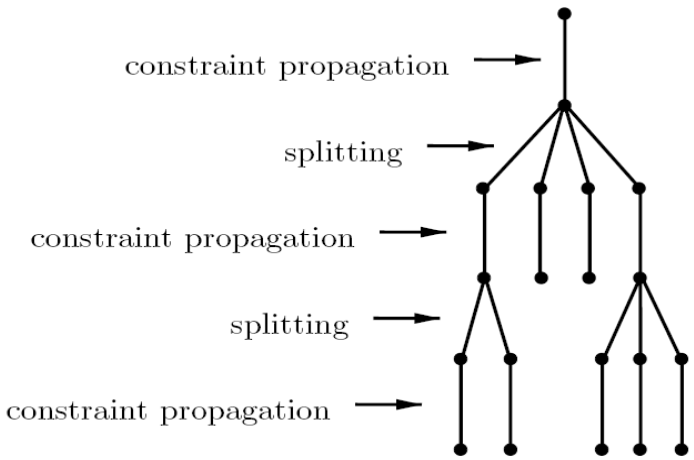


Fig. 2.2 A search tree for a CSP [4]

Variable and value ordering heuristics

To split the domain of a variable, we first select a variable and then decide how to split its domain. This process is guided by variable and value ordering *heuristics*. *Heuristic* is defined as a 'rule of thumb' based on domain knowledge from a particular application, which gives guidance in the solution of a problem [6]. Heuristic plays an important role in solving procedure not only in CP paradigm, but also in local search and meta-heuristics which will be detailed in section 2.7.

Variable and value ordering heuristics impose an ordering on the variables and values, respectively. The order in which variables and values are selected has a great impact on the search process. A variable ordering heuristic imposes a partial order on the variables with non-singleton domains. An example is the "most constrained first" variable ordering heuristic. It orders the variables with respect to the number of their appearance in the constraints. A variable that appears the most often is ordered first. It is likely that changing the domains of such variables will cause more values to be removed by constraint propagation. Another variable ordering heuristic is the "smallest domain

first" heuristic. This heuristic orders the variables with respect to the size of their domains. A variable that has the smallest domain is ordered first. The advantages of this heuristic are that less nodes are needed to be generated in the search tree, and that inconsistent CSPs are detected earlier. In case two or more variables are incomparable, we can for example apply the lexicographic ordering to these variables and obtain a total order.

A value ordering heuristic induces a partial order on the domain of a variable. It orders the values in the domain according to a certain criterion. An example is the lexicographic value ordering heuristic, which orders the values with respect to the lexicographic ordering [6]. Another example is the random value ordering heuristic, which orders the variables randomly. In case a value ordering heuristic imposes a partial order on a domain, we can apply the lexicographic or random value ordering heuristic to incomparable values to create a total order. A value ordering heuristic is also referred to as a branching heuristic because it decides the order of the branches in the search tree.

Search strategies

A search strategy defines the traversal of the search tree. In this thesis we apply the search strategies such as Depth First Search and Limited Discrepancy Search. We assume that all direct descendants of a node in a search tree are totally ordered, for example based on the value ordering heuristic.

First we describe Depth First Search (DFS) [3] which starts at the root node and proceed by descending to its first descendant. This process continues until a leaf is reached. Then it backtracks to the parent of the leaf and descends to its next descendant, if it exists. This process continues until it backtracks to the root node and all its descendants have been visited.

Next we describe Limited Discrepancy Search (LDS), introduced by Harvey and Ginsberg [14] (illustrated by Fig. 2.3). LDS is motivated by the following idea. Suppose

we have good heuristics to build the search tree, i.e. the first leaf that the search visits is likely to be a solution. If this leaf is not a solution, it is likely that only a small number of mistakes were made along the path from the root to this leaf. Hence, LDS visits the nodes next to the leaf whose paths from the root differ only in one choice from the initial path. LDS continues this process by gradually visiting the leaves with a higher discrepancy from the initial path. Formally, let P_0 be a node with ordered descendants $P_1, P_2 \ldots P_m$. The discrepancy of P_i is the discrepancy of P_0+i-1 for $i = 1; 2 \ldots m$. The discrepancy of the root node is 0. LDS can now be described as:

- Set the level of discrepancy $k = 0$. Start at the root node and proceed by descending to its first descendant provided that its discrepancy is not higher than k. This process continues until the search reaches a leaf. Then it backtracks to the parent of the leaf and descends to its next descendant, provided that it exists and its discrepancy is not higher than k. This process continues until it backtracks to the root node and all its descendants whose discrepancy is not higher than k have been visited. Set $k = k + 1$ and repeat this process until it is back at the root node and all its descendants have been visited.

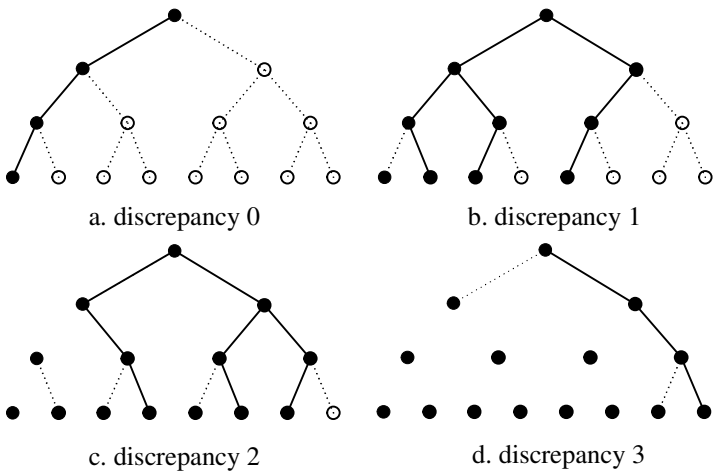


Fig. 2.3 Limited Discrepancy Search [14]

In CP, constraint modelling, variable/value order heuristics and search strategies interact in the whole procedure of problem solving. None of these decisions can be made independently from the others.

Constraint Optimisation Problem

The search for an optimal solution (or all optimal solutions) to a Constraint Optimisation Problem (COP) is performed similar to the search for a solution to a CSP. Recall that a COP consists of a CSP together with an objective function f to be optimised (see Definition 5). Assume (without loss of generality) that we want to minimize f. The objective function value is represented by a variable z. When we find a solution to the CSP, the corresponding value of z, say $z = a$, serves as an upper bound for the optimal value of f. We then add the constraint $z < a$ to all CSPs in the search tree and continue.

Perhaps the most widely used technique for solving COP is Branch-and-Bound, a well known method in both CP and OR. Next we give a brief introduction to the Branch-and-Bound in CP.

The Branch-and-Bound uses bounding heuristic to prune the search space. This (lower or upper) bounding heuristic, denoted by h, is a function that maps assignments (even partial assignments) to a numeric value to serve as an estimate of the objective function [15]. More precisely, h applied to some partial assignment is an estimate of the best values of the objective function applied to all solutions (complete assignments) that rise by extending this partial assignment [15]. Naturally, the efficiency of the Branch-and-Bound method is highly dependent on the availability of good heuristics. In such a case, the Branch-and-Bound algorithm can prune the search subtrees where the optimal solution does not settle. Note that there are two possibilities when the subtree can be pruned:

- there is no solution in the subtree at all,
- all solutions in the subtree are suboptimal only (they are not optimal).

Of course, the closer the heuristic is to the objective function, the larger the subtree that can be pruned. On the other hand, we need a reliable heuristic ensuring that no subtree where the optimal solution settles is pruned. This reliability can be achieved easily if the

heuristic is **admissible**. That is, in case of minimization problem, the heuristic (i.e. the lower bounding heuristic) is an underestimation of the optimisation function, i.e. the value of the heuristic function is not higher than the value of the objective function. In case of maximization problems, we require the heuristic (the upper bounding heuristic), to be an overestimation of the objective function. In both cases, we can guarantee the soundness and completeness of the Branch-and-Bound algorithm [15].

There exist several modifications of the Branch-and-Bound algorithm; we will present here the depth first Branch-and-Bound method that is derived from the backtracking algorithm for solving a COP. The algorithm uses two global variables to store the current upper bound (when minimizing the optimisation function) and the best solution found so far. It behaves like chronological backtracking algorithm except that the value of the heuristic function, i.e. lower bound is computed as soon as a value is assigned to the variable. If this lower bound value exceeds the upper bound, then the subtree under the current partial assignment is pruned immediately. Initially, the bound is set to (plus) infinity and during the computation it records the value of the objective function for the best solution found so far.

2.4.3 Global constraint

A global constraint is a constraint that captures the relationship among a number of variables. The same relationship can be expressed by a conjunction of several simpler constraints (primitive constraint). For example, global constraint AllDifferent(x_1,x_2,x_3) specifics that the values assigned to variables x_1,x_2,x_3 must be pair wise distinct. It also can be expressed by the conjunction of the simpler constraints as $x_1 \neq x_2$, $x_2 \neq x_3$, $x_3 \neq x_1$.

Global constraints capture the interesting substructures of a problem. It serves as building blocks for both of the problem modelling and the problem solving. A number of global constraints are practically useful and the efficient propagation algorithm exists with them. These global constraints usually encapsulate propagation algorithms. Industrial implementations of global constraints have been shown the effectiveness of it

for solving practical problems in different areas such as rostering, scheduling, resource allocation and configuration.

Example 8: ***Global cardinality constraint*** (*gcc*) is also named as *distribute* (see [5] pages 420-450). It bounds the number of times of the certain values been taken by variables. It is written as:

cardinality(***x*, *v*, *l*, *u***)
where ***x*** is a set of variables (x_1, …, x_n); ***v*** is a set of domain values of the variables ***x***; ***l*** and ***u*** are nonnegative integers defining the lower and upper bounds of times of the value ***v*** been taken for variable ***x***, respectively. The constraint defines that, for j = 1, …, m, at least l_j and at most u_j of the variables take the value v_j.

Consider for example the constraint cardinality ((x_1, x_2, x_3, x_4),(a, b, c),(1,1,0),(2,3,2)) with domains D_{x1}= D_{x3}={a}, D_{x2}={a, b, c}, D_{x4}= {b, c}. The constraint requires that at least one, and at most two, of the variables x_1, x_2, x_3, x_4 take the value a, and analogously for values b and c. Obviously a must be assigned to x_1 and x_3, which means a cannot be used again and therefore can be removed from the domain of x_2. It will be seen shortly that no other values can be removed from the domains.

The domain consistency filtering algorithm for cardinality is based on a network flow model and is presented in [16]. Some improved algorithms appear in [17], which shows that achieving domain consistency for cardinality is NP-hard. The algorithm in [18] also computes bounds consistency for cardinality.

There are other global constraints that have been proposed and studied, such as *sequence*, and *stretch*, etc. These global constraints are especially useful in the domain of personnel scheduling problem. We will study these constraints in detail in the following chapters.

2.4.4 Soft constraint

Many real-life problems are over-constrained. In NRPs for example, nurses often have conflicting preferences. To such problems there does not exist a feasible solution that

respects all the preferences. However, we still want to find some solutions, preferably one that minimizes the total number of conflicts. In case of the NRP example, we may want to construct a roster in which the number of respected preferences is satisfied as many as possible among the employees.

In classic CSP, we seek feasible solutions to a given problem. Hence, we cannot apply CSP directly to the over-constrained problems, because no solutions can be found. There have been several methods proposed as remedies. Most of these methods use so-called *soft constraints* that are allowed to be violated. The constraints that are not allowed to be violated are called *hard constraints*.

Valued-CSPs [19] and semi-rings-CSPs [20] are two *generic* paradigms for over-constrained CSP. They can be seen as extensions of the classic CSP, which allow tackling over-constrained problems or preferences between solutions to be dealt with.

There are various *specific* frameworks for over-constrained CSP. The simplest framework, Max-CSP framework tries to maximize the number of satisfied constraints. In this framework all constraints are either violated or satisfied, the objective is equivalent to minimizing the number of violated constraints. Max-CSP has been extended to the Weighted-CSP framework by [21] and [22], associating a degree of violation (not just a Boolean value) to each constraint and minimizing the sum of all the weighted violations. The Possibilistic-CSP framework proposed by [23] associates a preference to each constraint (a real value between 0 and 1) representing its importance. The objective of the framework is the hierarchical satisfaction of the most important constraints, i.e. the minimization of the highest preference level for a violated constraint. The Fuzzy-CSP framework proposed by [24-26] is somewhat similar to the Possibilistic-CSP but a preference is associated to each tuple of each constraint. A preference value of 0 means the constraint is highly violated and 1 stands for satisfaction. The objective is to maximize the smallest preference value induced by a variable assignment. All above mentioned frameworks can be described as a specification of the two generic paradigms. For more information about how the generic

and specific frameworks model and solve the over-constrained CSPs, we refer to several surveys and tutorial papers [27-29].

Another approach to model and solve over-constrained problems was proposed by [27] and refined by [28]. The idea is to identify a "cost" variable z with each soft constraint c, and replace the constraint c by the disjunction $((c \wedge (z = 0) \vee (\bar{c} \wedge (z > 0))$ where $\bar{c}$ is a constraint of the type $z = \mu(c)$ for some violation measure $\mu(c)$ depending on c. The newly defined problem is not over-constrained anymore.

If we are asked to minimize the (weighted) sum of violation costs, we can solve the problem with a traditional CP solver. In this thesis, we follow the scheme proposed by Regin et al. [27] to soften global constraints.

A violation measure for a soft constraint $c(x_1 \ldots x_n)$ is a function μ. This measure is represented by a cost variable z, which is to be minimized. There exist several useful violation measures for soft constraints, such as variable-based violation measure and decomposition-based violation measure [29]. The variable-based violation measure counts the minimum number of the variables that need to change their values in order to satisfy the constraints. The decomposition-based measure counts the number of the constraints in the binary decomposition that are violated. In this thesis, we introduce and apply variable-based measure for soft constraints.

Example 9 (variable-based violation measure): Problem

$$x_1 \in \{a,b\}, x_2 \in \{a,b\}$$
$$x_3 \in \{a,b\}, x_4 \in \{b,c\}$$
$$alldifferent(x_1, x_2, x_3, x_4)$$

is an over-constrained CSP. We need to soften the AllDifferent constraint by defining some violation measure, say μ. The variable-based violation measure μ_{var} is defined as the minimum number of variables that need to change values in order to satisfy the constraint. So for each assignment, we have the variable-based violation measure as shown in Table 2.1:

Table 2.1 Variable-based violation measure for assignments

x_1	x_2	x_3	x_4	μ_{var}
a	a	a	b	2
a	a	b	b	2
a	a	b	c	1
a	b	a	c	1
…	…	…	…	…

2.5 Operational Research techniques

Instead of following a definition, we will use the term Operational Research to specify a particular set of *methods* and *solution techniques* for the combinatorial optimisation problems listed in section 2.2. This set includes for example the techniques from Linear Programming, Integer Programming and Convex Quadratic Programming.

2.5.1 Linear Programming

There are many textbooks on Linear Programming and Integer Linear Programming.
A very good introduction to Linear Programming and Integer Programming are given by Wolsey and Nemhauser [8].

Linear Program: A Linear Program (LP) problem is characterized by a linear objective function in decision variables and by constraints described by linear inequalities or equations:

$$\begin{aligned} &\min c_1 x_1 + \ldots + c_n x_n \\ &s.t \;\; a_{11} x_1 + \ldots + a_{1n} x_n = b_1 \\ &\quad\; a_{21} x_1 + \ldots + a_{2n} x_n = b_2 \\ &\quad\; \ldots \\ &\quad\; a_{m1} x_1 + \ldots + a_{mn} x_n = b_m \\ &\quad\; x_1, \ldots x_n \geq 0 \end{aligned}$$

or using matrix notation, the compact form is:

$$\min\{\mathbf{c}^T \mathbf{x} \,|\, A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$$

where $\mathbf{x} \in \mathbf{R}^n$ is the vector of decision variables, $\mathbf{c} \in \mathbf{R}^m$ is the cost coefficient vector, $\mathbf{b} \in \mathbf{R}^m$ is the constraint vector and A is the constraint coefficient matrix with elements a_{ij}. For simplicity, we assume $n \geq m$, and the columns of A are indexed by the set $I = \{1,\ldots n\}$.

Let A_B be a basis of A, i.e. a non-singular square sub matrix of A, where the set B indexes over the columns. Let A_N be the sub matrix of A indexed by the columns in $N=I/B$. Then the set of constraints $Ax = b$ can be written as:

$$A_B x_B + A_N x_N = b$$

A solution to this equation is given by $x_B = A_B^{-1} b$ and $x_N = 0$. This solution is called a *basic* solution, and it is feasible if $A_B^{-1} b \geq 0$. The vector x_B contains the basic variables and the vector x_N constrains the non-basic variables. The **reduced cost** vector $\tilde{\mathbf{c}}^T$ is defined as:

$$\tilde{\mathbf{c}}^T = \mathbf{c}^T - \mathbf{c}_B^T A_B^{-1} A$$

The importance of the reduced cost vector is described by the following fundamental theorem: $x = (x_B, x_N)$ is an optimal solution if and only if $\tilde{\mathbf{c}} \geq 0$.

2.5.2 Integer Programming

Suppose that we have some problem instance is a minimization problem. If, as shown in Fig. 2.4, we draw a vertical line representing the objective value (the higher up this line, the larger the value), the objective value of optimal solution to the problem we are considering is somewhere on this line.

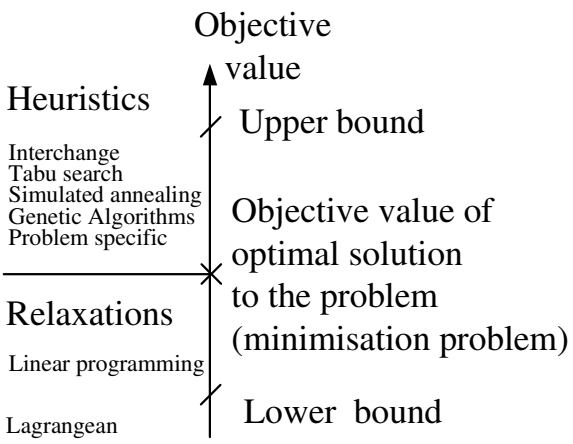


Fig. 2.4 Solution quality in a minimization problem

We do not know exactly where on this line the objective value of optimal solution lies. We denote this optimal value as * arbitrarily on the line. This optimal solution value conceptually divides the value line into two parts:

• above the optimal solution value are upper bounds, values which are above the (unknown) optimal solution value ;

• below the optimal solution value are lower bounds, values which are below the (unknown) optimal solution value.

In order to discover the optimal solution value, the algorithm that we develop must address both of the issues i.e. computing upper bounds and lower bounds. In particular the quality of these bounds is important to the computational success of the algorithm:

• we would like to find upper bounds that are as close as possible to the objective value of the optimal solution, i.e. as small as possible

• we would like to find lower bounds that are as close as possible to the objective value of the optimal solution, i.e. as large as possible.

Upper bound

Techniques for generating upper bounds are essentially beyond the scope of this thesis. Typically upper bounds are found by searching for feasible solutions to the problem.
A number of well-known general heuristic techniques are available to find feasible solutions to combinatorial optimisation problems, for example: interchange, tabu search, simulated annealing, and genetic algorithms, etc [30].

In addition, for any particular problem, we may well have techniques which are specific to the problem being solved.

Lower bound

One well-known general technique to find lower bounds is Linear Programming relaxation. In Linear Programming relaxation we take an Integer (or mixed-integer) Programming formulation of the problem and relax the integrality requirement on the variables. This gives a Linear Program which can be solved exactly using a standard algorithm (e.g. Simplex). The solution value obtained to this Linear Program gives a lower bound on the optimal solution to the original problem.

Another well-known (and well-used) technique to find lower bounds is Lagrangean Relaxation [31].

Tree search

The Branch-and-Bound algorithm is the classic tree search method to solve the Integer Programming with the help of lower bound and upper bound introduced above. In Branch-and-Bound, a tree search is used to recursively decompose the problem P into a disjunction of smaller subproblems P_i. This decomposition, or splitting procedure, creates child nodes from parent nodes of the tree, which is called *branch*. A

subproblem is not further decomposed when the node is either pruned (by feasibility or optimality pruning) or a leaf node is reached (all variables have value assignments).

In Branch-and-bound, *bound* (upper bound and lower bound) is used to prune the unpromising nodes. Every time when a feasible solution is found, such solution imposes an additional constraint, so that further solutions must have a better objective function value.

Lower bound (in a minimization problem) is an optimistic estimation of the objective function value of the optimal solution to the problem P_i. Suppose we have a method that can obtain the lower bound, if the lower bound is worse than the best solution found so far (i.e. upper bound), there is no need to solve the problem P_i. This is called *bound*. The method used to find the optimistic estimation of the optimal solution to the problem P_i is usually to solve a relaxed problem $R(P_i)$ which has the characteristics of problem P_i but is easier to solve.

Relaxation

There are various methods to generate a relaxation *R(P)* of a problem *P*. In the following we describe two techniques for modifying a problem and obtaining a relaxation. As we stated above, a relaxation can be used to obtain a lower bound which is important in Branch-and-Bound algorithm.

If the original problem *P* is described by linear constraints over integer variables, the removal of the integrity constraints for all integer variables leads to a Linear Relaxation and it can be solved by Linear Programming techniques such as Simplex algorithm.

Another common relaxation is Lagrangian Relaxation. In Lagrangian Relaxation, a set of constraints are removed from the problem and added to the objective function. The aim is to obtain an easier problem in which, however, the information associated to the

removed constraints is not lost. In fact, the addition of the constraints into the objective function allows penalising the solution that violates them.

2.5.3 Quadratic Programming

Quadratic Program problems have linear constraints, but the objective function *f* must be quadratic. Thus, the only difference between such a problem and a Linear Program problem is that some of the terms in the objective function involve the square of a variable or the product of two variables.

A number of special algorithms based upon the extending Simplex method have been developed for the Quadratic Program with **convex** quadratic objective function (for the minimization problem) [32]. These algorithms have been implemented in many Quadratic Program solvers.

2.6 The IBM ILOG suite

This thesis is the result of three years of research that was conducted and implemented based on IBM ILOG optimisation suite [33]. The IBM ILOG optimisation suite we applied in this thesis includes the following:

IBM ILOG Solver 6.2 is a C++ CP solver. It is a C++ library for CP. It includes:

- predefined classes of variables;
- predefined classes of mathematic, symbolic, and global constraints, with associated one (or more) propagation algorithm, together with a mechanism to implement new constraints;
- predefined search algorithms, together with a mechanism to write user defined tree search methods

The solver provides integer variables, enumerated variables (variables with finite set of domain), Boolean variables, and set variables (variables whose domain is a set of sets).

A constraint links the domain variables, with associated propagation algorithms. The completeness of propagation algorithms to a constraint varies and a trade-off can be made between the completeness of the propagation and the computational time.

Search in IBM ILOG Solver is typically a tree search. It provides a set of control methods that allow users to implement their own search algorithms.

When the CP system lacks a (often global) constraint that is needed to formulate a particular combinatorial optimisation problem, usually we have the choice of (1) switching to another CP system that has all required global constraints, (2) solving the relaxed model that does not require the lacking global constraints, (3) implementing the lacking constraint in the low level constraints in the current CP system, or (4) implementing a new global constraint in the system.

In this thesis, we apply the third option in a general way to tackle the combinatorial optimisation problem.

In this thesis, our algorithms are implemented based on the IBM ILOG Solver system. The individual global constraints and their filtering algorithms are embedded in the CP Solver systems. The CP Solver system provides a wide range of low level, primitive constraints along with the propagation algorithms for these constraints. The CP Solver system also provides the mechanism for combining the primitive constraints. In the state-of-art of CP Solver system, the propagation for the combining of primitive constraints is still very efficient by certain extension of the constraint propagations on the primitive constraints [34].

Therefore, Our research focus on how to apply available global constraints in the IBM ILOG Solver system and how to implement the lacking constraints in the low level constraints by combining them in the current CP system, instead of designing and implementing of new constraint and its propagation algorithm. We rely on the feasibility reasoning of the CP system and focus on how to design search algorithms which can be integrated with other techniques, i.e. OR or local search.

IBM ILOG CPLEX 10.0 is a C++ language solver for Linear Programming, Quadratic Programming and Mixed Integer Programming. Several optimisers (algorithms) are embedded in the CPLEX, such as Simplex, and Barrier, etc. for different problems. The optimisers can be chosen to solve the problems at hand according to the property of the problems. The CPLEX together with Concert Technology provide mechanism for user to write user defined search heuristics (branching rules, and node selection heuristics, etc.) for Branch-and-Bound to solve the Mixed Integer Programming, e.g. using *callback* to monitor and query information at each node of the Branch-and-Bound tree search. In this thesis we rely on the IBM ILOG optimisation products for implementing the proposed algorithms.

2.7 Heuristics and local search approaches

Except the exact methods we introduced above, a combinatorial optimisation problem coming from real life can be solved efficiently by heuristic methods which are generally based on two basic principles: heuristics/constructive heuristics and local search methods.

As we introduced in section 2.4.2, in the dictionary [35], heuristic is defined as "a 'rule of thumb' based on domain knowledge from a particular application, which gives guidance in the solution of a problem.... Heuristics may thus be very valuable in most of the time but their results or performance cannot be guaranteed."

Reeves [31] defines heuristic as "a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is".

Heuristics/constructive heuristics are typically the fast approximate methods. They generate the solutions from scratch by opportunely defined solution components to an

initially empty solution. This is being done iteratively until a solution is completed or other stop criteria are met. A well known constructive heuristic is *greedy heuristic*. An example of greedy heuristic is the nearest neighbour heuristic for the travelling salesman problem.

Constructive heuristics are usually very fast, but they often return the solutions of inferior quality. Local search methods start from some *initial solutions* and iteratively try to replace the current solution with a better one in an appropriately defined *neighbourhood* of current solution. Fig. 2.5 presents the high-level template of local search [36]. In the generation phase, a set of candidate solutions are generated from the current solution s. This set $C(s)$ is generally obtained by local transformations of the solution. In the replacement phase, a selection is performed from the candidate solution set $C(s)$ to replace the current solution. This process iterates until a given stop condition [6]. The common concepts for local search are the definition of the *neighbourhood* structure and the *initial solution*.

High-level template of local search

Input: initial solution s_0.
t=0;
Repeat
  Generate (C (s_t));// generate candidate solutions
  s_{t+1}=Select (C (s_t));// select solution from C (s_t) to replace the current solution s_t
  t=t+1;
Until stopping condition met
Output: Best solution found

Fig. 2.5 High-level template of local search [36]

The definition of the neighbourhood is a common and important ingredient of local search. The neighbourhood structure plays a crucial role in the performance of local search. If the neighbourhood structure is not adequate to the problem, local search maybe fail to solve the problem [6, 36].

Definition 10 Neighbourhood A neighbourhood function N is a mapping N: $S \rightarrow 2^S$ that assigns to each solution s of S a set of solutions $N(s) \subset S$.

A solution *s'* in the neighbourhood of *s* ($s' \in N(s)$) is called a neighbour of *s*. A neighbour is generated by the application of a move operator that performs a small perturbation to the solution *s*. For more properties and design of neighbourhood structure, we refer to book [36].

Usually, two main strategies are used to generate initial solution: a random and a greedy approach. There is always a trade-off between the use of random and greedy initial solution in terms of the quality of solutions and computational time. Generating a random initial solution maybe quick, but the local search may take much larger number of iterations to converge. In some constrained optimisiation problems, it is difficult to generate random solutions that are feasible. In this case, greedy algorithms are an alternative to generate feasible initial solutions [6].

The most basic local search method is usually called *iterative improvement local search*, since each move is only performed if the resulting neighbour solution is better than the current solution. There are typically two ways to choose the neighbour. One is *first-improvement*. A function scans the neighbourhood of current solution and returns the first solution that is better than current solution. The other is *best-improvement*. It explores the neighbourhood and returns the solution with best objective function value.

To prevent these simple local search methods from getting trapped at local optima, many advanced heuristic approaches, called **meta-heuristics** (or **extensions of local search**), have been developed [6, 30, 37]. In this thesis, the term local search includes the meta-heuristics.

In [6], meta-heuristics are defined as: "solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space" or "… any procedures that employ strategies for overcoming the trap of local optimality in complex solution space, especially those procedures that utilise one or

more neighbourhood structures as a mean of defining admissible moves to transition from one solution to another or to build or destroy solutions in constructive and destructive processes".

Heuristic and meta-heuristic methods have been investigated in recent years for tackling many hard problems, especially those combinatorial in nature. During the last 20 years many meta-heuristic approaches have been proposed. One commonly used classification distinguishes between single-point and population-based [38]. The former refers to the search methods that only maintain a single solution at each iteration while the latter manipulates a population of solutions. Examples of single-point approaches include simulated annealing, tabu search, Variable Neighourhood Search and large neighbourhood search, etc. [30]. Evolutionary algorithms, ant colony optimisation, and scatter search [6] can be regarded as population based methods.

2.8 Decomposition and solution algorithm

In this section, we review the decomposition methods and corresponding solution algorithms applied to solve the two combinatorial optimisation problems. We first introduce domain independent general decompositions methods and corresponding solution algorithms. These methods include Danzig-Wolfe decomposition and column generation algorithm, variable fixing applied as decomposition method when solving a MIP. We then introduce some ideas of decomposition methods applied in solving a specific application problem, NRPs. The detailed review of problem dependent methods, i.e. decomposition methods applied to NRP will be introduced in chapter 4 and those to PSP will be introduced in chapter 7 and 8.

2.8.1 Dantzig-Wolfe decomposition

Dantzig-Wolfe decomposition was introduced by Dantzig and Wolfe [39] and consists of reformulating a Linear Program problem into a master problem and a pricing problem for improving the tractability of large-scale problems. The master problem

typically has fewer constraints than the original problem, but the number of columns may be very large. The pricing problem generates columns for the master problem, which have the potential to improve the current solution.

Airline crew scheduling problem is a well know example which can be Dantzig-Wolfe decomposed and solved by column generation. The solution approach has also been applied to personnel scheduling problem, employee timetabling problems, etc. [40-43]. All problems tackled by Dantzig-Wolfe decomposition and column generation share some similar feature that the problem can be inherently decomposed. This can be seen as to select a subset of individual patterns (columns) from a huge pool of all possible weighted patterns (columns) to construct the best complete solution to the problem. The individual patterns should present some desired features of the problem [44]. For example, in airline crew scheduling problems, each schedule for the crew should satisfy a large set of working regulations. NRP is a type of personnel scheduling problem which shares similar features with the crew scheduling problem. This makes the Dantzig-Wolfe decomposition and column generation a good solution approach to NRPs. It also provides the possibility of hybrid methods based on the Dantzig-Wolfe decomposition where the subproblem can be solved by different techniques. This topic will be reviewed in the section 2.9.1.

More formally, in order to Dantzig-Wolfe decompose a problem; the constraint matrix should take on a certain structure and consist of a number of *independent* constraints and a number of *connecting* constraints. The independent constraints define certain specific features of the problem. Connecting constraints bind the columns together. Consider the problem:

$$\min \sum_j c_j x_j \tag{2-6}$$

$$s.t. \sum_j A x_j \le b_i \tag{2-7}$$

$$D x_j \le d \tag{2-8}$$

$$x_j \in R_+ \tag{2-9}$$

Where x_j represents decision variables; c_j represents objective function coefficients; b_i and d represent right hand side coefficients; and matrix A represents constraint coefficients a_{ij}; $i \in I = \{1, \ldots, n\}$, $j \in J = \{1, \ldots, m\}$. (2-7) represents the connecting constraints and (2-8) represents the independent constraints.

For example, in NRPs, the independent constraints can be those constraints which regulate working pattern and personal preferences for an individual nurse. The connecting constraints can be the constraints that regulate the whole roster.

We can define $X = \{x_j \in R_+, Dx_j \leq d\}$ and rewrite the problem into:

$$\min \sum_j c_j x_j$$

$$s.t. \sum_j A x_j \leq b_i \qquad (2\text{-}10)$$

$$x_j \in X \qquad (2\text{-}11)$$

Note that this problem only contains the connecting constraints (2-10). The variables x_j must satisfy the independent constraints, which thus are left out. The model holds fewer constraints than the original formulation, but the number of columns and variables may be very large.

Column generation is an efficient algorithm for solving Linear Programs with a large number of variables. The basic idea is to exploit problem substructures by *decomposing* a Linear Program into two complementary components: a *master problem* and a *pricing subproblem*. The master problem has a compact Linear Program formulation as defined by (2-9) (2-10) and (2-11) above.

Due to the large number of variables x_j, it is often impossible to solve the master problem directly. Column generation provides a way to obtain the solution indirectly [8, 45]. We define a much smaller problem, termed as *restricted master problem* (RMP) as follows:

$$\begin{aligned} &\text{(RMP) min } C\tilde{X} \\ &\text{subject to } \tilde{A}\tilde{X} \geq B \\ &\tilde{X} \geq 0 \end{aligned} \tag{2-12}$$

where $\tilde{A} \subset A$ is subset of *m'* columns and $\tilde{X}$ are the corresponding variables. The restricted master problem has less variables $\tilde{X}$, and is much smaller than the original master problem. An optimal solution to the restricted master problem provides *dual values* λ_i of each constraint i.e. $\sum_{j \in J} a_{ij} x_j \geq b_i$, of the original Linear Program.

We then need to add variables to $\tilde{X}$ and the corresponding columns to $\tilde{A}$ to yield a linear problem with the same solution value as the original master problem. In Linear Programming techniques, *reduced cost* is used to measure the improvement of the objective function coefficient for the change of the corresponding variable's value. The Linear Programming duality theory proves that only columns with *negative reduced cost* ($\pi_i < 0$) can be candidates to $\tilde{A}$. This is the way that the Simplex algorithm chooses columns internally for its basis. It can also be used to generate external columns in the column generation method [45].

Let vector $\alpha = (\alpha_1, \ldots, \alpha_n)^T$ represents a new column that we need to generate for the corresponding variables X of the master problem. α are variables of the pricing subproblem that characterize columns of A. Let F denote the feasibility region of the combinatorial objects represented by the columns of A, and let c_α denote the objective function coefficient associated to column α. With the optimal dual values λ_i associated with constraints $\sum_{j \in J} a_{ij} x_j \geq b_i$, we have the following *pricing subproblem* (PSub):

$$\begin{aligned} &\text{(PSub) } \pi_i = c_\alpha - \sum_{i=1}^{i=n} \lambda_i \alpha_i < 0 \\ &\alpha \in F \end{aligned} \tag{2-13}$$

where α are the decision variables of the pricing subproblem. Each vector α encodes combinatorial objects and solution characteristics. As stated in the pricing subproblem, $\alpha \in F$ means α is a valid solution for the subproblem.

Fig. 2.6 illustrates the column generation procedure for a Linear Program. Theoretically, it is necessary to generate all possible negative reduced cost columns before the generation phase terminates (i.e. $\{\alpha^{(1)}, \ldots, \alpha^{(k)}\} = \emptyset$). However, in practice the generation procedure is usually terminated when some conditions are met, i.e. a total number of iterations or time limit, and the master problem is then regarded as being solved [46, 47].

Algorithm 1. Column Generation for Linear Program
$\tilde{A}$: subset of feasible columns of A
λ : dual values
$\{\alpha^{(1)}, \ldots, \alpha^{(k)}\}$: columns with negative reduced cost

$\tilde{A}$:= obtain initial columns
Repeat
  λ := solve the restricted master problem $\tilde{A}$ to obtain dual values λ
  $\{\alpha^{(1)}, \ldots, \alpha^{(k)}\}$:= solve the pricing subproblem based on λ to obtain columns with negative reduced costs
  add columns $\{\alpha^{(1)}, \ldots, \alpha^{(k)}\}$ to matrix $\tilde{A}$
Until ($\{\alpha^{(1)}, \ldots, \alpha^{(k)}\} = \emptyset$) or termination condition is met

Fig. 2.6 The column generation method for Linear Program [45]

The variables of the Linear Program in the above defined column generation are continuous. If the column generation method is applied to solve Integer Programs, then the additional integer constraint $x_j \in Z^+$ is added.

The solution we obtained through column generation to the master problem is the Linear Program relaxation solution to the Integer Program. It quite often is not a valid solution due to the integer constraint $x_j \in Z^+$. In practice, two general approaches are usually used to generate integer solutions to the master problem. A standard Branch-and-Bound procedure to the restricted master problem with the current columns can be used to produce feasible integer solutions although the optimality is not guaranteed. Another approach, known as the Branch-and-Price approach [46], generates columns at each node of the search tree after branching to find the optimal solution.

2.8.2 Variable fixing

(Hard) variable fixing or diving has been used in MIP context to divide a problem into subproblems [48]. It assigns values to a selected *restricted subset of variables* of the original problem. A formal description of variable fixing is given in [48, 49]. In this thesis, variable fixing is applied to decompose the problem which will be detailed in chapter 8.

In the literature, tailored heuristics are designed based on the selection of *a restricted subset of variables*. The selected restricted subset of variables can reduce the analysis of the whole solution space to a promising region. This can be seen as a decomposition approach to the problem. Examples of such approach can be found for the knapsack problems. In the work [50], the authors propose the *core concept* (i.e. selected restricted subset of variables) for the 0/1 multidimensional knapsack problem. It has been shown to be very effective for heuristically solving the problem, achieving higher quality solution in shorter running time compared to the general IP methods. The core concept is extended to general 0/1 Integer Programming later. The extended core concept aims to reduce the original problem to a core set of variables.

Kernel search [51] is a decomposition solution framework which combines heuristic algorithms with an exact MILP solver. The steps of the kernel search are: (1) apply certain heuristics to identify the kernel, i.e. a restricted set of core variables, of the problem; (2) solve the relatively small kernel MILP problem exactly by a solver (which works as a black box); (3) identify further variables (by certain heuristics) to be inserted into the kernel; and (4) solve the updated kernel exactly again. The procedure continues until the size of kernel reach the computational limit of the exact solver.

The core concept will be applied in chapter 7 and 8 to PSPs.

2.8.3 Decomposition in NRPs

The idea of intelligently breaking up larger problems into smaller, easier to handle subproblems and then dealing with each subproblem in turn has been shown to work well on nurse rostering [52] and on other scheduling/timetabling problems [53].

In [54], constraints are categorised into shift constraints (which considered the number of staff and the skill category required for each shift), and nurse constraints (which considered the workload for each nurse including nurse preferences, consecutive shifts and the intervals between shifts). The nurse constraints were used to produce all feasible shift patterns of the whole scheduling period for each nurse, independently from shift constraints. The best combinations of these shift patterns are found using mathematical programming and meta-heuristics [54].

In [55], all the feasible weekly shift patterns are pre-defined and associated with costs which are related with preferences, requests, and the number of successive days, etc. These shift patterns are then used to construct nurse rosters by employing different heuristic decoders within a genetic algorithm to schedule both shifts and patterns for the best permutations of nurses.

In [56], high quality pre-defined schedules are employed to construct cyclic schedules for a group of nurses with the same requirements. Based on these partial cyclic schedules, the rest of the shifts are assigned to the rest of the nurses with different requirements. The problems can thus be seen as being decomposed into cyclic and non-cyclic parts.

2.9 The integration of CP and OR with LS

In this section, we review the current mainstream integration methods based on CP, OR and LS in the literature.

There are several survey papers reviewing the hybrid methods coming from different disciplines. Raidl and Puchinger review [57] the solution approaches combining Integer Programming with meta-heuristics for general combinatorial optimisation. Focacci et al. review [58] local search integrated with CP. Wallace [59] reviews the hybrid methods in CP.

In this thesis, we follow the structure of our research illustrated in Fig. 1.1 to categorise the researches in the literature. We have made an effort to highlight the key and interesting points which have not been previously studied. We do not intend to review exclusively the hybrid methods in literature; some more problem specific methods in related work (i.e. methods to nurse rostering problems or portfolio selection problems) are reviewed in the following corresponding chapters.

2.9.1 Integration of two exact methods

The combination of two exact methods, e.g. OR and CP, is a natural idea, because both methods can be used to solve the same problems, and they have complementary strengths and weaknesses [5]. Many different combinations have been studied in the past. They range from the specialized hybrid algorithms for specific problems up to the general integration in the two fields. The successful results have led to an annual international workshop on "The Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimisation Problems" (CP-AI-OR), started in 1999, and became a conference in 2004. We refer to [5, 60, 61] for a collection of successful combinations.

OR techniques consist of a large set of methods and solution techniques for combinatorial problems. This set includes for example the techniques graph theory, (Integer) Linear Programming and semi-definite programming, etc. In this thesis, we focus on combining CP and IP/LP, one of the most efficient combinations in hybrid optimisation.

The first form of the combination of CP with IP is CP based column generation or Branch-and-Price, where the subproblem is solved with CP. We refer this combination as CP-CG.

The CP-CG framework has been independently introduced by Junker et al. [42] and Yunes et al. [43] in two research groups both working on the crew management problems. In the last decade, the CP-CG framework has been applied to several different applications, such as crew assignment, bin packing, and graph coloring, etc. Here we generally categorise the application of it in two groups: basic approaches and enhanced methods. The basic approaches mainly are concerned with the modeling aspect of the problem while the enhanced methods are concerned with more advanced issues such as accelerated techniques and convergence issue of column generation.

Basic CP-CG

The pricing subproblem can be modeled in CP paradigm and the CP techniques can be used as a black box solver to solve the pricing subproblem.

In Yunes et al. [43], for the crew management problem, the master problem is formulated as a set partition problem, where each column of matrix A in (2-12) represents a roster of a crew member over a given planning horizon. The pricing subproblem is formulated as a constrained shortest path problem. The subproblem is usually solved by dynamic programming which is quite time consuming for the auxiliary graph model [43]. The CP techniques as an alternative are used to model and solve the pricing subproblem with finite domain integer variables and two global constraints: element and atmost. The first fail variable selection heuristic is the most effective method for this problem.

In Gabteni & Gronkvist [62], the problem is to find the minimum cost assignment of each aircraft to a set of flights. The master problem is formulated as a set partition

problem, where each column of matrix *A* in (2-12) represents an admissible sequence of flights. The partitioning constraints force each flight to be assigned to a unique aircraft. The pricing subproblem is defined over an acyclic connection network, where each possible flight to flight connection is an arc between two flights. The pricing subproblem is efficiently solved as a standard resource constrained shortest path problem. CP is used as a preprocessing algorithm to eliminate the infeasible arcs of the network to reduce the size of feasible solution.

In Easton et al. [63] for the traveling tournament problem (i.e. scheduling of games that involves the minimum traveled distances among team venues), a parallel solving procedure is proposed. The master problem is formulated as a set partition problem to force each team to be assigned to a single tour. The pricing subproblem consists of generating tours for each team that satisfy the sequencing constraint over home and away games. The parallel algorithm is used to check a pool of columns if the negative reduced cost columns are present. If they are present, a fixed number of columns are selected and added to the restricted master problem. Otherwise, the pool is refilled by CP column generator.

Table 2.2 Summarizing other applications which apply basic CP based CG in the literature [64]

Application	References
Urban Transit Crew Management	Yunes et al.[43, 65]
Airline Planning	Gronkvist [66, 67]
	Gabtebi and Gronkvist [62]
Traveling Tournament Problems	Easton et al. [63]
Two Dimensional Bin Packing	Pisinger and Sigurd [68]
Airline Crew Assignment	Junker et al. [42]
	Fahle et al. [45]
	Sellmann et al. [69]
	Hansen and Tomas [70]
Vehicle Routing Problem with Time Windows	Rousseau et al. [71]
Employee Timetabling	Demassey et al. [44]

Enhanced CP-CG

The bottleneck of the column generation algorithm is usually the solution to the pricing subproblem. A common technique to speed up the solution to the CP pricing subproblem is to use optimisation constraints that reduce the domain of each variable by both feasibility and optimality reasoning.

In Fahle et al. [45], the CP-CG is applied to solve an airline crew assignment problem (i.e. assigning each crew member of an airline to a set of activities). The pricing subproblem consists of generating rosters that satisfy a set of complex rules and regulations. The pricing subproblem is formulated as a CP model and solved by a CP solver. The authors propose a new global constraint with cost, called path-constraint to improve the efficiency of the subproblem solving. The effects of an incremental implementation of filtering algorithms developed for path-constraint are significant. The path-constraint is used together with other constraints to guarantee the feasibility of the generated rosters.

In Demassey et al. [44], the CP-CG approach is applied to an employee timetabling problem. The problem assigns a given set of tasks to a number of employees and minimizes the number of working employees. The master problem is formulated as a set partition problem, where the columns of matrix A in (2-12) represent feasible employee timetables. The pricing subproblem generates the minimum reduced cost timetable, defended as a sequence of activities satisfying ordering and cardinality constraints. The CP use cost-regular global constraint to model the pricing subproblem and a cost based domain filtering algorithm is designed to update both the upper bound and lower bound of the cost variables.

In solving large-scale IP problems by column generation, it becomes a critical issue to balance the computation time required to solve the LP relaxation with that required to compute an integer solution. In the context of CP-CG, this trade-off can be achieved by using special heuristics for solving the pricing subproblem.

The Lowest Reduced-Cost First (LRF) is a heuristic introduced by Fahle et al. [45] that consists of ordering the variables by decreasing reduced cost. The purpose of this strategy is to speed up the solution to the LP relaxation of the master problem by trying to generate columns that are good in terms of reduced cost. Unfortunately, this strategy is effective only during the first iterations of column generation, and becomes time consuming when the restricted master problem approaches its optimal value.

Other two labeling heuristics are introduced by Gendron et al.[72], the so-called dual strategy and the master strategy. The first strategy, dual strategy is similar to the LRF heuristic; it also sorts the variables by their reduced cost. Different from LRF, it considers the fact that CP-CG tends to generate similar columns. So it introduces randomization by a pseudo-random number *i* to choose not the smallest variable but the variable having the *i*th smallest reduced cost. This approach is concerned with the diversity of the columns. The master strategy is a heuristic that takes into account the constraints of the master problem to select the next variable to be assigned. The intuition is to generate columns that shall positively contribute to the solution to the integer master problem. For instance, when the master problem is a set partition problem, it is advantageous to generate columns that have nonzero coefficients uniformly distributed over each row. To achieve this aim, the master strategy first enumerates how many columns cover each row, and then it generates a column that considers the less covered rows. The computational results show that both the dual and the master strategy are useful. However, a combination of the two strategies could be even more effective and is worth investigating in future as suggested by the authors.

In CP-CG approach, slow convergence is another important issue affecting the performance of the approach. The slow convergence is partially due to the generation of the similar columns, particularly frequent in CP rather than other techniques [64]. The generation of very similar columns is mainly related with the standard CP search strategy, that is, depth first search.

In Sellmann et al. [69], to obtain *diverse* column, the problem dependent "diversity constraints" are added. These constraints are used to limit the times that a row is covered by every column. This idea is similar to the master strategy introduced above [72]. Limited discrepancy search is used as tree traverse strategy in a pure CP model to obtain *diverse* solution to the pricing subproblem.

In Rousseau et al. [73], a preliminary computational study on the vehicle routing problem with time windows showed that indeed the limited discrepancy search has an effect on the generation of *diverse* solution to the problem.

In Gualandi [74], the author proposes a shuffled static order of the decision variables to generate *diverse* columns: at each iteration of column generation, before the constraint solver begins the tree search, the vector of the decision variables is shuffled. This shuffled static order can be viewed as an implicit random breaking-ties strategy. The effect of shuffling is shown that it can dramatically reduces the number of iterations, since it produces distinct columns as the consequence of the constraint propagations.

2.9.2 Integration of exact method with local search

In [57], Raidl et al. present a general classification of existing approaches combining exact and (meta) heuristic algorithms for combinatorial optimisation. The two main categories are distinguished, see also Fig. 2.7.

- *Collaborative combinations.* In a collaborative environment, the algorithms exchange information, but are not part of each other. Exact and heuristic algorithms can be executed sequentially.
- *Integrative Combinations.* In integrative models, one technique is the embedded component of other techniques.

Here, we follow the classification to review the integration of exact methods with local search. Some of them has also been reviewed in [75].

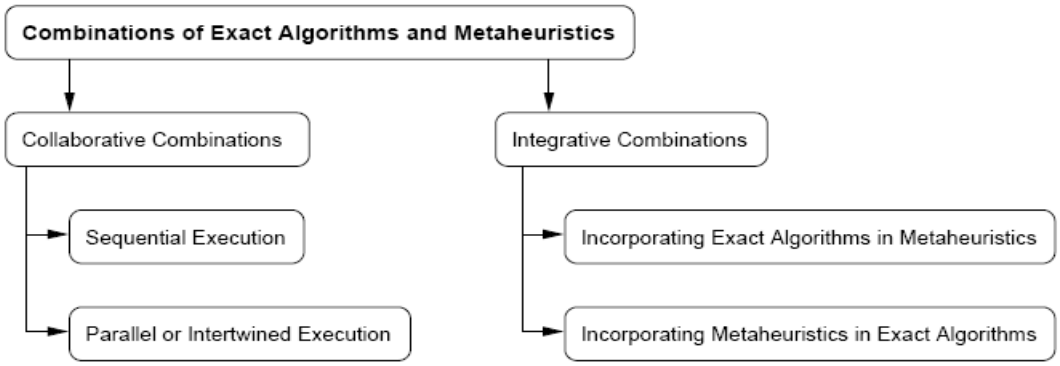


Fig. 2.7 Structural classification of exact methods and meta-heuristics combinations [57]

(1) Sequential execution:

A. Exact (CP, IP) then LS

In this class of integration, either the exact method is executed as a preprocessing before the meta-heuristics, or vice-versa. Sometimes, it is difficult to say if the first technique is used as the initialization of the second, or if the second is the post processing of the solutions generated by the first. This is one of the most direct and simple integrations. Here we present a brief review on it.

In [76], CP techniques is used to solve the relaxed problem of an original nurse rostering problem which only consists of some of the constraints in the first phase. In the second phase, adjustments with local search and tabu search are applied to improve the solution from the first phase.

In [77], the nurse rostering problems are modeled by two generic constraints within the CP paradigm to generate individual schedules. These individual schedules work as chromosomes which are put into second stage-genetic algorithm to make improvement.

B. LS then exact method

In general, the most natural approach of applying local search before an exact approach is that local search method can provide an upper bound for Branch-and-Bound (in a minimization problem, or a lower bound in a maximization problem). The meta-heuristic gives an initial solution (which serves as an upper bound) to define a reduced search space to the Branch-and-Bound. This can speed up the Branch-and-Bound search.

Another class of combination is rarer comparing with the above. The idea is that the local search procedure reaches a "plain" - an area where further improvement is hard to achieve. At this point a change to a complete search procedure such as Branch-and-Bound is possible. By learning which variable's values have proven their utilities during the local search procedure, the subsequent complete search can be restricted only to admit the values with a higher utility [78].

(2) Cooperative execution:

A. Exact (CP, B&B) in LS

Pesant & Nuijten [79] propose a neighbourhood model to find the best non-tabued neighbour of a given solution. A master problem model (local search model) and a neighbourhood model (CP model) are interacted by interface constraints. CP's pruning guides the local search to explore only the restricted, promising area of neighbourhood, leading to good approximate results in a reasonable time limit.

Shaw [80] used a local search method named Large Neighbourhood Search to solve the vehicle routing problems. During the exploring of the enlarged neighbourhood, limited discrepancy search (in CP) is used in the search tree to re-insert the visits.

In constraint-based local search proposed by Hentenryck [81], the architecture of a constraint-based local search algorithm consists of declarative and searching

components. The declarative component is used to maintain properties that can be queried to evaluate the effect of local moves. The searching component supports various abstractions to specify heuristics and meta-heuristics. A software toolkit named Comet has been developed.

In [82], the guided local search builds up penalties during a search. It uses the penalties to help local search algorithms escaping from local minima and plateaus. When the given local search algorithm settles in a local optimum, the guided local search modifies the objective function using a specific scheme. Then the local search will be operated using an augmented objective function, which is designed to bring the search out of the local optimum. The key of this method is in the way that the objective function is modified.

Cotta et al. [83] propose a framework which lays on the cooperation between genetic algorithm and a Branch-and-Bound algorithm. The Branch-and-Bound is used as an operator in the genetic algorithm. The resulting hybrid operator cleverly explores the dynastic potential (possible children) of the solutions being recombined, providing the best combination of the genome.

Jahuira et al. [84] propose hybridization between genetic algorithms and exact methods applied to the travelling salesman problem. The cooperation is introduced in the genetic functions as the authors replace the genetic crossover by a Branch-and-Bound algorithm and a minimal spanning tree algorithm.

Large neighbourhood search algorithms [80] are typically cooperation algorithms. These algorithms can be viewed as local search algorithms which use a large neighbourhood to improve the efficiency of the search. The exploration of this large neighbourhood can be either heuristic or exact. A survey of these methods can be found in [85]. Several studies propose applying exact methods to explore these large neighbourhoods to find the best solution in a subspace of the global search space to the optimised problem. These types of approaches have been proposed by Bent and Van

Hentenryck to solve the asymmetric travelling salesman problem [86], or Shaw for vehicle routing problem [80].

B. LS in exact(CP, B&B)

If we consider the cooperation between exact and heuristics methods, one of the most natural approaches is to design a heuristic to improve the search strategy of the exact method. An example of this type of cooperation has been proposed by Augerat et al. [87]. In this study, a Branch-and-Cut algorithm is proposed to solve a capacitated vehicle routing problem. The cutting plane generation is a crucial part of Branch-and-Cut algorithm. Indeed, it greatly determines their efficiency. The authors remark that the linear inequality resulting from the constraint capacities are those which provide the best cutting planes. So they propose different heuristic approaches (constructive heuristics, greedy algorithms, and tabu search algorithms) to extract a set of violated capacity constraints of the relaxed problem.

This advanced cooperation is not widely used for cooperative methods between exact algorithms and heuristic approaches. Indeed, in many studies, the authors use simple (node exploration) or specific heuristics (column generation) to optimise the exact search strategy.

Very recently, a new series of "matheuristics" workshop is proposed as a primary forum for researchers working on the hybrid methods based on mathematical programming and local search/ (meta) heuristics. The research exploits mathematical programming techniques in a (meta) heuristic framework, granting to mathematical programming approaches the problem robustness and time effectiveness which characterize metaheuristics. It also exploits the mathematical programming model formulation in the customization of a metaheuristic for the specific or general problems. We refer to [88] for a collection of successful combinations.

2.10 Conclusions

This chapter presents a review of optimisation techniques: CP, OR techniques and local search, which will be extensively investigated in the following chapters of this thesis.

CP and Integer Programming are exact optimisation methods to combinatorial optimisation problems. Global constraints, together with their propagation algorithms, serve as building blocks for both the problem modelling and the problem solving. They can be well used to model and solve the complex and large set of constraints presented in real-world combinatorial optimisation problems. The OR techniques, e.g. Linear Programming, can perform optimality reasoning through the solution to the relaxed problem of the original one, and they can also be used to reduce the search space of the problem.

Therefore, we will investigate hybrid CP approach to the NRPs in chapters 4, 5 and 6. The hybrid approach proposed in chapter 4 belongs to the category of "Sequential execution: Exact (CP, IP) then LS" introduced in section 2.9.2. These approaches are designed based on the properties of the problem and take the advantages of each component technique. We will also investigate a hybrid method which integrates CP with local search to get good feasible solutions, not necessarily the optimal solution, in a reasonable computational time in chapter 5. This hybrid approach falls in the category of "Cooperative execution, Exact (CP, B&B) in LS". CP-CG proposed in chapter 6 belongs to the category of "Integration of two exact methods" introduced in section 2.9.1.

For another application problem – the PSP, the basic problem can be modelled and solved by Linear Programming or Quadratic Programming. For the complex problems with side constraints, Branch-and-Bound algorithm, integrated with heuristics, i.e. node selection heuristic, and branching rules, etc. will be investigated in chapter 7. Another more general integration, where local search serves as branching rule in Branch-and-Bound will be investigated in chapter 8. Both of the hybrid approaches fall in the

category of "Cooperative execution, LS in Exact (CP, B&B)**".** These hybrid methods can seek good quality solutions, not necessary the optimal one, in a very limited computational time. At the same time, we can have the knowledge of the quality of this solution.

Chapter 3 Introduction to the application problems

3.1 Introduction

This chapter introduces two application problems we tackle by the integration methods. Firstly, we introduce the Nurse Rostering Problem (NRP) with terms used in the context. Then we present a brief overview of the modelling issues of NRP. We then review the previous research on the solution approaches to the NRP, categorised the methodologies used.

Secondly, we introduce the portfolio selection problem (PSP). The modelling issues and solution approaches are also introduced.

3.2 The nurse rostering problem

There is no formal definition of NRP due to the fact that a variety of the problems are present in the application. Usually the problem is described informally. In the description of NRP, some of the key terms and expressions are frequently used in literature. Here we first make a distinction between *schedule* and *roster*. In practice, the two words are often used interchangeably, but in this thesis, we denote a line-of-work for a nurse within the scheduling period as the individual nurse's schedule; whereas the overall timetable for all nurses (all schedules) is denoted as the nurse roster.

NRP is to assign each available nurse in a specific category to an individual schedule, i.e. a sequence of day-on and day-off duties. On each day-on, the nurse can be assigned to a particular shift (e.g., early, day, evening or night shift). The problem data such as the number of personnel in a ward, the number of personnel in each skill category, the demand of each category of nurses and the definition of shift types, etc. are determined at the earlier stage of staff planning, which is the first stage of the overall nurse workforce management [1, 89, 90].

There are two general types of nurse rostering: cyclical and non cyclical scheduling [91]. Each one has its advantages and disadvantages and is suitable for different situations. In cyclical scheduling, a single schedule for a fixed planning period is created, and assigned to all employees. The scheduling restarts once the end of the planning period is reached. Cyclical scheduling has a number of advantages. As everyone has the same schedule, nurses cannot feel whether their schedule is worse than that of anyone else. Secondly, once a good cyclical scheduling is produced, it can be reused until the scheduling requirements change. Cyclical scheduling does have disadvantages too. It is more challenging when the covering requirements are different from day to day, or week to week. The largest drawback in cyclical scheduling is that individual requests and preferences are very difficult to be taken into consideration and to be satisfied. So cyclical scheduling is less popular in practice[91].

Non cyclical scheduling, as the name suggests, is opposite to cyclical scheduling. Each nurse has a schedule which satisfies their personal preferences and requests. So it is more flexible than cyclical scheduling. However, it is generally much more difficult to solve. In this thesis, the NRPs investigated are all non cyclical.

The constraints in NRPs can vary from one hospital to another while the objectives can also vary. These have resulted in a whole range of NRP models and, correspondingly, a wide range of solution approaches that have been developed for these models.

In the following section of this chapter, we give a brief overview of the modeling issues of NRP. Then, we review the methods that have been used to solve NRPs of varying complexity.

3.2.1 Modelling the nurse rostering problem

Decision variables and domains

The nurse rostering is commonly described by a nurse-day view which is a direct depiction of two-dimensional duty rosters. Accordingly, the decision variables can be defined for each nurse on each day as s_{ij}, where i indexes the nurses and j indexes the days within the scheduling period. The domains of the variables consist of day-on and day-off duties.

This type of decision variable and domain are widely used in CP. For example, an off shift with 3 day-on (i.e. early, late, night) shifts can be defined as:

$$s_{ij} = \begin{Bmatrix} 0;\ if\ \ nurse\ i\ take\ Off\ \ shift\ on\ day\ j \\ 1;\ if\ \ nurse\ i\ take\ Early\ shift\ on\ day\ j \\ 2;\ if\ \ nurse\ i\ take\ Late\ shift\ on\ day\ j \\ 3;\ if\ \ nurse\ i\ take\ Night\ shift\ on\ day\ j \end{Bmatrix}$$

Table 3.1 shows part of a weekly roster where the shifts are allocated to the (total number of 8) nurses in a nurse-day view.

Table 3.1 Part of a weekly roster in a nurse-day view

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun
A	1	1	1	1	0	0	0
B	3	3	0	0	2	2	2
C	2	2	0	0	1	1	1
			…				

For 0-1 model applied in IP/MIP, the decision variables are usually customized to be s_{ijk}, where i, j are the same indexes as that for s_{ij}, k indexes the possible shifts in a day. In the above example, s_{ijk} is binary:

$$s_{ijk} = \begin{Bmatrix} 1,\ if\ \ nurse\ i\ work\ \ shift\ k\ \ on\ day\ j \\ 0,\ otherwise \end{Bmatrix}$$

Constraints

Constraints vary from different hospitals. Researchers and practitioners tend to define the constraints according to the requirements and situation of their own organizations. This can be seen in a large amount of literature in NRPs. Therefore, it is usually difficult to have a fair comparison among the different solution approaches. In order to provide a test bed for the algorithms we developed in this thesis, we choose to investigate a set of benchmark NRPs. These problems, on the one hand, are from real-world thus reflecting the request of hospitals; on the other hand, they are also tested by other researchers, accordingly the different solution approaches can be analyzed and compared fairly. Next, we present the constraints we are going to investigate in this thesis. These constraints commonly occur in the benchmark NRPs.

1. The shift coverage requirements must be fulfilled
2. Minimum rest time between shifts
3. Maximum number of shift assignments within the scheduling period
4. Maximum number of consecutive working days
5. Minimum number of consecutive working days
6. Maximum number of consecutive non-working days
7. Minimum number of consecutive non-working days
8. Maximum number of hours worked
9. Minimum number of hours worked
10. Maximum number of a certain shift type worked (e.g. maximum seven night shifts for the scheduling period)
11. Maximum number of a certain shift type worked per week (same as above but for each individual week)
12. Valid number of consecutive shifts of the same type
13. Free days after night shifts
14. Complete weekends (i.e. shifts on both Saturday and Sunday, or no shift over the weekend)
15. No night shifts before free weekends
16. Identical shift types during the weekend

17. Maximum number of consecutive working weekends
18. Maximum number of working weekends in four weeks
19. Shift type successions (e.g. Is shift type A allowed to follow B in the next day, etc)
20. Requested days on or off
21. Requested shifts on or off

Objective functions

Typically, we use standard objective functions in models, such as those in mathematical programming. For example, the objective $\min \sum p_{ij} s_{ij}$, where p_{ij} is the penalty cost of nurse i working on day j, s_{ij} are the decision variables, defines the purpose to minimize the total penalty cost for all nurses. In other situations, a penalty function can be used when feasibility cannot be guaranteed. The function is the penalty for violating constraints. This widely happens in over-constrained NRPs.

3.2.2 Solution approaches to nurse rostering problems

Several recent surveys of employee scheduling provide a large amount of information about problem models and solution methods [1, 89, 90]. In this section we review the key methods investigated in the thesis (i.e. CP, IP, and hybrid approaches) that have been used to solve NRPs of varying complexity. Some of the papers reviewed here are also reviewed in [1, 90]. In order to provide a new contribution to the research community, we have made an effort to review the papers by highlighting the key points about the feature of the problem and that of the solution methods.

(1) OR approaches: LP, and MIP, etc

In this section we review the publications which use Linear Programming and Integer Linear Programming methods to tackle the NRPs. These methods are usually used to solve the 0-1 model. The Integer Program problems are usually solved by Branch-and-

Bound algorithm, column generation, or Branch-and-Price algorithm, etc. The procedures such as branching strategy, bounding procedure, and column generation procedure are critical to the success of the algorithms.

One of the first exact optimisation approaches to NRPs was presented by Warner and Prawda [92]. The problem is formulated as a Mixed Integer Quadratic Programming problem. A solution to the problem represents a staffing pattern which specifies the number of nurses to cover the shifts for six wards. The goal is to minimize the nurse shortage costs while satisfying the total nursing personnel capacity, and the integral assignment constraint. The problem is decomposed into linear 0-1 programming master problem and small quadratic programming subproblems. Each feasible solution to the subproblems is a candidate solution for the master problem.

Bailey [93] presents an approach which combines the problem of shift planning and the assignment of those shifts to employees while considering some basic work pattern constraints. The objective is to minimize the understaffing subject to a fixed workforce size and overtime restriction. Linear Programming is efficient to identify the optimal shifts and on-off patterns. The shifts are then matched to the patterns heuristically, aiming to minimize the difference in a nurse's shift start time over the period.

Mason and Smith [40] describe column generation methods to efficiently solve a NRP using linear and Integer Programming techniques. Columns are generated by dynamic programming solving the shortest path problems with respect to the nurse's preferences for different shifts, and consecutive on-off pattern, etc.

Jaumard et al [41] solve a NRP with the objective of reducing salary cost, improving nurse preference satisfaction. They also use column generation techniques where the columns correspond to individual schedules for each nurse. The subproblem is a resource constrained shortest path problem.

Bard and Purnomo [94] combine the heuristic and Integer Programming methods to solve a NRP with up to 100 nurses and approximately 13 hard and soft constraints. The objective of the problem is to minimize the costs incurred from employing outside nurses and to maximize the satisfaction of nurses' working preferences. High quality individual nurse schedules are created by using a single or double shift swapping heuristic on a base schedule. These columns are then used to form a set covering problems which is solved by Branch-and-Bound. The authors find that, for most of the instances that the algorithm is tested on, the majority of the computational time is spent on generating columns rather than Branch-and-Bound. More recently, Bard and Purnomo propose a nurse rostering model which combines cyclic and preference scheduling in [95]. The problem is solved using Lagrangian relaxation and Branch-and-Price. Maenhout and Vanhoucke [96] present an exact Branch-and-Price algorithm for NRP incorporating different branching strategies.

As we can see from the publications discussed above, column generation is often used in OR approaches to NRPs. This is due to the feature of the NRPs. The columns in NRPs represent the possible work patterns for individual nurses. In the earlier publications, a restricted set of columns is predefined for assignment. More recently, the columns are generated by OR algorithms, such as shortest path algorithm, dynamic programming. And the heuristics can be integrated into the column generation procedure by modifying other columns via swapping assignments. More sophisticated methods such as Constraint Programming based column generation emerge more recently.

(2) Constraint Programming

Darmoni et al [97] use CP to solve the scheduling problems in a French hospital. The model is build based on Charme, a Constraint Programming language, consisting of a certain level of constraints to be satisfied. The solution procedure of the CP model consists of three main parts. Constraint propagation is first performed on each variable domain to deduce reduced domain. The smallest domain first heuristic is applied as the

variable selection heuristic in Charme. A search strategy trying to ensure fair scheduling among nurses is applied. The approach is able to produce satisfactory schedules over a planning horizon up to 6 weeks.

Weil et al [98] apply a CP solver: ILOG solver to solve a NRP with only a number of typical constraints such as minimum day off and consecutive shift pattern constraints. The authors present how to model the constraints in ILOG solver as a CSP. The system can find one feasible solution or all feasible solutions according to the request of the user. With respect to the soft constraints, the system can only provide the solution satisfying hard constraints and indicate if the solution violates soft constraints or not.

Cheng et al [99] present a CP method for solving a week long NRP in a Hong Kong hospital. A redundant modeling idea is described, which involves formulating the same problem in two distinct ways (shift to nurse and nurse to shift assignment). During the search, both formulations are simultaneously updated and fed back into each other. For each soft constraint, a branching decision is posted. One branch is to add the soft constraint to the model; the other branch is without the soft constraint. A final result to the problems is presented by the percentage of the satisfaction of soft constraints.

Metivier et al [100] propose a hybrid approach which emphasizes the application of soft global constraints. The interaction among the global constraints is investigated through the communication among them. The filtering algorithm can be more efficient when the constraints which share a common set of variables are considered together.

Wong and Chun [101] apply CP to solve the NRP with the help of meta-level reasoning and probability-based order heuristic. The meta-level reasoning is executed before the search to generate redundant or implied constraints from the existing constraints. These new constraints can help in further reducing the search space. Probability-based ordering is used as a value ordering heuristic. It approximates the probability of value assuagements occurring in the solution set and thus uses this information to guide the search.

(3) Hybrid approaches

Hofe [102] combines the ideas of heuristic local search with CP techniques to create an automated nurse rostering system tested in a German hospital. It models the problem as a Hierarchical Constraint Satisfaction Problem (HCSP) with fuzzy constraint. The constraints are organised into hierarchies of different priorities to reflect their importance. The fuzzy constraint allows a constraint to be partially satisfied and partially violated. The HCSP are solved by heuristics which are used to identify and repair violations.

Li et al. [76] present a hybrid AI approach to a class of over-constrained NRPs. Their approach has two phases. The first phase solves a relaxed version of the problem which only includes hard rules and part of the nurses' requests for shifts. It applies a forward checking algorithm with non-binary constraint propagation, variable ordering, random value ordering and compulsory back-jumping. In the second phase, the adjustments are made by descend local search and tabu search to improve the solution. The experiments show that the approach is able to solve this class of problem well.

Demassey et al. [44] investigate a CP based column generation approach which emphasizes the cost-filtering algorithms of optimisation constraint. The authors introduce a new optimisation constraint- *cost regular* for a global constraint- *regular*. The optimisation constraint links a cost to the decision variable assignments. Its filtering algorithm is based on the computation of the shortest and longest paths in a layered directed graph. The approach is applied to an employee timetabling problem where the columns are generated with the help of cost-regular constraint.

Sellmann et al. [69] develop two different algorithms to tackle the large-scale optimisation problem of airline crew assignment. The first one is an application of the CP based column generation framework. The second approach performs a CP based heuristic tree search.

The literature listed here includes general approaches to NRPs. More related work with each of the algorithms we proposed will be introduced in the following corresponding chapters.

3.3 The portfolio selection problem

Portfolio selection is one of the most relevant and studied topic in finance. The problem is primarily concerned with finding a combination of assets that satisfies an investor's needs the best. These needs can be basically expressed as minimizing the risk and guaranteeing a given level of returns. The foundation to portfolio selection as we know today is laid by Harry M. Markowitz by a quadratic optimisation model - mean–variance model (MV). The basic MV model selects the composition of assets which either achieves a predetermined level of expected return while minimizing the risk, or achieves the maximum expected return within a pre-defined level of risk.

Some aspects of the financial theory underlying the Markowitz's model and related models are out of the scope of our research. These financial theories include: the assumption about the independence of the investor's believes and the expected return and risk of assets, the assumption about the investor's utility functions etc. Our research focuses on the algorithm design and problem solving based on the Markowitz's model.

3.3.1 Modelling the portfolio selection problem

The basic model: Markowitz mean-variance model

Markowitz's mean-variance (MV) model [103] is concerned with a trade-off between the expected return and the risk. In this formulation, the risk of the portfolio is measured by the covariance among the selected assets. The MV formulation provides a fundamental basis for the modern portfolio selection theory in financial investment.

The Markowitz MV model is as follows:

$$\min \sum_{i=1}^{i=n} \sum_{j=1}^{j=n} \sigma_{ij} w_i w_j \tag{3-1}$$

$$s.t. \sum_{i=1}^{i=n} r_i w_i = R \tag{3-2}$$

$$\sum_{i=1}^{i=n} w_i = 1 \tag{3-3}$$

$$0 \le w_i \le 1, i = 1, ...n$$

Where n is number of assets $A = \{ a_1, \ldots, a_n \}$. Each asset a_i is associated with an expected return (per period) r_i, and each pair of assets $< a_i, a_j >$ has a covariance σ_{ij}. The covariance matrix $\sigma_{n \times n}$ is symmetric and each diagonal element σ_{ii} represents the variance of asset a_i, while the covariance σ_{ij} represents the correlated risks between pairs of assets. A positive value R represents the expected return.

To obtain the expected return, rational investors should pick a combination of diversified assets, i.e. a portfolio, to reduce the risk which is measured by the covariance of the combined portfolios. A portfolio can be represented by a set $W = \{ w_1, \ldots, w_n \}$, where w_i represents the percentage wealth invested on asset a_i. The value $\sum_{i=1}^{i=n} \sum_{j=1}^{j=n} \sigma_{ij} w_i w_j$ represents the variance of the portfolio, and is considered as the measure of the risk associated with the portfolio.

Variables and domains

In the basic MV model, the variables w_i are real and their domain is $0 \le w_i \le 1$, represents the percentage wealth invested on the asset.

In practice, there are a wide range of real-world trading constraints. These include the cardinality constraint (a limit on the total number of assets hold in the portfolio), the minimum position size constraint (bounds on the amount of each asset), the minimum trade size constraint (bounds on the amount of transaction occurred on each asset) and transaction costs, etc. When such constraints are considered and added to the basic MV

model, usually integer variables are needed. These will be investigated in chapters 7 and 8.

Objective functions

In the basic MV formulations, the objective can be either to minimize the risk (3-1) (satisfying a given return), or maximize the return (not exceeding a given maximum risk), or both. In the former cases the problem is single-criterion, while in the latter case it is multi-criteria. The problem can be modeled as a multi-objective problem with two conflicting objectives: minimize the variance, denoting the risk associated with the portfolio, whilst maximizing its profits. Essentially, the optimization problem is to find portfolios amongst the n assets that satisfy these two objectives simultaneously. An optimal portfolio is one that has the maximum return with the minimum risk and the set of all these optimal portfolios will form the **efficient frontier** illustrating the trade-off between the conflicting objectives, as represented by the line in Fig. 3.1. This efficient frontier will be used to evaluate the quality of solutions in chapter 7 and 8.

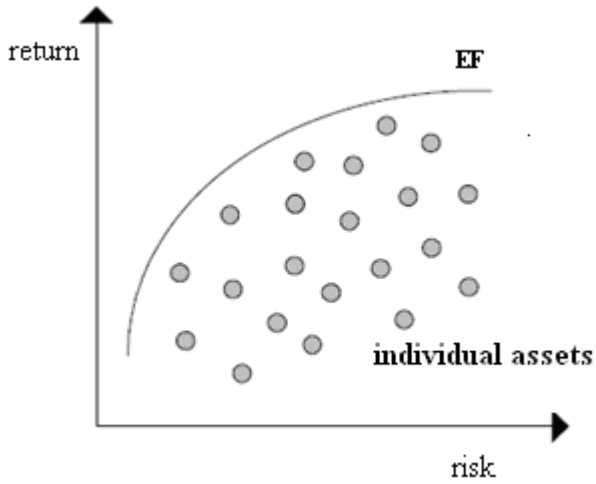


Fig. 3.1 Efficient Frontier (EF) which defines the trade-off between returns and risk in a portfolio of assets

In this thesis, we focus on the single-criterion problem. The applications on the single-objective formulation (in which the risk has to be minimized) very often solve a

portfolio selection problem instance with given expected return R. Solving the instance for R ranging over values from a finite set can give an estimation of the efficient frontier.

In the basic MV model, covariance is applied as risk measure in the objective function. Applying which term to measure the risk associated with the portfolio, to a certain extent, determines the complexity of the model built. Besides applying covariance as the risk measure of the portfolio, several other risk measures have been investigated in the literature and practice. In [104], the authors propose to use the mean absolute deviation as a measure of risk and formulate the first Linear Programming model for the problem. They show that the mean absolute deviation model, under the assumption of a normal distribution of the return, is equivalent to the quadratic MV model. Later on, in [105] the mean absolute semi-deviation is proposed instead of the mean absolute deviation as a risk measure to reduce the constraints in the mathematic model. More recently, some researchers focus on other risk measures such as value at risk and conditional worst expectation [106, 107].

Constraints

There are two constraints in the basic MV model: return (3-2) and budget (3-3) constraints. They are the most important constraints in portfolio selection problems, because they characterize the essential part of the problems. Return constraint (3-2) presents that the expected return should be met. Budget constraint (3-3) means that all the capital must be investigated.

As we stated before, in practice, there are wide range of real-world trading constraints. These include the cardinality constraint, the minimum position size constraint, the minimum trade size constraint and the transaction costs, etc. We illustrate all of the portfolio selection model attributes (variables, objectives and constraints) that will be investigated in this thesis in Fig. 3.2.

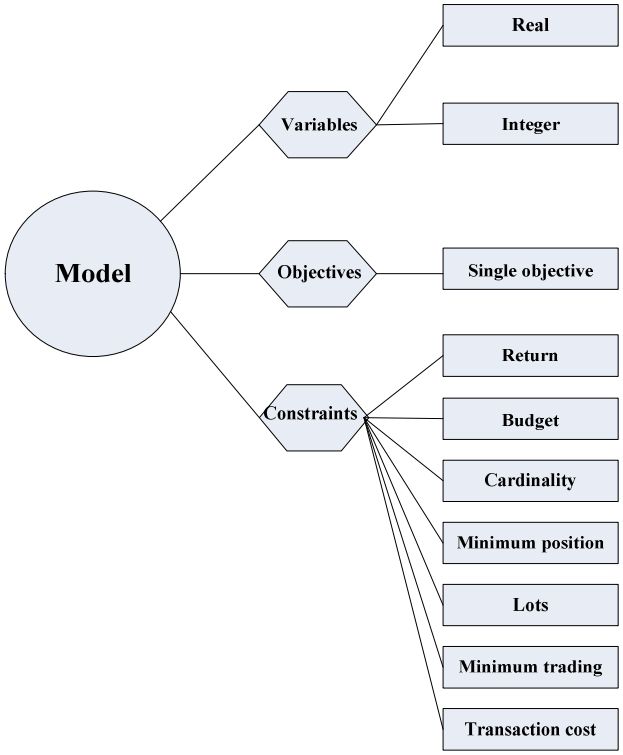


Fig. 3.2 Variables, objective and constraints of portfolio selection problems

3.3.2 Solution approaches to portfolio selection problems

In literature, different techniques and approaches to PSPs can be generally categorised into three groups according to which and how the techniques are applied: (1) heuristics, (2) local search hybridised with exact methods in a sequential manner, and (3) exact method as the main body with the assistance of heuristics.

Heuristic (or meta-heuristic) methods have been applied to constrained PSPs, especially for those of large-scale. In [108], a simulated annealing algorithm is proposed to solve the MV model with additional constraints. Moving operators (direction of moves and amplitude of moves) are designed with domain knowledge to deal with the different constraints. For constraints that must be strictly satisfied, an "all-feasible" approach is applied to enforce the satisfaction of the constraint and forbid the generation of any

solutions violating the constraint. The other constraints are handled by a "penalty" approach which adds a penalty for each violated constraint in the objective function.

In [109], the authors highlight the different shapes of the constrained efficient frontiers compared with that of unconstrained problem, and show that certain portions of the efficient frontier are disconnected. What's more, three heuristic algorithms, which are genetic algorithms, tabu search and simulated annealing, are used to plot the efficient frontier.

In [110], different neighbourhood relations (i.e. structures) such as idR(increase, decrease, Replace), idID(increase, decrease, Insert, Delete) and TID(Transfer, Insert, Delete) are devised to define the quantity of the move in a tabu search algorithm. The results are improved compared with those in [109].

In another group of approaches, exact methods are hybridised with local search algorithms. For example, in [111], local search is used as the master solver to select the assets to be included in the portfolio, and quadratic programming is used as the slave solver to minimize the risk (variance).

In [106], an exact Branch-and-Bound approach is proposed based on a heuristic partition of the initial problem into two subproblems, and a simple local search is used to construct the initial solution.

Both of the above two groups of approaches, i.e. pure local search techniques [108-110] and hybrid approaches where the local search serves the main role [106, 111], cannot guarantee or provide a measure of the solution quality. We do not know how far we are from the optimal solution(s). To achieve a measurable solution quality in a reasonable computational time, exact methods with heuristics are applied to the constrained PSPs. In [112] and [113], exact B&B solution approaches are proposed for the problem subject to the buy-in threshold constraint, lots constraint and cardinality constraint. However, these constraints are considered separately which leads to three independent

models. In [114], an exact solution approach is proposed to the PSP under stochastic and integer constraints. A static branching rule and a dynamic branching rule are proposed for Branch-and-Bound in order to obtain the optimal solution, where these constraints are also considered separately.

Chapter 4 Hybrid CP with Variable Neighbourhood Search approach to nurse rostering problems

4.1 Introduction

This chapter and the following chapters 5 and 6 present three integration methods to our first application problem – the nurse rostering problem.

The research starts from the property of the NRPs. A large set of constraints (i.e. working regulations and personnel preferences) are present in the NRPs. Some of them are logic constraints and very complex. This key property of the problem makes CP techniques a good option to model the complex constraints and solve the problem.

What is more, the real-world NRPs we are tackling are over-constrained. That is, there is no feasible solution if all of the constraints must be satisfied. Therefore, soft constraints are applied to model the conflicting preferences of nurses. We seek an (optimal) feasible solution that minimizes the violation of soft constraints.

As introduced in section 1.1, in this chapter, we decompose the problem according to the constraints. That is, we first consider certain set of constraints only. A feasible solution is generated by CP techniques with respect to this set of constraints first, and then the rest of constraints are handled by a second stage *local search*. Therefore, this hybrid method can be represented by a two-stage framework "feasible solution + improvement".

Under this framework, a feasible initial solution is first constructed directly by solving a CSP model which consists of all of hard constraints. However, this initial solution cannot be efficiently improved by the second stage local search shown by our experiments. We propose a ***sequence based initial solution generation*** method to construct feasible initial solutions. The basic idea of this approach is based on the

observations that high quality nurse rosters consist of high quality shift sequences. Therefore, at the first stage, constraint satisfaction model is used to generate weekly rosters that consist of high quality shift sequences satisfying a subset of constraints. An iterative forward search is then adapted to extend them to build the complete feasible solutions. Variable and value selection heuristics are employed to improve the efficiency. At the second stage, a simple Variable Neighourhood Search is used to quickly improve the feasible solution obtained. By decomposing the problems into solvable subproblems for CP, the search space of the original problems is significantly reduced. Thus the feasible solutions are generated efficiently by CP while the optimisation of the feasible solutions relies on the second stage local search.

4.2 Problem description

Here, we first describe one of the benchmark NRPs we will tackle in this thesis. The benchmark NRP (named ORTEC) we are tackling are derived from real-world problems in intensive care units at a Dutch hospital. The problem consists of assigning a predefined number of shifts of four types (i.e. early, day, late and night shifts) within a scheduling period of 5 weeks to 16 nurses of different working contracts in a ward. Twelve of the full-time nurses work 36 hours per week. One and other three part-time nurses work maximally 32 and 20 hours per week, respectively. The problems have a number of variants with respect to the number of nurses, the number of shift types, the number of skill levels and the length of scheduling period, etc., but the main constraints are similar. We define the main problem here and test a number of its variants and several other problems in the experiments. More details can be found in [115] and at http://www.cs.nott.ac.uk/~tec/NRP/. Table 4.1 presents the definitions and the daily coverage demand of the four shift types in the problems.

Table 4.1 Shift types and demand during a week. Each shift covers 9 hours including one hour of resting time, except that the night shift contains no resting time. So there are 8 actual working hours for each of these shift types.

shift type	Start time	End time	Demand						
			Mon	Tue	Wed	Thu	Fri	Sat	Sun
Early	07:00	16:00	3	3	3	3	3	2	2
Day	08:00	17:00	3	3	3	3	3	2	2
Late	14:00	23:00	3	3	3	3	3	2	2
Night	23:00	07:00	1	1	1	1	1	1	1

We present a summary of the constraints in ORTEC in Table 4.2, followed by the explanations of them. Details of all constraints in different problems we are concerned with in this thesis are listed in Appendix.

Table 4.2 Summary of constraints in the benchmark nurse rostering problems, more details in Appendix.

Hard constraints	Only one shift on the same day for each nurse
	Exact coverage requirement (no over/under cover)
	Working time
	Shift patterns
Soft constraints	Workload balance
	Pattern preferences
	Shift patterns

Constraint

Hard constraints (denoted by H)

H1 Demand needs to be fulfilled (i.e. all the requested shifts in Table 4.1 must be covered).

H2 For each day, one nurse can only be assigned to one shift.

H3 Within a scheduling period, a nurse is allowed to exceed the number of hours for which he/she is available for his/her department by at most 4 hours.

H4 The maximum working time per week is on average 36 hours over a period of 13 consecutive weeks.

H5 The maximum number of night shifts is 3 per period of 5 consecutive weeks.

H6 A nurse must receive at least 2 weekends off duty per 5 week period. A weekend off duty lasts 60 hours including Saturday 00:00 to Monday 04:00.

H7 The length of a series of consecutive night shifts is at least 2. Following them, a 42 hours rest is required.

H8 The number of consecutive night shifts is at most 3.

H9 The number of consecutive shifts (workdays) is at most 6.

	Soft constraints (denoted by S)	**Weight**
S1	For the period of Friday 23:00 to Monday 0:00, a nurse should have either no shifts or at least 2 shifts (Complete Weekend).	1000
S2	Avoid sequence of shifts with length of 1 for all nurses.	1000
S3	For all nurses, the length of a series of *night* shifts should be within the range [2, 3]. It could be part of, but not before, another sequence of shifts.	1000
S4	The rest after a series of *day*, *early* or *late* shifts is at least 2 days.	100
S5a	For nurses with availability of 30-36 hours per week, the number of shifts is within the range [4, 5] per week.	10
S5b	For nurses with availability of 0-30 hours per week, the number of shifts is within the range [2, 3] per week.	10
S6a	For nurses with availability of 30-36 hours per week, the length of a series of shifts should be within the range of [4, 6].	10
S6b	For nurses with availability of 0-30 hours per week, the length of a series of shifts should be within the range [2, 3].	10
S7	For all nurse, the length of a series of *early* shifts should be within the range [2, 3]. It could be within another series of shifts.	10
S8	For all nurse the length of a series of *late* shifts should be within the range of [2, 3]. It could be within another series of shifts.	10
S9a	An *early* shift after a *day* shift should be avoided.	5
S9b	An *early* shift after a *late* shift should be avoided.	5
S9c	A *day* shift after a *late* shift should be avoided.	5
S10	A *night* shift after an *early* shift should be avoided.	1

These constraints are extracted from the descriptions of the working regulations, rules and personal preferences from head nurses and administrations of the hospital. These constraints are categorised into two groups: hard constraints and soft constraints, as defined below:

Hard constraints must be satisfied in order to obtain the feasible solutions for use in practice. A roster satisfying all hard constraints is usually termed as feasible.

Soft constraints are not obligatory but are desired to be satisfied as much as possible. In real life, a roster which satisfies all hard and soft constraints usually does not exist. The violations of soft constraints in the roster can thus be used to evaluate the quality of the solutions. Common soft constraints in NRPs aim to generate rosters with a balanced workload so that human resources are used efficiently.

The categories of hard and soft constraints, indicated above by S and H, are given by ORTEC, as well as the weights of the soft constraints. Actually, the boundary between the hard and soft constraints is vague in different real-world NRPs. In this thesis we apply the benchmark data set to test our algorithms.

The objective of NRPs can be defined as to find a feasible roster (i.e. which satisfy all hard constraints) with the lowest possible penalty caused by soft constraint violations, i.e. to minimize a weighted sum of the penalties from all violations of soft constraints.

4.3 CP approach to NRPs

As we reviewed in chapter 3, the solution approaches from different disciplines to NRPs have been intensively investigated. Due to the presence of the large set of constraints (i.e. working regulations and personnel preferences), NRPs are computationally challenging. This key property of the problem makes CP techniques a good option to solve the problem due to the following facts:

- CP has the strength of modelling the problem with the global constraints as introduced in chapter 2. Several global constraints, such as cardinality, and sequence, etc. capture the structure of the NRPs well.
- Propagation algorithms of these global constraints make them efficient to find feasible solutions to the problem.
- Most importantly, the solution approach of CP consists of modelling, propagation and searching and it can be integrated with other techniques.

Therefore, in this chapter we investigate a hybrid CP approach to NRP where CP plays a key role in the solution procedure.

In real nurse rostering settings, we noticed that the problems are nearly always over-constrained. Some of the early research works reviewed in chapter 3 are effective in solving small scale problems with fewer constraints, but are not flexible to deal with large-scale problems with more complex constraints. We start our investigation by applying the CP techniques to NRPs and testing pure CP's ability of solving the large-scale, over-constrained NRPs.

In this section, we first present how to model the constraints in NRPs within the CP paradigm by applying the global constraint that can capture the structure of NRPs. Then we model the problem as a COP and solve it by pure CP. The aim of this pure CP approach is two-fold: firstly, we can test CP's ability of handle the problem at hand. That is, how many constraints can by handled by pure CP techniques. Secondly, based on the observation of this pure CP approach, we propose a pre-processing that can be applied as an initialization heuristic to generate initial solution which will be applied in chapters 5 and 6.

4.3.1 Modelling the constraints

We first present notions that will be applied in the models as follows:

--N: set of nurses (index i)

--D: set of days in the scheduling period (index j)
--S: set of shift types, i.e. *Early, Day*, *Late*, *Night*, *Off*.

For each nurse i and each day j in the scheduling period, we define an assignment variable s_{ij} that indicates which shift is assigned to nurse i on day j. This decision variable has finite domain, i.e. $D(s_{ij}) = S$.

In addition to the assignment variables, we define some auxiliary variables in order to implement the constraints of the problems. These auxiliary variables will be introduced and explained in the context of constraint representation and implementation.

CP is a very flexible technique to model a rich set of constraints due to its powerful declarative ability. In the most simple and straightforward way, we could define the constraints in NRPs by using primitive constraints, e.g. use "if s_{ij} = late, $s_{ij+1} \neq$ early" to express that no early shift is allowed after a late shift.

Global constraint in NRPs

Global constraint is a substitute of a set of primitive constraints and is usually equipped with efficient propagation algorithms to remove inconsistent values from variables' domains. A list of global constraints with propagation algorithms have been presented for different application domains in [116]. In this work we use the global constraints and the soft versions of some global constraints to model some of the constraints in our problems. NRP is a nice application domain for CP because it showcases several of the global constraints developed over the years. We first review these global constraints that are needed, establishing a notation and concentrating on their implementation and filtering capability (in IBM ILOG Solver as we introduced in chapter 2 section 2.6). We then go back to the constraints listed in section 4.2, and model the NRPs we are tackling by using these global constraints.

Sum constraint [page 445 of [5]] considers a set of variables $\boldsymbol{x} = (x_1, \ldots, x_n)$; the constraint Sum($\boldsymbol{x}$) returns the sum of the values of variables.

Global **Cardinality** constraint (*gcc*) is also named as *distribute* (see [5] pages 420). It bounds the number of times of certain values being taken by variables. It is written as follows:

$$cardinality(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{l}, \boldsymbol{u})$$

where $\boldsymbol{x}$ is a set of variables (x_1, …, x_n); $\boldsymbol{v}$ is a m-tuple of domain values of the variables $\boldsymbol{x}$; $\boldsymbol{l}$ and $\boldsymbol{u}$ are m-tuples of nonnegative integers defining the lower and upper bounds of the times value $\boldsymbol{v}$ being taken by variable $\boldsymbol{x}$, respectively. The constraint defines that, for j = 1, …, m, at least l_j and at most u_j of the variables x take value v_j.For example, the constraint that a nurse i should work at most 3 night shifts in the whole period $j=1…n$ can be expressed as $cardinality(x_{ij}$, Night, 0, 3), $j=1…n$.

Stretch constraint (see [5] pages 444) is written as follows:

$$stretch(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{P})$$

where $\boldsymbol{x}$ is a set of variables (x_1, …, x_n); $\boldsymbol{v}$ is a m-tuple of possible domain values of $\boldsymbol{x}$; $\boldsymbol{l}$ and $\boldsymbol{u}$ are m-tuples of lower and upper bounds for $\boldsymbol{x}$, respectively. $\boldsymbol{P}$ is a set of patterns, i.e. pairs of values (v_j, v_j‘), requiring that when a stretch of value v_j immediately precedes a stretch of value v_j‘, the pair (v_j, v_j‘) must be in $\boldsymbol{P}$.

A *stretch* is a sequence of *consecutive* variables that take the same value, i.e., $x_{j-1} \neq v$, x_j, …, $x_k = v$ and $x_{k+1} \neq v$. This constraint also restricts that any stretch of value v_j in $\boldsymbol{x}$, should a length within the range [l_j, u_j]. Thus the *stretch* constraint puts bounds on how many *consecutive* days a nurse can work on each shift, and which shifts can immediately follow another. The constraint can omit the restriction on pattern $\boldsymbol{P}$ which represents no restriction on the pattern.

For example, the following constraint

$$stretch(s_{ij}, Night, 2, 3, P), P = \{(Night, Off)\}, j=1…n$$

restricts a nurse having consecutive night shifts within the length [2, 3], and the only shift type allowed following the night shift is off (as given in *P*).

Sequence constraint is written as follows:

$$sequence(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{w})$$

where $\boldsymbol{x}$ is a set of variables (x_1, …, x_n); $\boldsymbol{v}$ is a m-tuple of possible domain values of $\boldsymbol{x}$; $\boldsymbol{l}$ and $\boldsymbol{u}$ are m-tuples of lower and upper bounds for $\boldsymbol{x}$, respectively. This constraint also restricts that any sequence of value v_j in $\boldsymbol{x}$ with a length of $\boldsymbol{w}$, the length should be within the range [l_j, u_j].

The difference between **sequence** and **stretch** is that the **stretch** constraint counts *consecutive* variables of a certain value, while **sequence** does not have this restriction.

Table 4.3 summarises the global constraints we applied in modelling and solving the NRPs, including their corresponding implementation in IBM ILOG Solver. We list the level of consistency that is claimed to be achieved by the Solver. However, we do not necessary to achieve these consistency levels because this may lead to a very expensive running time. We made a balance between the level of consistency and the execution time spends on it.

As we introduced in section 2.6, the ILOG Solver system provides a wide range of low level, primitive constraints along with propagation algorithms for these constraints. The CP Solver system also provides the mechanism for combining the primitive constraints. In this state-of-art of CP Solver system, the propagation for the combining of primitive constraints is efficient by certain extension of constraint propagation on primitive constraints [34]. Therefore, in this thesis, the stretch constraint is implemented based on two constraints: IloSequence and IloIfThen, where IloSequence is used to restrict assignments of a certain value and IloIfThen is used to identify the consecutive variables.

Table 4.3 Summary of the global constraints in modelling nurse rostering problems

Global constraint	Implementation in ILOG solver	Level of consistency	Filtering algorithm
Sum	IloSum	Bound consistency	AC [4]
Cardinality	IloDistribution	General arc consistency	Regin [16]
Sequence	IloSequence	General arc consistency	Regin [117]
Stretch	IloSequence &IloIfThen	General arc consistency	Regin [16]

Soft global constraint

The *crisp* (hard) version of global constraints defines the CSP, i.e. a tuple of values is either allowed or not allowed. When we deal with the preferences in the problem (soft constraint in our NRPs), *soft* constraints are applied, where the objective is to minimize the violation of these soft constraints by using the associated violation measure. Therefore, the global constraint has to be extended to handle the constraint violations. We use ~ hereinafter to denote the soft version of the constraint.

To define violation measure of ~*gcc(**x, d, l, u**),* we introduce a "shortage" function *s* and an "excess" function *e* [29] for each domain value to measure the negative and positive deviation with respect to the lower bound *l* and upper bound *u*:

$$s(x,v)=\begin{cases} l-|x=v|, if\ |x=v|\le l \\ 0, otherwise \end{cases}$$

$$e(x,v)=\begin{cases} |x=v|-u, if\ |x=v|\ge u \\ 0, otherwise \end{cases}$$

where $|x=v|$ denotes the times that variable *x* takes value *v*.

Then the violation measure for ~*gcc(**x,d,l,u**)* is defined as follows:

$$\mu^{\sim gcc}=\sum_{v\in D} s(x,v)+e(x,v)$$

For example, if nurse i prefers to work on day shifts within range [4, 5] per week, represented by ~$gcc(s_{ij}$, Day, 4, 5), j=1…7 , and a schedule below or over this range leads to a penalty of weight 100, then for the schedule of a week l = [Day, Day, Day, Off, Off, Off, Off] for nurse i, a penalty can be calculated as $w\mu^{\sim gcc}(l) = 100 \times (4 - 3) = 100$, where $\left|s_{ij} = Day\right| = 3$, $lower = 4$ and $upper = 5$ and the violation measure for ~$gcc(s_{ij}$, Day, 4, 5), j=1…7 is 1.

In our model of NRPs, we implement the ~stretch constraint with auxiliary variables and a mapping which have been successfully applied in [102, 118], to measure the violations of constraints. For a given sequence l, a variable is used to indicate the starting point of the shift concerned. The shortage function $s(x, v)$ and excess function $e(x, v)$ as applied above in ~*gcc* are used to measure the violations of ~*stretch*. Although this implementation leads to an increased number of variables, the resulting constraints are linear and easy to solve in the CP system.

4.3.2 CP approach to NRPs

Based on the hard and soft version of global constraints we introduced above, we model the ORTEC problem as a COP which consists of all of the constraints listed in section 4.2.

Model (Pure CP Complete COP)

Decision variable: $s_{ij}, D(s_{ij}) = S, i \in N, j = \{1,2,3,4,5,6,7.....\}$

D_{jk}: coverage demand of shift type k on day j, $j = \{1,2,3,4,5,6,7.....\}, k \in S$, given in Table 4.1.

Objective:

$$Minimize \sum_{C_e \in C} w_{C_e} \mu^{C_e}$$

where μ^{C_e} is violation measure for soft constraint C_e. w_{Ce} is the weight of soft constraint C_e. It subject to the following constraints:

H1 Coverage constraint. A number of different shifts must be covered throughout the scheduling period in order to guarantee the coverage of service. This constraint is modeled as $gcc(s_{ij}, S, D_{jk}, D_{jk}),\ i \in N, j = \{1,2,3,4,5,6,7.....\}, k \in S$

H2 For each day, one nurse can only be assigned to one shift. This constraint is implicitly satisfied by assigning exactly one value to each constrained variable.

H3 Within a scheduling period, a nurse is allowed to exceed at most 4 hours more than his/her available working time. Each shift has 8 hours working time. This constraint is modeled as $sum\ (8 \times f_{ij}) \leq h_m + 4\ ,j=1\ldots n,$ where $f_{ij} = \begin{cases} 1, if\ \ s_{ij} \neq off \\ 0, otherwise \end{cases}$. h_m is the available working hours for a nurse of category m in the scheduling period.

H4 Maximum 36 hours working time per week. This constraint is modeled as $sum(8 \times f_{ij}) \leq 36, j=\{1\ldots.7\ldots\}$

H5 Maximum 3 night shifts in the scheduling period. This constraint is modeled as $gcc(s_{ij}$, Night, 0, 3), $j=\{1\ldots.7\ldots\}$. This constraint applies to the whole scheduling period, but it also restricts weekly scheduling.

H6 At least 2 weekends off in the scheduling period. This constraint is modeled as $gcc(s_{ij}$, Off, 2, *5*), in conjunction with a If-Then constraint: if s_{ij}=off, then s_{ij+1}=off, *j=6,13,20,27,34*

H7,H8 The length of a series of consecutive night shifts is at least 2. Following them, a 42 hours rest is required. At most 3 consecutive night shifts in the scheduling period. These two constraints are modeled as a single constraint $stretch(s_{ij}$, Night, 2, 3, $P)$, P = {(Night,Off)}, $j=\{1\ldots.7\ldots\}$

H9 At most 6 consecutive working days. This constraint is modeled as $stretch(s_{ij}$, ~Off, 1, 6), $j=\{1\ldots.7\ldots\}$. Here ~off represents not off shift, and P is omitted that represents no restriction on the pattern.

S1 Complete weekend. From Friday 23:00 to Monday 0:00, a nurse should have either no shifts or 2 shifts. The violation measure of this soft constraint is

$$\mu_{ij}^{S1} = \begin{cases} 1, if\ \ s_{ij} \neq s_{ij+1} \\ 0, otherwise \end{cases} \text{ and } u_i(S1) = \sum_{j=6,13,20,27} u_{ij}$$

S2 Avoid a sequence of shifts of length 1 for all nurses. The violation measure of this

soft constraint is $\mu_{ij}^{S2} = \begin{cases} 1, if\ \ s_{ij} = off, s_{ij+1} \neq off, s_{ij+2} = off \\ 0, otherwise \end{cases}$ and *j=1…n-3.*

S3 For all nurses, a series of night shifts should be within [2, 3]. It could be part of, but not before, another sequence of shifts. This constraint is implicitly satisfied by constraint H7 and H8.

S4 At least 2 days off after a series of day, early or late shifts. This is modeled as ~*stretch* (s_{ij}, Off, 2, 5), *j=1…n* and the violation measure is calculated as introduced before.

S5 For full time nurses, the number of working shifts should be within [4, 5] per week. This is modeled as ~*gcc*(s_{ij}, ~Off, 4, 5), *j=1…7, j=8….14…*for the corresponded week. ~Off represents a day-on. For part time nurses, the number of labor shifts should be within [2, 3] per week. This is modeled as ~*gcc*(s_{ij}, ~Off, 2, 3), *j=1…7, j=8….14…*for the corresponding week.

S6 For full time nurses, the length of a series of shifts should be within [4, 6]. This can be modeled as ~*stretch*(s_{ij}, ~Off, 4, 6), *j=1…n.* For part time nurses, the length of a series of shifts should be within [2, 3]. This can be modeled as ~*stretch*(s_{ij}, ~Off, 2, 3), *j=1…n*

S7 For all nurses, the length of a series of early shifts should be within [2, 3]. This is modeled as ~*stretch*(s_{ij}, Early, 2, 3), *j=1…n*

S8 For all nurses, the length of a series of late shifts should be within [2, 3]. This is modeled as ~*stretch*(s_{ij}, Late, 2, 3), *j=1…n*

S9 An early shift after a day shift should be avoided. The violation measure is $\mu_{ij}(S9) = \begin{cases} 1, if\ \ s_{ij} = Day, s_{ij+1} = Early \\ 0, otherwise \end{cases}$ and *j=1…n.* An early shift after a late shift should be avoided. A day shift after a late shift should be avoided. This can be modeled in a similar way.

S10 A night shift after an early shift should be avoided. This can be modeled in a similar way as S9.

We do not implement any special designed propagation algorithms for the soft constraints. This model is solved by pure CP techniques. The experimental results are presented in section 4.5.

From the experiments, it was observed that finding even just feasible solutions with respect to *all hard and soft constraints* for the large-scale problem ORTEC is very time consuming (see Table 4.6). Therefore, we design a heuristic relaxation approach to construct feasible solutions to the problem.

As defined above, a violation measure can be used to model soft constraints in over-constrained problems, where no feasible solution exist if all constraints have to be satisfied. However, propagation upon these soft constraints, compared with that on hard constraints, is not very efficient [119]. Specific propagation algorithms have been designed in literature. In our work, we do not rely on the designing of these specific propagation algorithms. An indirect constraint relaxation method is applied to obtain initial solutions as shown in Fig. 4.1. Firstly soft constraints are sorted increasingly according to their weights (importance). Then start by treating all soft constraints crisply (which will obviously lead to no feasible solutions), we relax the soft constraint with the least weight step by step until a feasible solution can be found. This method has been shown to be very fast and efficient due to the powerful propagation in CP to find feasible solutions.

This initial solution heuristic will be applied in chapters 5 and 6.

Algorithm. Initial solution generation
1: Sort the soft constraints increasingly according to their weights;
2: Add the sorted soft constraints and all the hard constraints into constraint set C;
3: Solve the problem $P(X, D, C)$ as a constraint satisfaction problem by CP;
If $P(X, D, C)$ = infeasible, then relax the soft constraint c_i with the least weight, remove c_i from C; go to step 3;
Else return the feasible solution.

Fig. 4.1 Initial solution generation

4.4 Problem decomposition and hybrid CP approach to NRPs

As we introduced in section 1.1, we will decompose the NRP according to the constraint of the problem. That is, we are first concerned with certain selected set of constraints only. Feasible solution is generated by CP techniques with respect to this set

of constraints first. Then the rest of constraints are tackled by a second stage *local search* method. Therefore, this hybrid method can be represented by a two-stage framework "feasible solution + improvement".

4.4.1 Problem decomposition

Decomposition is one option to deal with large-scale problems with complex constraints. Decomposition techniques have been investigated recently in NRPs. Since most of the NRPs are over-constrained and difficult to solve directly, decomposing the original problem into subproblems which are easier to solve is well motivated. There are several ways of decomposition in the nurse rostering literature: (1) Decomposition by constraints: construct solutions only subject to a subset of constraints of the problem. Based on the solutions obtained, further adjustment or improvement is made to satisfy the rest of the constraints. (2) Decomposition by variables: The roster for the whole ward of nurses consists of the schedule for each nurse in the ward. So the personnel schedule is first generated subject to all related constraints for each nurse. Then these schedules are combined to construct the whole roster. The boundary between these two types of decomposition is usually vague because of the nature of the NRPs. For example, when the problem is decomposed by variables, i.e. to generate schedule for each nurse, only the schedule related constraints are included (subset of constraints of the whole problem), so it can also be seen as decomposed by the constraints.

The problem ORTEC we are solving has a very large search space, for which a systematic tree search is computationally expensive and cannot provide a solution even after one day's computation which is shown by computational experiments in section 4.5 (see Table 4.6) by solving the Model (Pure CP Complete COP). We thus investigate a two-stage hybrid CP approach:

- Stage I: Initial solution construction by CP.
- Stage II: A Variable Neighourhood Search is then used to improve the solution built from stage I.

We first present how to construct initial feasible solution under this two-stage framework.

When we construct the initial feasible solutions, we can apply the initial solution heuristic proposed in Fig. 4.1. This is a straightforward way to apply CP to construct a feasible initial solution subject to certain set of constraints. Therefore, we name this method as "direct" initial solution construction. Another more "indirect" initial solution construction method is also investigated which is named as sequence based initial solution construction.

Stage I: Initial solution construction

4.4.2 Direct initial solution construction

To apply this direct initial solution construction, we model the problem as a CSP problem in Model (Direct Initial CSP) with all of the constraints identified by initial solution generation heuristic presented in Fig. 4.1. In this Model (Direct Initial CSP), all of the constraints can be formulated in a similar way as shown in Model (Pure CP Complete COP). The initial feasible solution then is improved by the second stage local search.

From the computational experimental results presented in Fig. 4.6 of section 4.5, we can see that the violation of certain high weight soft constraints cannot be eliminated by the local adjustment of the local search. The final solution after the local search still has violation of high weight soft constraint which is presented by the high objective value. The similar observation has also been identified by several researches in [102, 120, 121]. Therefore, we propose another initial solution construction method to obtain better quality initial solutions.

4.4.3 Sequence based initial solution construction

The underlying idea of this initial solution construction approach is based on some common features of high quality rosters - they consist of high quality shift sequences satisfying a set of constraints in the problems. Therefore, we generate these high quality shift sequences first. Based on them, the initial feasible solution can be constructed.

In this work, we categorise the constraints into two groups: *sequence* and *schedule* constraints, which are considered separately at different steps of the first stage of the hybrid CP approach. In the first step, only sequence constraints are considered in the CSP model Model (Sequence based Initial CSP) to generate weekly rosters with high quality shift sequences. In the second step, both sequence and schedule constraints are included in another COP model Model (Sequence based Initial COP) to extend the weekly rosters to build the complete roster. The two groups of constraints are described as follows:

- *Sequence* constraints are applied when generating shift sequences for each nurse within weekly rosters, and
- *Schedule* constraints are applied when the weekly rosters are extended to the complete rosters for all nurses.

Two groups of constraints are listed as follows:

	Type		**Type**
H1	Both*	S1	Sequence
H2	Both*	S2	Sequence
H3	Schedule	S3	Sequence
H4	Sequence	S4	Sequence
H5	Both*	S5	Sequence
H6	Schedule	S6	Schedule
H7	Both*	S7	Schedule
H8	Both*	S8	Schedule
H9	Both*	S9	Both*
		S10	Both*

We also define the following terms that are frequently used in the rest of the chapter:

- *Shift sequence* is the sequence of shifts assigned to each nurse within weekly rosters;
- *Weekly roster* is the one week roster consists of shift sequences for all nurses;
- *Roster* is the complete assignment of shifts within the scheduling period to all nurses, i.e. the complete solution to the problem.

We decompose the problems into weekly subproblems, and then extend the weekly rosters obtained to complete solutions. Two CP models are thus defined, where different variables and their corresponding domains are given with respect to shift sequences in weekly rosters and complete solutions.

The first model is Model (Sequence based Initial CSP) (subjects to a subset of constraints). It models the decomposed problem which is concerned with weekly rosters. It should be noted that some of the sequence constraints are soft constraints in the problem (i.e. S1, S2, S3, S4, S5, S9, and S10). Since we try to build weekly roster as a CSP model (In a CSP model, constraints are strictly satisfied or not. It cannot deal with soft constraints), some of the sequence constraints in soft constraints (i.e. S5, S9 and S10) are relaxed (not include in the Model (Sequence based Initial CSP)). Some of the soft sequence constraints with high weights are restricted (they are modeled as hard constraints in the Model (Sequence based Initial CSP)), such as S1, S2, S3 and S4. The reason why we use this approach is that we try to consider as many constraints as possible in the Model (Sequence based Initial CSP) model. An initial test showed that restricting high weight soft constraints did not prevent the generation of feasible solutions. If it did, we cannot model these soft constraints as hard ones.

Model (Sequence based Initial CSP)

Decision variable: $s_{ij}, D(s_{ij}) = S, i \in N, j = \{1,2,3,4,5,6,7\}$

D_{jk}: coverage demand of shift type k on day j, $j = \{1,2,3,4,5,6,7\}, k \in S$, given in Table 4.1.

The sequence constraints which we are concerned with in Model (Sequence based Initial CSP) is modeled as following:

H1 Coverage constraint. A number of different shifts must be covered throughout the scheduling period in order to guarantee the coverage of service. This constraint is modeled as $gcc(s_{ij}, S, D_{jk}, D_{jk}),\ i \in N, j = \{1,2,3,4,5,6,7\}, k \in S$

H2 For each day, one nurse can only start one shift. This constraint is implicitly satisfied by assigning exactly one value to each constrained variable.

H4 Maximum 36 hours working time per week. This constraint is modeled as $sum(8 \times f_{ij}) \leq 36, j=\{1....7\}$

H5 Maximum 3 night shifts in the scheduling period. This constraint is modeled as $gcc(s_{ij}$, Night, 0, 3), $j=\{1....7\}$. This constraint applies to the whole scheduling period, but it also restricts weekly scheduling.

H7,H8 The length of a series of consecutive night shifts is at least 2. Following them, a 42 hours rest is required. At most 3 consecutive night shifts in the scheduling period. These two constraints are modeled as a single constraint $stretch(s_{ij}$, Night, 2, 3, $P)$, $P = \{(\text{Night,Off})\}, j=\{1....7\}$

H9 At most 6 consecutive working days. This constraint is modeled as $stretch(s_{ij}$, ~Off, 1, 6), $j=\{1....7\}$. Here ~off represents not off shift, and P is omitted that represents no restriction on the pattern.

S1(H) Complete weekend. From Friday 23:00 to Monday 0:00, a nurse should have either no shifts or 2 shifts. This can be modeled as hard constraint: $s_{ij} = s_{ij+1}$, i=6

S2(H) Avoid a sequence of shifts of length 1 for all nurses. This can be modeled as hard constraint: if s_{ij} =Off and s_{ij+1} =~Off, then and s_{ij+2} =~Off, $j=\{1....5\}$.

S3(H) For all nurses, a series of night shifts should be within [2, 3]. It could be part of, but not before, another sequence of shifts. This constraint is implicitly satisfied by constraint H7 and H8.

S4(H) At least 2 days off after a series of day, early or late shifts. This can be modeled as hard constraint: $stretch\ (s_{ij}$, Off, 2, 5), $j=\{1....7\}$.

As stated above, the soft sequence constraints with high weights are restricted (they are modeled as hard constraints in the Model (Sequence based Initial CSP)), such as S1, S2, S3 and S4, denoted by (H) above.

Model (Sequence based Initial COP)

Decision variable s_{iw}: represents the shift sequence of one week length assigned to nurse i in week w. The domain of variables is the permutations of the shift sequences generated by the first model, i.e. {(*0011444*), (*4400022*), …}. The model is presented as follows:

Objective:

$$Minimize \sum_{C_e \in C} w_{C_e} \mu^{C_e}$$

where μ^{C_e} is violation measure for soft constraint C_e. w_{Ce} is the weight of soft constraint C_e. It subject to additional constraints:

H3 Within a scheduling period, a nurse is allowed to exceed at most 4 hours more than his/her available working time. Each shift has 8 hours working time. This constraint is modeled as $sum\ (8\times f_{ij}) \le h_m+4$,$j=1\ldots n$, where $f_{ij} = \begin{cases} 1, if\ \ s_{ij} \neq off \\ 0, otherwise \end{cases}$. h_m is the available working hours for a nurse of category m in the scheduling period.

H6 At least 2 weekends off in the scheduling period. This constraint is modeled as *gcc*(s_{ij}, Off, 2, *5*), in conjunction with a If-Then constraint: if s_{ij}=off, then s_{ij+1}=off, *j=6,13,20,27,34*

S5 For full time nurses, the number of labor shifts should be within [4, 5] per week. This is modeled as ~*gcc*(s_{ij}, ~Off, 4, 5), *j=1…7, j=8….14*…for the corresponded week. ~Off represents a day-on. For part time nurses, the number of labor shifts should be within [2, 3] per week. This is modeled as ~*gcc*(s_{ij}, ~Off, 2, 3), *j=1…7, j=8….14*…for the corresponding week.

S6 For full time nurses, the length of a series of shifts should be within [4, 6]. This can be modeled as ~*stretch*(s_{ij}, ~Off, 4, 6), *j=1…n* and the violation measure is calculated as introduced before. For part time nurses, the length of a series of shifts should be within [2, 3]. This can be modeled as ~*stretch*(s_{ij}, ~Off, 2, 3), *j=1…n*

S7 For all nurses, the length of a series of early shifts should be within [2, 3]. This is modeled as ~*stretch*(s_{ij}, Early, 2, 3), *j=1…n*

S8 For all nurses, the length of a series of late shifts should be within [2, 3]. This is

modeled as ~*stretch*(s_{ij}, Late, 2, 3), *j=1…n*

S9 An early shift after a day shift should be avoided. The violation measure is $\mu_{ij}(S9) = \begin{cases} 1, if\ \ s_{ij} = Day, s_{ij+1} = Early \\ 0, otherwise \end{cases}$ and *j=1…n*. An early shift after a late shift should be avoided. A day shift after a late shift should be avoided. This can be modeled in a similar way.

S10 A night shift after an early shift should be avoided. This can be modeled in a similar way as S9.

Here we only present the Model (Sequence based Initial COP) with soft constraints (denoted by ~) but without the implementation and optimisation of soft constraints. The optimizing of the soft constraints in the Model (Sequence based Initial COP) is fulfilled by a Variable Neighourhood Search which will be detailed later.

Weekly roster construction

Weekly rosters which consist of high quality shift sequences are firstly generated by Model (Sequence based Initial CSP). The algorithm used is a systematic backtracking Depth First Search. The *first-fail* principle is used as the variable order heuristic. One illustrative example of weekly roster generated by Model (Sequence based Initial CSP) is given in Table 4.4. These shift sequences for each nurse satisfy all the sequence constraints in Model (Sequence based Initial CSP), so they are of high quality and are desired to be preserved in the final complete solution. By using Model (Sequence based Initial CSP), thousands of weekly rosters can be generated in seconds (8.7E5 approximately, see experiments in section 4.5). We randomly select 50 initial weekly rosters to build the complete solutions by using the iterative forward search.

Table 4.4 An illustrative example of weekly (partial) roster.

	Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**	**Sun**
Nurse 1	O	O	D	D	N	N	N
Nurse 2	N	N	O	O	O	E	E
…	…						

Roster construction by Iterative Forward Search

Iterative forward search [122] works upon feasible incomplete solution (weekly rosters generated by the above step). It iteratively extends these blocks into a complete solution. Fig. 4.2 presents the pseudo code of the search algorithm.

The algorithm extends the current partial solutions by assigning values to variables until all the variables have been assigned values. If it succeeds, the one-week roster will be extended to a two-week roster and we continue in the same way. The number of outside iterations corresponds to the number of weeks in the whole roster (4 iterations to build 5 weeks' roster in the problem here). The inside iterations of the procedure assign values to the variables iteratively. When a conflict occurs after a value has been assigned to a variable, the latest variable is un-assigned and another value is tried (backtracking). If all the values have been tried and the search cannot continue consistently, the search starts from the outside iteration and attempts another set of initial weekly roster blocks (for example, another 50 initial weekly rosters will be chosen randomly) to continue.

```
Procedure IFS (initial weekly roster block i = 1)
  outside iteration repeat
    iteration = 0;
    current solution = initial weekly roster i;
    inside iteration repeat
      select variable and value;    //with or without heuristic selection
      assign value to variable;
      current solution = initial weekly roster i + assigned variable;
      un-assign conflict variable;
    until(allWeeklyVariableAssigned)
    if(canContinue(initial weekly roster i))
      iteration = iteration + 1;
    else
      initial weekly roster block i = i + 1;
  until(allVariableAssigned)
  complete solution = current solution
end procedure
```

Fig. 4.2 Pseudo-code of the iterative forward search algorithm

The above algorithm is parameterized by two heuristics, variable selection and value selection heuristics. In this work we compare these two heuristics with a random rule and evaluate their effects within our hybrid CP approach:

1. Randomly select variables and values during the search
2. Select variables and values by following heuristics:
   a) Variable selection heuristic: first-fail principle, by which the nurses with heavier workload from previous iteration are selected first;
   b) Value selection heuristic: night shift sequences first.

The variable selection heuristic chooses the next variable in the search based on the information collected in the previous iterations of the search. The shift sequences assigned to each nurse are recorded and the nurses are ranked by their workloads. The heavier workload the nurses have received, the more likely a conflict will occur later with respect to the workload constraint. Therefore we follow the first-fail principle to consider the heavier workload nurses first in the next step of the search.

The night shift is the most important shift in the problems, due to the fact that it is involved in a number of hard constraints (H5, H7, and H9) and soft constraints (S2, S3) with high weights of 1000. Therefore we assign night shift sequences first. The rest of the sequences are of the same importance and are randomly selected and assigned to the nurses.

Stage II: Variable Neighourhood Search

4.4.4 Second stage local search

A simple Variable Neighbourhood Descent is applied to improve the solution built from Stage I. Two neighbourhood structures are employed in the algorithm; both have been widely used in meta-heuristics in the nurse rostering literature [115]. Note that this work is mainly a hybrid CP approach rather than designs elaborated meta-heuristics. The two neighbourhoods are defined by the following moves upon a complete roster (illustrated in Fig. 4.3):

- Neighbourhood structure 1: re-assign a shift to a different nurse working on the same day.
- Neighbourhood structure 2: swap shifts assigned to two nurses on the same day.

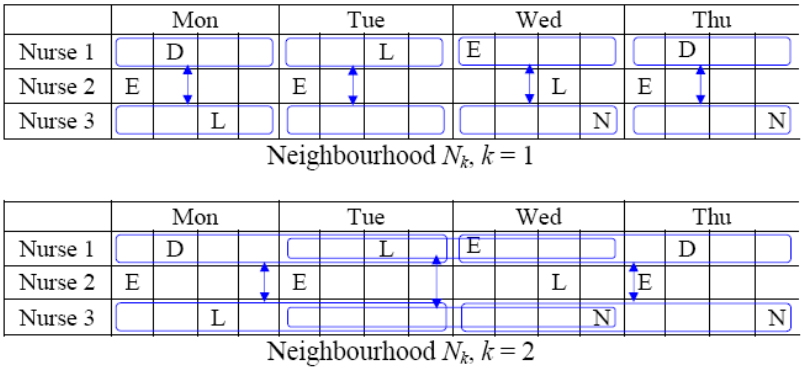


Fig. 4.3 Two neighbourhood structures. A small part of the scheduling period is shown. An arrow denotes a possible move in the neighbourhood [123].

The pseudo-code of the Variable Neighourhood Search is presented in Fig. 4.4. The neighbourhoods (by the smaller neighbourhood structure 1) are repeatedly examined for possible improving moves. When there are no improving moves by using neighbourhood structure 1, neighbourhoods by larger neighbourhood structure 2 are examined. Then the search switches back to neighbourhood structure 1 again. This process is repeated until there are no improving moves left by using both neighbourhood structures 1 and 2.

Initialization select neighbourhood structures N_k, $k = 1,2...k_{max}$;
construct an initial solution x;
Repeat until no improvement is obtained:
(1) Select $k = 1$;
(2) Repeat the following steps until $k = k_{max}$:
(a) Explore to find the best neighbour x' of x ($x' \in N_k(x)$);
(b) Move or not. If the solution thus obtained x' is better than x, set $x = x'$ and $k = 1$; otherwise, set $k = k + 1$;

Fig. 4.4 Pseudo-code of the Variable Neighourhood Search algorithm [124]

The Variable Neighourhood Search searches upon the feasible solutions built from the first stage. The feasibility of the solutions is preserved during the search by considering all the constraints in the problem.

4.5 Experimental results

We evaluate our hybrid CP approach upon a set of benchmark NRPs instances, publicly available at http://www.cs.nott.ac.uk/~tec/NRP, where a range of problems collected from industry and scientific publications are presented. These chosen benchmarks have been the mostly tested problems in the literature due to their complex constraints. The rules, regulations and objectives have been directly taken from the real-world cases and preserved with the essential characteristics, see Table 4.5.

It is important to note that the difficulty of the problems not only depends on the number of shift types, the number of nurses and the length of the scheduling period, but also on the complex constraints involved (see all constraints in Appendix). In Table 4.5, the largest problem ORTEC presents to be the most difficult, where 12 instances (of 12 months) have been widely tested by a number of approaches in literature. The other two simpler problems, i.e. Gpost and Valouxis, although highly constrained, are of relatively smaller size. Instances A, B, and C are variants of Gpost with relaxation on some constraints. Instances ORTEC#1 to #4 are variants of ORTEC with relaxation on some constraints. They are used to tune our CP search and provide insight of the effects of different components. The same set of problems, i.e. Gpost, Valouxis, and ORTEC will be tested in chapters 5 and 6 to evaluate other proposed hybrid approaches.

For all problems, 6 runs are carried out on an Intel(R) Core(TM) 2CPU 1.86GHz machine with 1.97GB memory, from which average results are presented.

Table 4.5 Characteristics of the benchmark nurse rostering problems. Instances A, B, and C are variants of Gpost with relaxation on some constraints. Instances ORTEC#1 to #4 are variants of ORTEC with relaxation on some constraints.

	Number of Shift types	Number of Nurses	Period of Schedule(day)	Number of Skill Levels
A	2	8	7	1
B	2	8	28	2
C	2	8	28	2
Gpost	2	8	28	1
Valouxis	3	16	28	1
ORTEC#1-#4	4	16	35	1
ORTEC#Jan-#Dec	4	16	35	1

Experiment I. Pure CP and Hybrid CP Approaches.

We first evaluate the hybrid CP approach compared to the pure CP approach to the benchmark problems presented in Table 4.5. Here the so-called pure CP approach uses a complete COP model Model (Pure CP Complete COP) in which all hard and soft constraints are included to solve this set of problems of the original size without decomposition. The depth-first Branch-and-Bound search is used as the search algorithm. Table 4.6 presents the results and demonstrates their abilities to handle constraints in different problems. The column “problem size” in the table gives the number of variables and the number of constraints in the CP model. It is observed that the pure CP approach can only handle small scale instances (measured by the number of variables and constraints) but cannot produce solutions for large-scale instances even after 24 hours running. The hybrid CP approach can obtain results for all these large-scale instances within 1 hour.

Table 4.6 Results of pure CP and hybrid CP approaches to nurse rostering problems of different characteristics. "-" indicates that no solutions can be obtained within 24 hours.

Data	Problem Size		Pure CP	Hybrid CP
	Variables	Constraints	(within 1 hour)	(within 1 hour)
A	722	2109	8	8
B	3460	4600	0	0
C	3639	4612	10	10
Gpost	7897	5866	5	5
Valouxis	8321	9867	-	120
ORTEC#1	6672	22380	-	616
ORTEC#2	8208	28562	-	786
ORTEC#3	8624	29108	-	650
ORTEC#4	8720	29234	-	616

Experiment II. Variable and Value Selection in the Hybrid CP Approach.

Another set of experiments is carried out to evaluate the effect of variable and value selection heuristics in the CP search upon problem instances presented in Table 4.5. It is observed that random selection rule can easily cause a large number of violations to the high weight penalty constraints, mainly due to the bad assignments of night shifts. The solutions produced by using this rule cannot be further improved in the second stage. Table 4.7 presents the results of the CP search by using different variable and value selection rules. Both of them can obtain results within 1 hour.

Table 4.7 Results with random and heuristic variable and value selection rules in the hybrid CP approach. Mean values of 6 running results are presented.

Problem	Random Selection	Heuristic Selection
Gpost	18	8
Valouxis	160	80
ORTEC#1	1686	616
ORTEC#2	1035	786
ORTEC#3	635	650
ORTEC#4	705	616

Table 4.8 presents the evaluation of six basic variable ordering heuristics in the Solver for problem Gpost. The number of choice points and fails encountered during the search indicates that the MinSizeInt and MinMaxInt heuristics perform the best, with no statistically significant differences. The MinSizeInt heuristic is randomly picked and used in the following CP search procedures.

Table 4.8 Evaluation of six variable ordering heuristics for problem Gpost

Heuristics	No. of choice points	No. of fails	CPU (sec)	Variable ordering strategies
MinSizeInt	8966	7995	1.3	the smallest domain first
MaxSizeInt	10706	9723	1.5	the largest domain first
MinMinInt	10703	9720	1.5	the least minimal bound first
MaxMinInt	11978	10995	1.8	the greatest minimal bound first
MinMaxInt	9290	8319	1.2	the least maximal bound first
MaxMaxInt	126003	11620	1.8	the greatest maximal bound first

The value ordering heuristic we applied is night shift first. The night shift is the most important and complicated shift in the problems, due to the fact that it is involved in a number of hard constraints (H5, H7, and H9) and soft constraints (S2, S3) with high costs of 1000. Therefore we assign night shift first. The rest of the shifts are of the same importance and are randomly selected and assigned to the nurses.

Experiment III. The Hybrid CP Approach on Large-scale Benchmarks.

According to the results in Table 4.6, we can see a pure CP model for the entire problem cannot produce good solutions if there is a realistic runtime restriction. In addition, the basic VNS alone is not applicable as it cannot produce feasible solutions for all data instances.

The behaviour of hybrid CP approach is illustrated in Fig. 4.5 on the ORTEC January and February instances. The initial feasible solutions with cost 1639 on the January instance and cost 5361 on February are generated by CSP and iterative forward search. Fig. 4.5 depicts the improvement of the solution cost for the hybrid CP approach on the January and February instances. Although the values differ among other various instances, the characteristic shapes of the curves are similar.

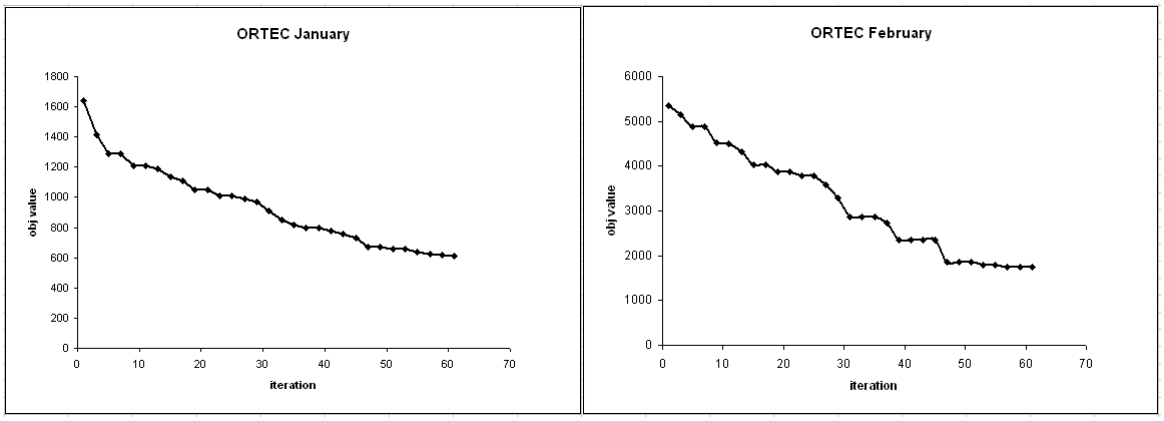


Fig. 4.5 Behaviour of the hybrid CP approach on the ORTEC January and February instances

Initial solutions of different quality have been tested in order to investigate the influences of the quality of initial solution to the quality of final solution as shown in Fig. 4.6(based on the ORTEC January instance).

Four initial solutions with different quality, i.e. objective value, are tested. Initial solutions #1, #2 and #3 are generated by our iteration forward search based on the sequence generated by Model (Sequence based Initial CSP). Initial solution #4 is a randomly selected solution generated by Model (Direct Initial CSP). These initial solutions are fed into the second stage VNS with the same computational time. The

behaviour is illustrated in Fig. 4.6. It can be seen that initial solution #1, #2, #3 generated by iterative forward search based on Model (Sequence based Initial CSP) can be improved by VNS step by step. The violation of soft constraints can be eliminated by the local adjustment of VNS. For the randomly selected initial solution #4 generated by Model (Direct Initial CSP), the violation of the hard constraint cannot be eliminated by the local adjustment of VNS. The final solution after VNS still has violation of hard constraint which is represented by the high objective value.

There are several possible reasons to explain this phenomenon. Firstly, the nurses in the same category (i.e. full time or part time) have same constraints and preferences. Therefore, there is symmetry between the generated lines of schedules of two nurses in the same category. Because of this symmetry, in VNS, swap of single shift between two nurses in the same category may not make any improvement. Secondly, swap of single and two shifts between two nurses is inefficient. More sophisticated neighbourhood structure is needed. This is also observed by other researchers. In [102, 120, 121], the authors highlight the disadvantages of some of the basic local search algorithms which change one variable assignment at a time. The effectiveness of simultaneously making multiple value assignment changes is showed. This leads to our investigation of how to improve the second stage local search in following chapter 5.

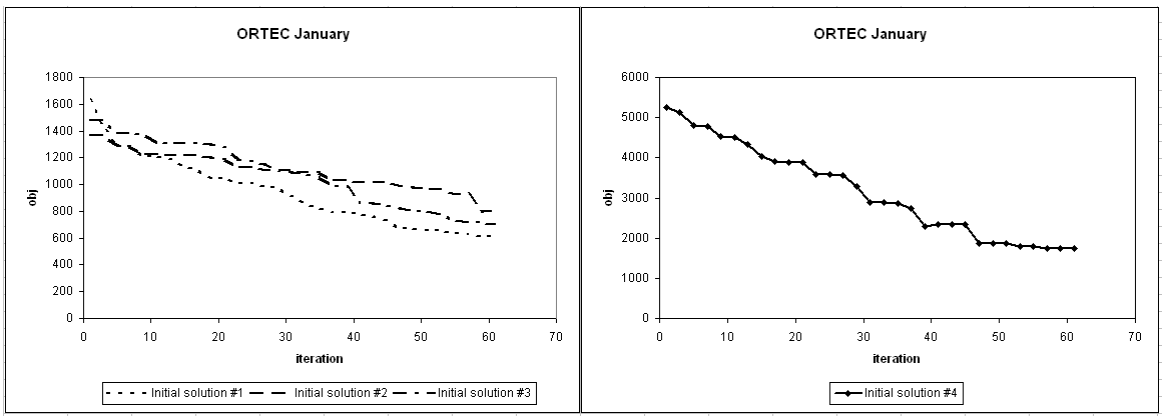


Fig. 4.6 Behaviour of the hybrid CP approach on the ORTEC January instances with different initial solutions

Table 4.9 Results from the hybrid CP approach, compared to current approaches in the literature, best results are in bold.

Problem instances	Hybrid GA [125]	Hybrid VNS [115]	Hybrid IP[123]	Hybrid CP approach
ORTEC#Jan-#Dec	(1 hour)	(1 hour)	(1hour)	(½ hour)
Jan	775	735	**460**	616
Feb	1791	1866	**1526**	1736
Mar	2030	2010	**1713**	2766
Apr	612	457	**391**	956
May	2296	2161	2090	**1786**
Jun	9466	9291	8826	**8700**
Jul	781	481	**425**	650
Aug	4850	4880	3488	**2171**
Sep	615	647	**330**	1300
Oct	736	665	**445**	616
Nov	2126	2030	**1613**	1620
Dec	625	520	**405**	496

Table 4.9 presents the results from the hybrid CP approach compared to those from other current approaches on twelve large real-world NRP instances (ORTEC#Jan-#Dec). The first approach is a hybrid genetic algorithm which has been developed by ORTEC, Netherlands in the commercialised software HarmonyTM [125]. The second approach is a hybrid Variable Neighourhood Search with a heuristic ordering as the construction method [115]. Hybrid IP [123] is a method which applies IP model to construct initial solution and a Variable Neighourhood Search to make improvement to them. The meta-heuristic algorithms (e.g. genetic algorithms and Variable Neighourhood Search) have been delicately designed using the domain knowledge to solve the problem. This domain knowledge has been applied in both the designing of initial solutions and

delicate neighbourhood structures. Our hybrid CP with VNS mainly relies on CP, while only a simple VNS is used to improve the solutions obtained by CP.

In our hybrid approach, CP in the first stage generates weekly rosters in a short time (on average of 370 seconds, depending on the number of constraints in the model). These blocks are the permutations of high quality shift sequences. The iterative forward search procedure with Model (Sequence based Initial COP) terminates when a complete solution is found. Then the simple Variable Neighourhood Search obtains the improved solution within 1 minute. The overall process takes up to 30 minutes. Hybrid IP [123] performed best among all of the approaches with a longer computational time (i.e. 1 hour), compared with our approach. The Hybrid IP method spent most time on solving the IP model intensely which explains its better performance. Within a much shorter computational time, our hybrid CP approach obtained the best results for 3 out of 12 problems compared to the current best approaches in the literature. This result is satisfactory since our hybrid CP with VNS only applies a very simple neighbourhood structure.

We have also test the performance of our hybrid CP approach with longer running time either by extending the number of initial solutions or allowing extra running time in Stage II for improvement. It is observed that the extra number of initial solutions has no impact upon the final solution mainly because all the selected initial solutions are of similar quality as shown in Fig. 4.6. The Variable Neighourhood Search usually improves the initial solutions within minutes; it did not show significant improvement in longer running times. What is more, based on the conclusion drawn on Fig. 4.6, in VNS, swaps of single and two shifts between two nurses are inefficient. More sophisticated neighbourhood structure is needed. It is a disadvantage that the basic neighbourhood structure changes only one or two variable assignments at a time. This motivates us to make multiple value assignment changes. We will investigate how to improve the second stage local search in following chapter 5.

4.6 Conclusions

In this chapter, we model and solve the nurse rostering problems by a hybrid CP approach. The work has been published at Applications and Innovations in Intelligent Systems XVI, see List of Publications.

Several fundamental elements in CP approach to the nurse rostering problem are investigated. A model is built with primitive and global constraints. This pure CP approach is firstly tested on small scale instances and it can provide solutions within 1 hour. For the large-scale instances, this pure CP approach cannot provide solution in 1 hour due to the complex constraints. Therefore, a decomposition and hybrid approach is proposed. The decomposition is based on some common features of high quality rosters - they consist of high quality shift sequences satisfying a set of constraints in the problems. The feasible solution subject to only a subset of constraints is firstly generated by solving the corresponding CSP model. Then the complete feasible solution is constructed using an iterative forward search. The further improvement of the feasible solution is gained using a second stage local search method.

The experimental results demonstrated that global constraints can model the complex regulations in NRPs well. For the global constraints applied in NRPs, efficient propagation algorithms associated with them in IBM ILOG Solver enable the efficient feasible solution generation. However, in this hybrid approach, the second stage local search is a rather simple VNS. Two simple neighbourhood structures are applied in the VNS. Our experimental results motivate the further investigation of the second stage local search in the following chapter 5.

Chapter 5 Constraint-directed Large Neighbourhood Search to nurse rostering problems

5.1 Introduction

In chapter 4, we decompose the problem by constraints and design the first hybrid method under the framework "initial solution +improvement". In chapter 4, the effort is focus on the construction of feasible solution by Constraint Programming techniques in the hybrid method.

In this chapter, the local search in the second stage is further enhanced by using the information of constraints. We start the research from the identification of potential issues in local search that can be improved while solving the NRPs.

A local search algorithm typically starts from an initial solution (an assignment of values to all the decision variables) and iteratively moves to neighbouring solutions, defined by neighbourhood operator(s), with the hope of improving the quality of the solution measured by a function *f*. Function *f* measures the quality of solutions to the problem at hand with regard to different requirements and constraints.

A wide range of research issues have been addressed with the aim of achieving efficiency of local search algorithms. Among these we are focusing on three of them in this work: (1) Neighbourhoods: neighbourhoods are potential successor states of the incumbent solution in the search. (2) Function *f*: the function *f* measures the quality of solutions to the problem. The values of the function are usually used to direct the search to better states. (3) Feasibility: one of the critical issues in local search is to consider the feasibility while reasoning the optimality of the solution [81, 126], especially for those highly constrained real-world combinatorial optimisation problems.

Chapter 5 Constraint-directed Large Neighbourhood Search to nurse rostering problems

The most basic and common neighbourhood structures in nurse rostering problems are single shift neighbourhood and block neighbourhood. These basic neighbourhood structures have also been applied in our second stage VNS in chapter 4. Existing research shows the inefficiency of them for large and complex nurse rostering problems [102, 120]. This is also observed through our experiments in chapter 4.

For highly constrained and large-scale nurse rostering problems, very large-scale neighbourhood search techniques [85] have been successfully applied to the problems. Dowsland [127] shows that the chain neighbourhoods (i.e. a sequence of on/off day swaps between nurses) are able to lead the search to escape from local optima that single on/off day swaps cannot escape from. Louw et al. [120] use an ejection chain approach where compound move is applied instead of single move. Burke et al. [120] defines several heuristics to identify chains of swaps of on/off day between nurses in their work of variable depth search and the experiment results show its efficiency. However, these methods, especially the design of neighbourhood structures are tailored to the problem instances at hand.

All of these local search techniques have shown their efficiency to solve large and constrained nurse rostering problems. However, there is a scope of improvement with respect to the three research issues mentioned above for the following reasons: (1) For large and constrained nurse rostering problems, it is quite common that highly sophisticated and problem-tailored neighbourhoods are needed, such as the chain of moves in [120]. However, extensive expertise is needed to design dedicated neighbourhood operators, especially for the problems having various constraints [85]. (2)Value of function f quite often provides a very limited guidance of the unknown search space. It neglects other information which may be useful during the search (such as the satisfaction of constraints [128]). (3) At each move of the local search, solutions need to be checked to preserve feasibility (or neighbourhood operators need to be defined so that only feasible neighbours can be generated) [126].

Chapter 5 Constraint-directed Large Neighbourhood Search to nurse rostering problems

Using the information of constraint to direct local search presents an interesting attempt to make the search procedure more general and informative. For example, optimal solution of Linear Programming relaxations are often exploited to repair integral infeasible solutions [57]. Another successful example is the usage of *global constraint* in the CP paradigm for local search algorithms [128]. The global constraints can support local search approaches from two aspects: (1) Using constraints to define the neighbourhoods. For example, in [80], constraints are used to define the neighbourhood so that only feasible neighbours can be generated. (2) Applying the violation measure of the constraint as evaluation function to direct the search. In the local search approach for constraint satisfaction [81], the conflicts of an assignment are used to direct the local search to move to the assignments with less conflicts.

In chapter 4, a hybrid CP approach is applied to the large and constrained nurse rostering problems, where a meta-heuristic algorithm - VNS is applied to improve the initial solution obtained by the first stage CP search. In this chapter, we still take the advantages of CP's feasibility reasoning which has been proved in chapters 4. What is more, the constraint itself is utilised in the procedure of local search in a more close and interplayed manner, so that the proposed approach can benefit from both CP's feasibility reasoning and local search's efficiency.

In this chapter, we develop a constraint-direct large neighbourhood search approach which integrates the constraint into Large Neighbourhood Search algorithms with respect to the three research issues mentioned above. (1) For the problem at hand, we define *general* neighbourhood structures by *constraint*. That is, instead of setting tailored neighbourhood operators, we use the violations of constraints to detect the fragment (variables) of the solution which needs to be improved. This (usually) large neighbourhood area is re-optimised by using the search in CP. (2) In local search, for an unconstrained optimisation problem, a function *f* can easily express the objective of the problem. However, for highly constrained problems, function *f* needs to be modified to properly measure the infeasibility of the candidate solutions. We distinguish the *objective function* and *evaluation function* in this work. Each global constraint has its

corresponding evaluation function and it is used to measure the feasibility or optimality (in terms of violation measure) of an assignment. Therefore, there are more than one evaluation functions, so that more information can be utilised to guide the search. The *objective function* aggregates the violations of constraints into a single summed value. (3) With CP we can restrict the search to the feasible solution space.

5.2 Literature review on global constraints applied to local search

The information about global constraint can be utilised in local search in different ways. As introduced in chapter 2, a global constraint is defined on a set of lower-level primitive constraints, so it presents some features that cannot be presented by individual primitive constraints. This means using global constraint for local search allows us to revise a current state of search on a more global level [128]. A simple example is that, in a CSP with global constraint AllDifferent, we can associate the constraint AllDifferent a cost, which depends on the variables' assignments. The satisfaction of the constraint AllDifferent can be transformed as minimization of the cost. A value of zero for the cost means satisfaction. The constraint AllDifferent knows how to reduce this cost in local search by the operations associated with the constraint (such as domain reduction, and achieving arc consistency, etc.).

Several related work in the literature have been investigated to use constraint information during the search procedures.

In [128], the global constraints are used to exploit the domain-specific information in dynamic job shop scheduling problems by inducing two constraint-specific search controls. The first constraint-specific search control is based on the global *Resource* constraint. It uses the cost of the *Resource* constraint to guide the search to reassign the values to variables. The second constraint-specific search control is based on *Task* constraint; the domain inconsistency information of the constraint guides the search to reassign the values. Experiment results show that, with the inconsistency information of

global *Resource* constraint and *Task* constraint, the local search has better understanding of the search space (the cost distribution of each constraint is illustrated) and moves to good solutions much faster.

In [129], the authors propose to generate more general and automatic neighbourhood in local search, instead of designing trivial neighbourhood tailored to the problem. The neighbourhood is defined automatically by the volume of *propagation* of a constraint. This idea provides a different perspective view of local search by guiding the search based on the properties (i.e. consistency of the constraints) of the current solution to the problem.

In [130], the neighbourhood is viewed from a constraint perspective, as opposed to a variable perspective which is often the case. To construct a neighbourhood from a variable perspective, we usually start from a set of variables and apply changes to one or more of these variables, while evaluating the effect of these changes to the objective function of the problem. From a constraint perspective, the neighbourhood is obtained from a set of constraints. The authors exploit the structure of the constraints and based on that, decreasing, preserving and increasing neighbourhoods are designed for the constraints.

In the new architecture of constraint solver Comet [81], the constraint not only serves a natural tool to *express* the problems, but also plays a novel role in the *search*. The constraints maintain a number of properties incrementally and they provide algorithms to evaluate the effect of various operations (i.e. value reassignment, and swap values for two variables, etc.) on these properties. Typical properties include the violation degree of a constraint and the set of variables violating the constraint. In this architecture, the search is driven by these properties to feasible solution or optimal solution in the search space.

In this proposed work, we integrate CP with local search within a Large Neighbourhood Search (LNS) scheme. A pre-processing procedure is performed to identify hard

constraints from soft ones of the nurse rostering problems according to the feasibility reasoning by the CP solver. Then we apply a two-stage approach to the problem. In the first stage, an initial solution is constructed by solving the Model (Direct Initial CSP) given in chapter 4. In the second stage, LNS is implemented on the problem considering soft constraints (i.e. Constraint Optimisation Problem). The key feature of the second stage is that we also use the information of constraints to direct the local search as shown in the referred work above [81, 128-130]. The difference to the above work is that, instead of implementing an ad-hoc search/propagation algorithm for each global constraint, we only use the properties of the constraints to identify the fragment (set of variables) need to be optimised. Then the conventional tree search in CP is used to re-optimise the selected area. By doing this, we can take the advantage of the powerful feasibility reasoning of the CP solver and efficient search ability of the local search at the same time. Our proposed approach is easy to implement and maintain, and benefits from both CP and local search.

5.3 Modelling nurse rostering problems

We implement a pre-processing approach which utilises the feasibility reasoning of the CP solver. This procedure is illustrated by Fig. 4.1 where it is used to generate initial solution. The feasible solution obtained from this pre-processing is served as the initial solution for the local search.

How to model the problem with global constraints and soft global constraints has been described in detail in chapter 4.

5.4 Constraint-directed Large Neighbourhood Search

5.4.1 The framework

As we stated in the introduction, it is important to distinguish the *objective function* from the *evaluate function* in the local search. The former represents the objective to be

optimised to the problems, while the latter represents the function guiding the search process over the search space. There can be more than one evaluate function to guide the search process over the search space. In most meta-heuristic algorithms in nurse rostering problems, the objective function is used as the evaluation function. But when it combines feasibility and optimality measures, using individual evaluate function can give better guiding of the search toward promising solutions [81, 128].

In this work, we use costs of global constraints as the evaluation function in local search. The evaluation function is defined as the violation of each global constraint. The objective function is to minimize the total sum of evaluation functions. The information of constraints such as the cost and violation are used as indicators to select the neighbourhood (the set of variables) and implement the move (re-optimise).

LNS is firstly proposed by Shaw [80] to solve the vehicle routing problem. The basic idea of LNS is to iteratively relax (destruct) and then re-optimise (reconstruct) a part of the solution, with the hope to find better solutions over iterations. CP is used to generate the new assignment for this relaxed part of variables and add bound to the search to ensure that the new solution found is better than the current one.

Fig. 5.1 presents the two-stage approach we employ which embeds CP into LNS to solve nurse rostering problems. In the first stage, a feasible initial solution is constructed based on Model (Direct Initial CSP), i.e. with only hard constraints. In the second stage, LNS is used to improve the initial solution iteratively considering the cost of soft constraints. The LNS is parameterized with different strategies to choose the fragment which represents the low quality part (poor assignment of variables) of the solution to be re-optimised to obtain improved solutions iteratively.

```
Stage 1: construct initial solution: solve (Model (Direct Initial CSP))  // Model is presented in chapter 4;
Stage 2:
While  stopping criteria not met do
  Choose the low quality fragment to be relaxed with strategy i //test the strategies one by one;
  Freeze the remaining variables // fix the variables which are outside the fragment to their current values;
  Re-optimize the fragment using CP by solving Model (Pure CP Complete COP) // Model is presented in chapter 4;
      If found improved solution
          Update solution
      End if
End while
```

Fig. 5.1 The large neighbourhood search scheme with CP as the re-optimiser

The success of the LNS depends on two main factors: (1) The identification of the fragment of appropriate size with regard to the crucial part of the solution; (2) The search effort needed to optimise this fragment. The crucial part of the solution (assignment to subset of variables) can be indicated by the violation of the global constraints put on these variables. So it is straightforward that we reassign values to the variables that violate constraints involved according to their current inconsistency. After the fragment is selected, Branch-and-Bound search in CP is applied to re-optimise the selected fragment.

5.4.2 Fragment selection strategies

As observed by several researchers, the key issue in designing efficient LNS is the selection of the fragment, i.e. set of variables to be relaxed and re-optimised. For instance, in job shop scheduling problems, the fragments usually include the critical path of the schedules [131]. In routing problems, cluster removing techniques have been used [132]. To a certain extent, the success of LNS depends on the adequacy of this fragment with regard to the problematic parts of the solution.

The roster (solution) we construct for nurse rostering problems has a 2-Dimensional row/column structure as introduced in section 3.2.1. Each row represents the schedule for a nurse and each column represents a day assignment in the scheduling period. The constraints in the model can thus be categorised as row/horizontal constraints and column/vertical constraints. In our problem, there is only one hard constraint, coverage

constraint, which can be seen as a column constraint. All other constraints which are related with shift patterns and preferences can be seen as row constraints. This 2-Dimensional structure of rosters determines the basic structure of the fragment selected.

Strategy 1: sliding window

The first fragment selection strategy selects all nurses on certain length of days. We denote the size of the fragment as s, $s=n\times l$ where n is the total number of nurses and l is the number of days selected in the fragment. Fig. 5.2 presents a simple example of this strategy. The n is set to 8 which is the total number of nurses and l is set to 7. This fragment looks like a time window on all of the nurses. We slide this window one week by another along the whole schedule period to select different fragment of the problem. So we name this strategy as sliding window.

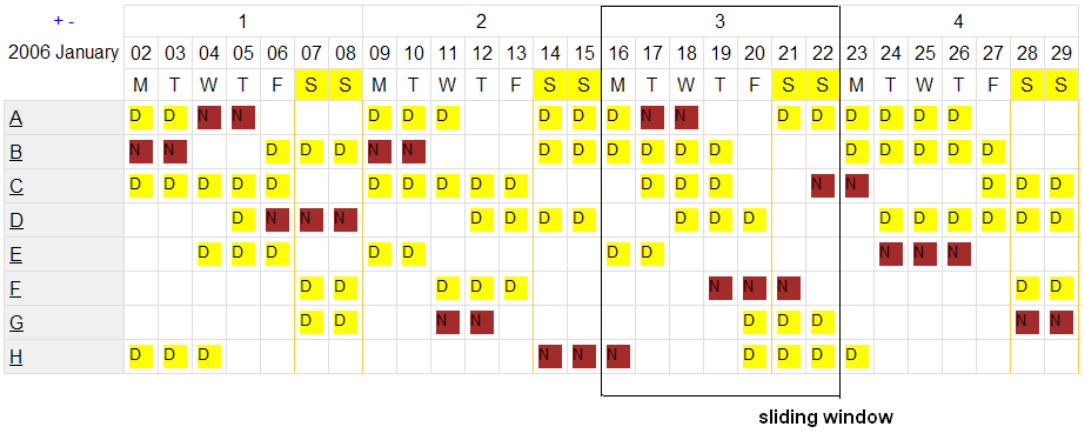


Fig. 5.2 Fragment selection strategy 1: sliding window

In this strategy, we always set n as the total number of nurses in the problem. The size of l can be adjusted according to the size of the different problems to keep a manageable size of fragment s. That is, the larger the n is (i.e. larger number of nurses), the smaller l is.

Fragment selected by this strategy makes it possible to swap certain length l of shifts between any nurses, sot that it can cover neighbourhood structures as single shift swap neighbourhood and block swap neighbourhood investigated in [120].

If l is fixed, the sliding window strategy selects fragment in a deterministic way. The different parts of the problem are selected and re-optimised one by one.

Strategy 2: sliding window with overlap

In strategy one, the sliding widow slides along the horizon of the rosters and usually leads to violations of constraints over the variables at the boundaries of the window. This is because of the interleaving of the constraints over the same variable on the boundary. For example, in Fig. 5.3, if we relax and re-optimise the variables within the sliding window while freezing all variables outside of the window, the night shift N on January 14 and 15 in the shift sequence of NNN for nurse H will possibly be adjusted in the re-optimisation subject to the sequence constraint without seeing the one N shifts on January 16 which lies outside of the window. Strategy 2 considers the overlap between variables over the boundaries of the sliding windows by adding the additional variables over the boundaries to the fragment.

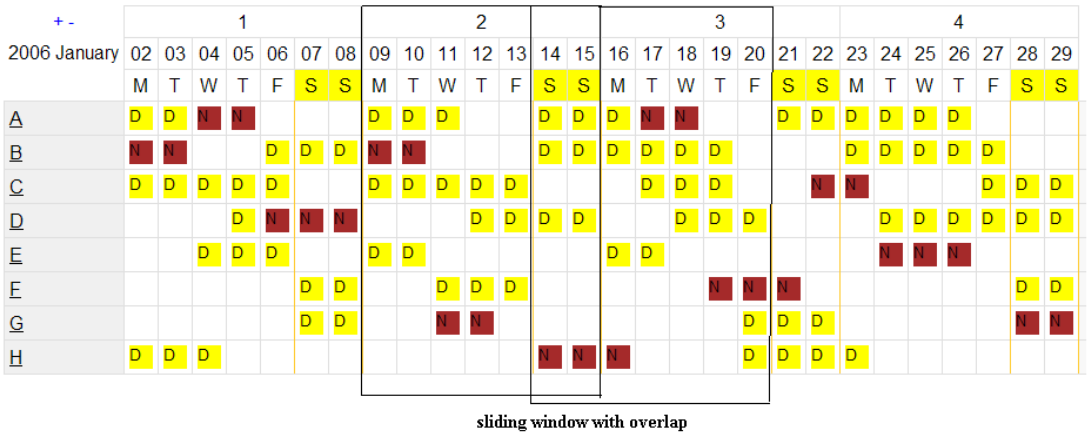


Fig. 5.3 Fragment selection strategy 2: sliding window with overlap

Strategy 3: selection according to the cost of horizontal constraints

Both of above two strategies select fragment of all the nurses in the problem. When the problem is large, more general and efficient strategy is needed to enable an efficient re-

optimisation within the LNS procedure. Strategy 3 utilises the information of constraints themselves to direct the LNS search. Those nurses who have the highest cost (i.e. the most violated horizontal constraints) are selected, and the variables restricted by these constraints are selected into the fragment.

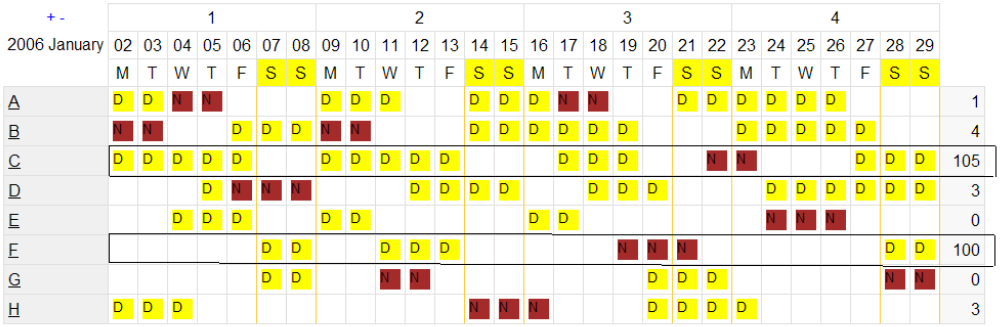


Fig. 5.4 Fragment selection strategy 3: selection according to the cost of horizontal constraints

Fig. 5.4 presents an example of how strategy 3 is used to select the fragment by cost. Firstly the cost of constraints for each row (nurse) is calculated. The k rows of the most violations will be re-optimised. In this strategy, l is the length of the whole scheduling period, and n is defined as k, which is a parameter in the LNS algorithm. Starting with an initial value, k remains the same if the search makes improvement in the current iteration, and is increased by 1 if no improvement can be made. By increasing the value of k, larger areas of the search space can be explored.

We investigate three strategies in this work to choose the fragment of variables. The first two strategies cover **all nurses** in the selected days, so it is possible to swap certain length of shifts between any nurses. The third strategy covers **all days** for selected nurses, so it is possible to swap any blocks of shifts between certain nurses. The fragment selected by these strategies can cover all neighbourhood structures applied in nurse rostering problems, such as single shift neighbourhood, block neighbourhood, and even chains neighbourhood in the literature [120]. What is more, it is more general than those specifically designed neighbourhoods.

5.4.3 Re-optimisation on the fragment

In [132], the influence of the size of the fragment on the performance of LNS is investigated. The advantages of a small fragment are that the fragment can be re-optimised much quickly and lead to an improving solution *if* one exists. The advantage of a larger fragment is that it is more likely that an improving solution *does* exist in the fragment, but more computational time is needed.

We can apply the Branch-and-Bound of CP to re-optimise the fragment. We investigate the relation between the fragment size and search strategies (measured in computational time) in section 5.5.

5.5 Experimental results

5.5.1 Pre-processing and framework

The first step of our constraint-directed LNS approach is a pre-processing procedure to identify *solvable* set of constraints. The solution from this pre-processing is served as the initial solution of the LNS at the next stage, where the remaining small set of soft constraints is to be optimised.

We first evaluate our initialization method in terms of the solution quality and computational time, results shown in Table 5.1. All the problems tested are over-constrained, i.e. no solution can be found if all constraints are imposed crisply. By relaxing soft constraints with different weights (importance) in an increasing order, a feasible solution can be obtained where the least important constraints are relaxed. Due to the strong propagation in CP to detect infeasibilities, this initialisation method is very efficient, i.e. computational time is close to zero second.

Table 5.1 Results from the pre-processing method. For problem Gpost, "-" indicates feasible solutions have been found so there is no need to relax more soft constraints.

Problem	All constraints		Relax $w \leq 10$		Relax $w \leq 40$		Relax $w \leq 100$	
	obj	CPU(sec)	obj	CPU(sec)	obj	CPU(sec)	obj	CPU(sec)
Gpost	infeasible	0	18	50	-	-	-	-
Valouxis	infeasible	0	infeasible	0	1120	65	-	-
ORTEC	infeasible	0	infeasible	0	Infeasible	0	1686	112

5.5.2 Test on fragment selection strategies in the constraint-directed LNS

Fig. 5.5 presents the improvement of solutions at each iteration of LNS by using the three fragment selection strategies on small problem Gpost. Strategy 1 (sliding window) performs the worst, making very limited improvement. Strategy 2 (sliding window with overlap) and Strategy 3 (selection according to the cost of horizontal constraints) have similar performance although they use quite different ways to select the crucial fragment. Both of them make improvement at the early stage of search.

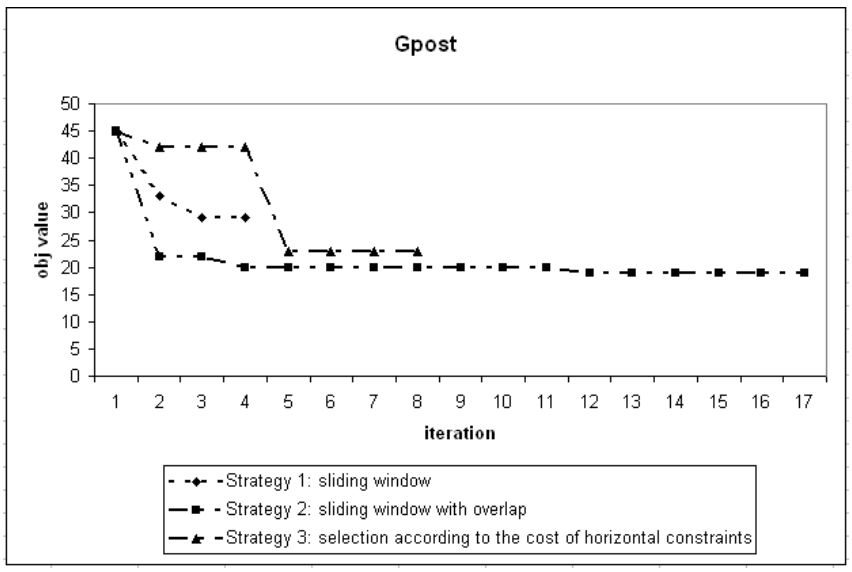


Fig. 5.5 The decrease of objective function value over iterations of LNS using three different fragment selection strategies for problem Gpost

To obtain more insights, we test our algorithm on the large problem ORTEC. All the following experiments and discussions are based on this large problem ORTEC. Fig. 5.6 presents the information of the performance from the three fragment selection strategies. The strategies are tested based on the same initial solution. The terminations of the first two strategies are deterministic. That means the number of iterations executed is fixed after l (the length of sliding window) is settled. The number of iterations equals to the length of scheduling period divided by l. Strategy 3 terminates in a different way. It picks out k rows of the schedules with the highest cost to re-optimise iteratively until the time limit is reached. From Fig. 5.6 we can see that Strategy 1 can improve solution from 1639 to 929 while strategy 2 can improve solution from 1639 to 610. Strategy 3 performs the best, improving the initial solution to 435.

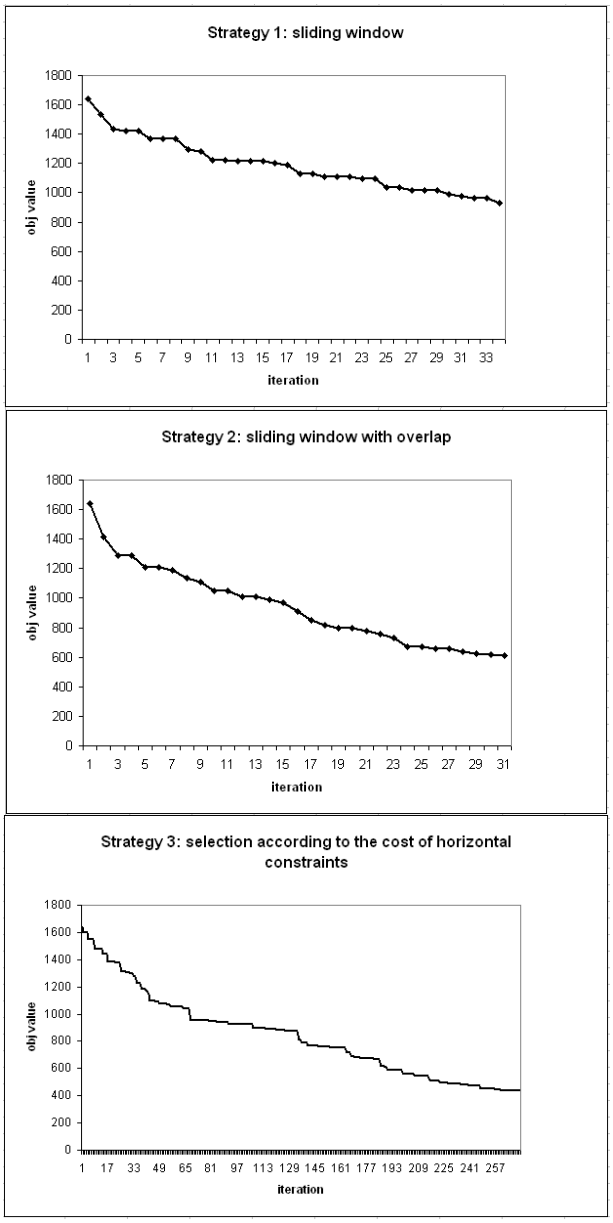


Fig. 5.6 The decrease of objective function value over iterations of LNS using three different fragment selection strategies for problem ORTEC

5.5.3 Test on the effort of search

We test the CPU time spent on each iteration in different size of the fragment with respect to each strategy.

Fig. 5.7 shows the improvement of solutions over the iterations with different fragment size for Strategies 1 and 2. As stated above, the size of the fragment is denoted by $s = n \times l$. For Strategies 1 and 2, n is fixed as 16 (the number of nurses in the problem) and different l are compared in Fig. 5.7. For Strategy 3, l is fixed as 35 (the length of the scheduling period) and n is distributed within [4, 12], illustrated in Fig 5.8. The improvement of solutions over iterations with Strategy 3 can be seen from the last plot in Fig 5.6.

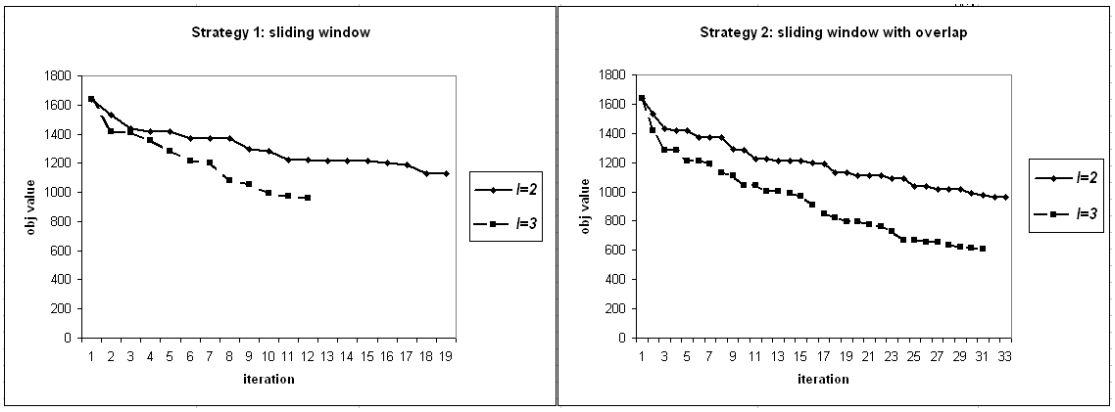


Fig. 5.7 The decrease of objective function value over iterations of LNS with different fragment size for strategies 1 and 2 for problem ORTEC

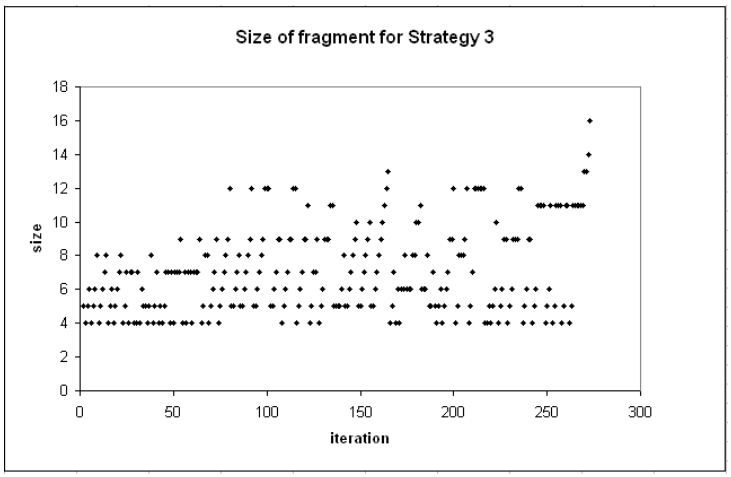


Fig. 5.8 Size of fragment for strategy 3, denoted by *n* for problem ORTEC

Table 5.2 presents the improvement of solutions and their corresponding CPU time with different fragment sizes for Strategies 1, 2, and 3.

It can be seen that the sizes of fragment has significant impact on both the quality of solutions and the computational time. With larger fragments, the objective value improvements more, but the corresponding computational increases.

Table 5.2 Comparison of solution improvement and computational time of LNS with different fragment sizes for problem ORTEC

Fragment strategy and its size	Avg CPU time/iteration	Total CPU time	Improved obj value
Strategy 1 s=16×2	0.1	1.8	1129
Strategy 1 s=16×3	20	300	929
Strategy 2 s=16×2	0.1	3.2	964
Strategy 2 s=16×3	24	760	610
Strategy 3 s=7(Avg)×35	0.4	115	435

5.5.4 Comparison with other approaches in the literature

We compare our results with other methods in the literature in Table 5.3. Hybrid GA, Hybrid VNS and Hybrid IP are the same methods which are compared with in previous chapter 4. Hybrid CP is a CP approach hybridised with a Variable Neighourhood Search, investigated in chapter 4.

Since both chapter 4 (i.e. hybrid CP with VNS) and chapter 5 (i.e. hybrid CP with LNS) are under the two-stage hybrid approach framework "feasible initial solution + improvement", we first make a comparison between these two methods. In order to have a fair comparison, we set the computational time limit as ½ hour, the same as that in Hybrid CP in chapter 4. Comparing to the Hybrid CP with VNS in chapter 4 [133], using the same computational time, the constraint-directed LNS obtained better results for 9 out of 12 instances. Comparing to the approaches in [115, 123, 125], our constraint-directed LNS can obtain better results on 6 out of the 12 instances within only half of computational time. These competitive results obtained within much less computational time limit demonstrate the efficiency of the constraint-direct LNS.

We also emphasize that in all other approaches, i.e. [115, 123, 125], the design of neighbourhood is very delicate. It requires experts in the domain knowledge. However,

in the hybrid CP with LNS proposed in this chapter, the LNS is enhanced by constraints. More specifically, the design of neighbourhood is very general and the search is guided by the information of constraints. In conclusion, the results demonstrate that the integration of CP with LS can find good quality solutions in less computational time for highly constrained NRPs.

Table 5.3 Results compared with other methods in the literature. The best results are shown in bold.

Method	Hybrid GA[125]	Hybrid VNS[115]	Hybrid IP[123]	Hybrid CP with VNS[133]	CP+LNS
CPU time	1 hour	1 hour	1 hour	½ hour	½ hour
Jan	775	735	460	616	**395**
Feb	1791	1866	1526	1736	**1261**
Mar	2030	2010	**1713**	2766	1831
Apr	612	**457**	391	956	731
May	2296	2161	2090	**1786**	2111
Jun	9466	9291	8826	8700	**6201**
Jul	781	481	**425**	650	751
Aug	4850	4880	3488	2171	**2121**
Sep	615	647	**330**	1300	851
Oct	736	665	445	616	**395**
Nov	2126	2030	1613	1620	**1136**
Dec	625	520	**405**	496	1101

Recall that the initial solution of LNS actually applies the Model (Direct Initial CSP) in chapter 4. Comparing the computational time shown in Table 5.1 and the direct initial solution generation method of chapter 4 (i.e. nearly 30 minutes), it can be seen that the sequence based initial solution generation strategy proposed in chapter 4 can generate better quality initial solution than the direct initial solution generation strategy. However, it needs more computational time to construct a feasible solution.

There is a trade-off between the quality of the initial solution and the search effort on the second stage local search. The larger is the neighbourhood, the less is the sensitivity

of the initial solution to the performance of the local search. More specifically speaking, although the quality of the initial solution generated by the direct initial solution generation strategy (fed to LNS) is worse than the sequence based initial solution generation strategy, the second stage LNS can produce better improved solution comparing with VNS (with the sequence based initial solution generation strategy).

5.6 Conclusions

In this chapter, we implement a constraint-directed local search in a LNS scheme to large and constrained nurse rostering problems. The search is restricted in feasible region preserved by CP and iteratively improved by LNS. Three different fragment selection strategies are proposed to select crucial part of solution to be relaxed and re-optimised. The issue of how much search effort should put to re-optimise the fragment is also investigated in order to get good solutions in reasonable computational time. The experiment results show that the proposed constraint-directed LNS is a simple yet efficient algorithm to provide good quality solution in less computational time for nurse rostering problems. This chapter is based on the work published at Proceeding of the 6th International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS'09) at the 15th International Conference on Principles and Practice of Constraint Programming (CP'09).

The problem decomposition in both chapters 4 and 5 are based on the constraints. A two-stage hybrid approach framework "feasible initial solution + improvement" are applied in both of the chapters.

Chapter 6 CP based column generation approach to nurse rostering problems

6.1 Introduction

6.1.1 Background

In chapters 4 and 5, hybrid methods are investigated under the framework "initial solution + improvement". CP is efficient to construct initial feasible solutions while local search and meta-heuristics are efficient to *heuristically* improve the solutions. However, optimality cannot be guaranteed or proven. We desire to solve the problem efficiently, as well as know the quality of solutions, e.g. how far obtained solution is away from the optimal one.

NRPs can be modeled and solved by OR techniques such as Linear Programming [134], Integer Programming [135] and Mixed-Integer Programming [136] due to their strengths in optimality reasoning and relaxation. For these large-scales Integer Program, one classic decomposition method, Dantzig-Wolfe Decomposition, can be applied to decompose the large-scale Integer Program. The column generation is an efficient algorithm for solving the decomposed problem efficiently as we introduced in section 2.8.1 of chapter 2.

As we introduced before in chapter 2, airline crew scheduling problem is a well know example which can be Dantzig-Wolfe decomposed and solved by column generation. The solution approach has also been applied to personnel scheduling problem, employee timetabling problems, etc.[40-43]. All problems tackled by Dantzig-Wolfe decomposition and column generation share some similar feature that the problem can be inherently decomposed. This can be seen as to select a subset of individual patterns (columns) from a huge pool of all possible weighted patterns (columns) to construct the best complete solution to the problem. The individual patterns should present some

desired features of the problem [44]. For example, in airline crew scheduling problems, each schedule for the crew should satisfy a large set of working regulations. NRPs which belong to personnel scheduling problem share similar feature with crew scheduling problem. This makes the Dantzig-Wolfe decomposition and column generation a good solution approach to NRPs.

In early related works, the pricing subproblem is usually solved by dynamic programming [40]. However, in the works of [42] and [43], the results shown that using dynamic programming in pricing subproblem solving is very time consuming due to the large set of constraints presented in the problem. Hence, The CP based CG approach has been firstly introduced in [42] and [43] to model and solve the crew assignment problem. It is a hybrid decomposition approach where CP is used to solve the pricing subproblem and CG is used to handle the master problem. The CP-CG approach has since been widely applied to airline crew scheduling [65], vehicle routing [71] and bin packing [68] problems.

Due to its ability of strong modelling and feasibility reasoning, CP within the CP-CG approach is used to generate the large pool of patterns subject to the constraints in the problem. The selection procedure of patterns is processed by column generation. Integer variables in the master problem represent which columns are chosen to construct the complete solution to the problem. The Linear Program relaxation of the Integer Program master problem is used to iteratively derive the optimality and solve the problem of subset of columns. Through the integration, the hybrid CP-CG approach benefits from both the feasibility reasoning of CP and the optimality reasoning of Linear Programming.

In the literature, CP-CG is mainly applied to airline crew assignment problems [42, 43, 45] at the early stage. Research mainly focused on solving the pricing subproblem which is captured by a *shortest path constraint*. The efficiency of the CP-CG algorithms mainly rely on the development of efficient *cost filtering* algorithms for this *shortest path constraint*.

CP-CG has also been applied to personnel scheduling or personnel timetabling problems, where some of them also focus on the *cost filtering* algorithms in CP subproblem solving. In [44], a *cost-regular* constraint is used to model the key features of a set of personnel scheduling problems. Its cost filtering algorithm is calculated based on the *shortest path of the layered directed graphs*. The complexity of the constraints is not presented in the work. For the subproblem, the shift scheduling problem, all constraints are modelled as hard constraints. In a real-world case study of personnel timetabling for a bank, the authors pointed out that the *cost-regular* constraint is inefficient when the working regulations are complex and scheduling period is long, which are often the case in real-world problems.

In [72], instead of only focusing on the cost filtering in CP as in most of the previous work [42, 43, 45], the authors proposed two search strategies to tackle the problem of slow convergence and the difficulty of reaching integer solutions in CP-CG. To speed up the reduction of the objective value of the linear relaxation of the master problem, the dual strategy selects the shift with the l largest dual value to drive the search towards solutions with negative reduced cost. To speed up the convergence to integer solutions, the master strategy stores the information of shifts that have been assigned, and choose and assign the less used shift first. This strategy can be seen as a heuristic which choose diverse columns to help obtain integer solutions.

6.1.2 Motivations

As indicated in [46, 47], theoretically, the pricing subproblem asks for the columns with the minimum negative reduced cost at each iteration of CG until all of the columns with negative reduced cost have been found. The pricing subproblem is thus a minimization problem. However, in practice, the pricing subproblem is usually solved as a decision problem. That is, any column which has negative reduced cost (i.e. more than one column and not necessarily with the minimum reduced cost) can serve as the candidates to enter the restricted master problem.

Chapter 6 CP based column generation approach to nurse rostering problems

Generating feasible columns instead of optimal ones presents an easier problem. However, feasible but not optimal columns lead to a slower convergence to the optimum with respect to the number of required iterations of column generation. This slow convergence in the CG approaches is the first issue we try to address in this chapter.

Another issue we face in developing efficient CP-CG is that the CP pricing subproblem tends to generate similar columns [46, 47] due to the default depth first search in CP. This leads to the difficulty of reaching integer solutions of the master problem.

With regard to the above mentioned research issues, we aim to present in this chapter an application of the CP-CG approach to complex nurse rostering problems. In order to deal with the slow convergence of the CP-CG, we propose to apply a cost threshold, working together with the negative reduced cost constraint, to price out *good quality* columns. That is, not only the generated columns have negative reduced cost, but also their original costs (i.e. cost coefficient in the objective function) are below a threshold. This cost threshold is repeatedly updated (reduced) according to the solution information collected during the procedure of the CG. With regard to the similarity of the generated columns, we apply the Depth Bounded Discrepancy Search to obtained diverse columns. This strategy, to some extent, contributes to reaching integer solutions of the master problem. The main contributions of the chapter are as follows:

- We propose a CP-CG solution procedure where a complete model formulates all constraints in several real-world benchmark nurse rostering problems which we are concerned with.
    1. Various constraints have been modeled in the CP paradigm by using primitive and (soft) global constraints;
    2. Instead of generating columns for each nurse, we only generate columns to each category of nurses to eliminate symmetry.
- We apply the Depth Bounded Discrepancy Search in CP-CG to obtain diverse columns for the master problem concerning the integrality. An adaptive bound

tightening strategy is devised to adaptively obtain high quality columns during the problem solving.

6.2 Modelling the nurse rostering problem

In this section, we first present the model of the master problem within the CP-CG framework. As stated above, a nurse roster consists of the assignment of schedules (a sequence of day-on shifts and day-off) to each nurse to ensure that sufficient employees are present to perform the shift duties required (coverage constraint). The master problem can thus be formulated as an Integer Program to pick subsets of feasible schedules to construct a complete roster with the minimal cost. At this stage, we do not need to consider the detailed constraints which restrict the working patterns, etc. All these constraints are encapsulated as the features of columns $\{\alpha(1), \ldots, \alpha(k)\}$ as shown in Fig. 2.6. These features of columns will be formulated and taken care of in the CP paradigm within CP-CG.

6.2.1 Formulating the master problem as Integer Program

Problem size parameters:

--N: set of nurses (index i)
--D: set of days in the scheduling period (index j)
--S: set of shift types, i.e. *Late*, *Early*, *Night*, *Off*, etc (index k)

Nurse parameters:

--G: set of nurse categories (i.e. different working contracts, e.g. 20, 32 or 36 hours per week, respectively) (index m)
--F_m: set of feasible schedules for nurse category m with respect to related constraints of contract stipulations (index l)

--a_{ilmjk}: is 1 if schedule l for nurse i in category m covers the required shift k on day j; 0 otherwise

--c_{il}: penalty of schedule l violating the related constraints of contract stipulations of nurse i

Demand coverage parameter:

--R_{jk}: coverage demand of shift type k on day j

Decision variables:

-- y_{il}: binary decision variables in the master problem, taking value 1 if nurse i is assigned to schedule l; 0 otherwise.

With the above parameters and notations we have the **Master Problem Formulation (NRPs MP)**:

$$\text{(NRPs MP)} \quad \min \sum_{m \in G} \sum_{i \in N} \sum_{l \in F_m} c_{il} y_{il} \tag{6-1}$$

$$\text{s.t.} \sum_{m \in G} \sum_{i \in N} \sum_{l \in F_m} a_{ilmjk} y_{il} = R_{jk}, \forall j \in D, k \in S \tag{6-2}$$

$$\sum_{l \in F_m} y_{il} = 1, \forall i \in N \tag{6-3}$$

In the nurse rostering problems MP, the objective function is linear over the schedules y_{il}. The penalty of the entire solution (roster) is defined as the sum of the penalties of the selected schedules, i.e. objective (6-1) of this master problem aims to minimize the sum of penalties associated with the individual schedules y_{il} the nurses are assigned to. Constraint (6-2) defines the required number of nurses for each shift on each day (exact coverage). Formulating the coverage constraint as such allows flexible substitutability between nurses, i.e. schedules are exchangeable among nurses of the same category m. Constraint (6-3) assigns exactly one schedule to each nurse that is feasible to his or her specific related constraints. The similar formulation of the restricted master problem has also been applied in [96]. One thing should be note that constraint (6-2) works as the connecting constraint as we introduced in section 2.8.1.

In the above seemingly simple model, a large amount of complexity is actually hidden in the definition of schedule l in y_{il}, i.e. all the constraints in Appendix are implicitly modeled by the definition of schedule l, and the generation of each column (feasible schedule) must be subject to these constraints. The definition of schedule l works as the independent constraints as we introduced in section 2.8.1. Since all related constraints are encapsulated into schedule l, there is no need to change the master problem within CP-CG with respect to different instances of nurse rostering problems.

6.2.2 Formulating the pricing subproblem in CP

Now we consider the features of these columns, i.e. to model the subproblem concerning all constraints (except the coverage constraint). As stated in the introduction of column generation, the pricing subproblem with the general form:

$$\text{(P)}\ \pi_i = c_\alpha - \sum_{i=1}^{i=n} \lambda_i \alpha_i < 0; \tag{6-4}$$

$$\alpha \in F; \tag{6-5}$$

concerns two groups of constraints, the negative reduced cost (6-4) and the feasibility constraints (6-5).

Considering the reduced cost constraint (6-4), we define the reduced cost π_{il} in our problem as follows:

$$\pi_{il} = c_{il} - \gamma_i - \sum_{m \in G} \sum_{i \in N} \sum_{l \in F_m} \lambda_{jk} a_{ilmjk} \tag{6-6}$$

--π_{il} is the reduced cost of column l for nurse i

--c_{il} is the cost coefficient of column l for nurse i

--γ_i is the dual value of constraint (6-3) for nurse i

--λ_{jk} is the dual value of constraint (6-2) for shift k on day j

--a_{ilmjk} corresponds to the coefficients matrix in (6-2) of the master problem

In the equation of the reduced cost π_{il} in (6-6), γ_i and λ_{jk} are the dual values obtained from the Linear Program solution of the master problem (NRPs MP) (known). c_{il} and a_{ilmjk} need to be obtained from the solution of the pricing subproblem (unknown). Each schedule l for nurse i is a sequence of shifts that satisfy all the related constraints ($l \in F_m$) and introduces a new column in the master problem with cost coefficient c_{il} (we use the term "cost" in the CP pricing subproblem which corresponds to "penalty" in the master problem).

The modeling of feasibility constraint (6-5) in our CP-CG requires careful consideration for the complex nurse rostering problems. The issue of how to efficiently model the complex constraints in CP has been studied in our previous work in chapter 4. The *cardinality* (x, v, l, u) and *stretch*(x, v, l, u, P) constraints, together with primitive constraints are applied to model the problem.

We now model our CP pricing subproblem within CP-CG by defining the detailed feasibility constraint (6-5) $\alpha \in F$.

The nurse rostering problems which we are concerned with are defined as the follows:

$$\text{(NRPs P)}\quad c_{il} = E(l) = \sum_{C_e \in C} w_{C_e} \mu^{C_e}(l)$$

where $E(l)$ represents the evaluation of schedule l. $\boldsymbol{C}$ is the set of constraints. The evaluation of soft constraint Ce is calculated by $w_{C_e} \mu^{C_e}(l)$, where w_{Ce} is the weight of the constraint given in Appendix and $\mu^{Ce}(l)$ is the violation measure of soft constraint.

The constraints we modeled here are from one of most complex benchmark nurse rostering problems ORTEC which is given in chapter 4. It can be modeled in a similar way in this pricing subproblem. Therefore, we do not present the suproblem here.

In our CP-CG framework, we are concerned with two different types of decision variables, namely binary variables in the Integer Program model and finite domain variables in the CP model. A communication variable has thus been introduced to link

these two types of decision variables and reflect the interactions between the two models:

--x_{imjk}: binary variables as communication variables between the master problem and the pricing subproblem. It takes value 1 if nurse i in category m is assigned to shift k on day j; 0 otherwise. For example, if $s_{ij} = k$, where nurse i is in category m, then $x_{imjk} = 1$. Note that x_{imjk} serves the role of extracting a_{ilmjk} coefficients from s_{ij}.

After (NRPs P) is solved and the values of c_{il} and s_{ij} have been obtained, the communication variable x_{imjk} transforms s_{ij} in the pricing subproblem to the coefficient a_{ilmjk} in the master problem. For example, schedule l = [Day, Day, Night, Night, Off, Off, Day] can be transferred as a column $[11000010011000]^T$. Values of c_{il} and a_{ilmjk} are used to calculate the reduced cost π_{il} in (6-6).

6.3 Solution Procedure

As stated above, in the CP-CG approach CP is used to solve the subproblem (NRPs P) to generate columns for the master problem. To combine these columns (schedules) generated by the subproblem into a complete roster, the master problem (the binary Integer Program problem NRPs MP) tries to minimize the total cost (the sum of costs from all chosen schedules) by choosing exactly one line of schedule for each nurse, while satisfying the coverage constraint.

The overall solution procedure of CP-CG is illustrated in Fig. 6.1. A feasible solution is firstly generated and fed into the restricted master problem (NRPs MP). Each column in the initial solution is associated with a cost calculated by $c_{il} = E(l) = \sum_{C_e \in C} w_{C_e} \mu^{C_e}(l)$, and the highest cost is used as the threshold in the CP pricing subproblem. The candidate columns which preserve feasibility and have a cost of below the cost threshold are generated by solveCPSubProblem($\tilde{c}$, *DDS*) using the Depth Bounded Discrepancy Search (DDS) strategy[137]. Columns with negative reduced costs are then priced out from these candidate columns and added to the restricted master problem to solve the

Linear Program problem again. New lower bound of Linear Program relaxation and the new dual value λ are derived in the Linear Program, as shown in the inner loop in Fig. 6.1.

Within the outer loop in Fig. 6.1, the Branch-and-Bound is run based on the generated columns to obtain an integer solution y_{il}. The solution serves as the upper bound of the master problem. The cost threshold is then updated as the highest cost of all columns in the current solution. The column pool is then emptied and DDS restarts to generate columns with a tightened bound. The parameters of DDS direct the search to different sections of the tree. If the cost threshold remains the same, parameters of the DDS search will be adjusted to find columns in different parts of the search tree. This bound tightening mechanism aims to avoid generating columns of high costs. The whole procedure will stop until certain condition is met (no improvement of $c_{il}y_{il}$ in a certain number of iterations). An integer solution with certain gap to the Linear Program relaxation is obtained.

In the literature, the integer solutions to the master problem are obtained by either running the Branch-and-Bound on the Linear Program relaxation or running the Brand-and-Price algorithm. In our CP-CG, instead of running Branch-and-Price at each node to derive optimal solutions, we run Branch-and-Bound on the generated diverse columns which are of good *quality* by using search strategies for the CP to derive integer solutions to the master problem.

Algorithm. CP based Column Generation Approach
$\tilde{A}$: subset of feasible columns of A
λ : dual values
$\{\alpha^{(1)}, \ldots, \alpha^{(k)}\}$: columns generated by CP
$\tilde{c}$: cost threshold of columns

$\tilde{A}$:= get initial columns // see section 3.1
$\tilde{c}$:= initial cost threshold
Repeat
  Repeat
    A_p := empty column pool
    λ := solve the RMP $\tilde{A}$
    A_p :=$\{\alpha^{(1)}, \ldots, \alpha^{(k)}\}$ = solveCPSubProblem($\tilde{c}$, *DDS*) // see section 3.2
    price columns with negative reduced costs from A_p and add them to $\tilde{A}$
  Until stop condition is met (number of iteration)
  y_{il} = solve the integer solution of MP $\tilde{A}$ using B&B
  update the cost threshold $\tilde{c}$ // see section 3.3
Until termination condition is met (without improvement of $c_{il}y_{il}$)

Fig. 6.1 The CP based column generation solution procedure

To illustrate more clearly what are columns of good structure, i.e. diverse columns, we present a small example here. Assume that we already have a set of columns as follows:

l_1 = [Day, Day, Night, Night, Off, Off, Day]
l_2 = [Day, Day, Day, Day, Off, Off, Off];
l_3 = [Day, Day, Day, Off, Off, Off, Off];
l_4 = [Day, Day, Day, Day, Day, Off, Off].

The master problem (NRPs MP) chooses exactly one column for each nurse to construct a whole roster which satisfies the coverage constraint (for example, 3 Day shift and 1 Night shift are assigned on the first day). It can be seen that all of current columns $l_1l_2l_3l_4$ have Day assignment on the first day. To satisfy the coverage constraint, new column such as l = [Night, Night, Day, Day, Off, Off, Day[which has Night shift assignment on the first day (of course all these columns need to have negative reduced cost as well) are expected.

6.3.1 Initial solution

We apply our initial solution generation strategy proposed in our previous work in chapter 4 here to generate initial solution for CG.

6.3.2 Depth Bounded Discrepancy Search to obtain diverse columns

In CP, Depth First Search (DFS) [3] a standard search strategy. It traverses the search tree by searching down to the leaf of one branch before starting another branch. Whenever a dead-end of a branch with no solution is reached, the search goes back to an upper depth of the search tree, i.e. backtracking, and continues to search down another branch. The main drawback of DFS is that, given a limited computational time, even for problems of moderate size, it can only explore a very small part of the search tree before moving to another part, returning very similar solutions (with only the last several variables taking different values) [3, 69].

Limited Discrepancy Search (LDS) [14] is an alternative search strategy to explore the search tree iteratively based on the innovative idea of discrepancy. A discrepancy is "any decision point in a search tree where we go against the heuristic"[14]. Assume a heuristic orders the branches in a left-to-right manner by estimating which branch is more likely to contain solutions. For convenience, we assume that taking the left branch follows the heuristic. Taking the right branches breaks the heuristic, i.e. a discrepancy. LDS explores the search tree in a series of DFS with k discrepancies, where each DFS defers the heuristic k times (i.e. going to the right branches k times), $k = 0,\ldots,(d–1)$, d is the depth of the tree. LDS thus has the chance to explore the right part of the search tree and is likely to find the solution quicker compared to the standard DFS [14, 69, 73].

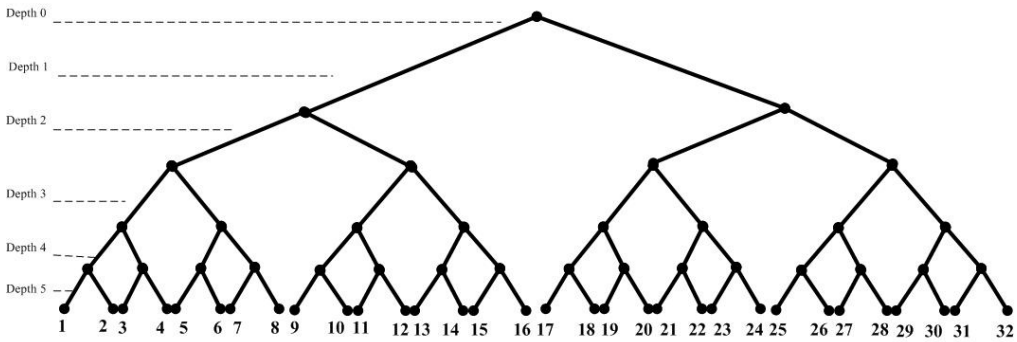
LDS treats all the discrepancies the same irrespective of the depth at which the discrepancies happened. However, heuristics tend to be less informative and make more mistakes near the root of the search tree [137]. Based on the idea of LDS, the Depth Bounded Discrepancy Search (DDS) [137] is also an iterative process (d iterations of

DFS). However, discrepancies in DDS happen at early stage of the search, bounded by depth i in iteration i, $i = 0,\ldots,(d-1)$, i.e. all discrepancies must happen at and above depth i, not allowed below depth i. That is, at depth i of the tree, the search must take the right branch (discrepancy) of the node. At depth above i, the search explores both the right and left branches. Below depth i, the search must take the left branch (i.e. follow the heuristic, no discrepancy allowed in the later stage) at all nodes. The latest discrepancy in DDS happens at depth i in the tree. This controls (forces) the search to traverse to different parts of the tree, resulting into diverse solutions.

Fig. 6.2 illustrates how DDS traverses the tree. Without loss of generality, the binary tree represents assignments of the simplified variables sj, $j = 1,\ldots,5$, and the domain of sj is (Day, Off). We assume the left and right branch takes the value Day and Off, respectively. In the first iteration of DDS, depth $i = 0$, so no discrepancy happens. The search takes the left branch at all nodes, leading to path (1) DDDDD. In the second iteration, depth $i = 1$, DDS obtains path (17) ODDDD by taking the discrepancy at depth 1 (i.e. takes the value Off at the right branch). In the third iteration, the discrepancy must happen at depth $i = 2$, thus leading to paths (9) DODDD and (25) OODDD. Following the same rule, it can be seen that the paths explored by DDS (illustrated at the bottom of Fig. 6.2) lead to diverse assignments, i.e. early variables in the assignments also take different values.

In the default settings in ILOG Solver, based on the standard Depth Bounded Discrepancy Search (DDS)[29], an extended DDS search is defined by introducing three additional parameters (*depth, width, MaxDiscrepancy*) as follows. The first parameter *depth* restricts the depths the search explores to be between *depth*×*i* and *depth*×(*i*+1) in iteration i. That is, in the first iteration, $i = 0$, the search explores the nodes at depths above *depth*. In the second iteration, $i = 1$, it explores the nodes between *depth* and *depth*×2, and so on. The second parameter *width* is used to restrict the number of paths explored by limiting the number of discrepancies occur between *depth*×*i* and *depth*×(*i*+1). The third parameter, *Max Discrepancy*, restricts the total number of

discrepancies, i.e. it defines the total number of times the search is forced (diversified) to different parts of the tree.



depth	path															
0	1															
1	17															
2	9	25														
3	5	13	21	29												
4	3	7	11	15	19	23	27	31								
5	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

Fig. 6.2 Depth Bounded Discrepancy Search

6.3.3 Pricing subproblem with threshold

In our CP-CG approach, it is easy to generate feasible candidate columns due to the efficient constraint handling in CP. However, among the huge number of columns, most are of poor quality with high cost, and are not helpful to reduce the objective function value. The issue of selecting "good" candidate columns to reduce the computational time in column generation was first discussed in [138]. In [139], existing columns with reduced cost of zero have been used with fast local improvement algorithms to construct columns with positive reduced costs for their maximization problem.

To eliminate poor columns in CP-CG, we introduced an additional cost bounding constraint, threshold $\tilde{c}$, to the pricing subproblem. The enhanced model for the subproblem based on (NRPs P) is presented as follows:

$$\text{(En NRPs P): } l \in F$$

$$\pi_i < 0$$

$$c_i < \tilde{c}$$

This (En NRPs P) model is solved by CP as a constraint satisfaction problem to seek feasible solutions as candidate columns. This cost threshold plays an important role in the search procedure. In the research of cost filtering algorithms for Constraint Optimisation Problems (or the problem with soft constraints), the algorithm associated with the (global) constraints can be used to filter the domain of the *cost variables* AND *decision variables* (we refer to Focacci's work [140] for further discussion). To achieve these propagations, cost filtering algorithms need to be implemented for each soft constraint in the model, as in the work in [44, 119, 141]. However, there is a trade-off between the time needed and the efficiency of the algorithm.

As we stated before, in our work, we do not implement cost filtering algorithm of each soft constraint to filer the domain of cost variable c_{il}. Rather we use this cost (i.e. the violation measure of soft constraint) to filter the domain of decision variables s_{ij}. The filtering rule is: if the cost of a sequence of assignment at node i is greater than the upper bound (cost threshold), we remove this value from its domain, i.e. the node i is pruned.

We should note that this cost threshold does not completely prevent the generation of columns with bad (large) original cost which may have good reduced cost. These columns may make other columns fit in very nicely to help reaching the integer feasibility of the master problem. The bound tightening mechanism adaptively tightens the bound. At the beginning of the solution procedure, the cost threshold is set at a relatively high value thus columns with large cost also have the chance to enter the master problem. By adaptively tightening the cost threshold, the search gradually accepts better columns with smaller cost.

This filtering rule, working with the feasibility pruning (the domain consistency rule) can help to accelerate the tree search. The traverse in the tree (i.e. the generation of columns) is controlled by both the search strategy DDS (with its parameters) and the upper bound of the columns (the cost threshold) to obtain good quality columns.

6.4 Experimental results

6.4.1 Algorithm setting

The hybrid CP-CG approach is implemented in C++, linking ILOG CPLEX 10.0 to solve the Linear Program and Brand-and-Bound for the Inter grogram problem, and ILOG Solver 6.2 to solve the CP pricing subproblem as well as to provide initial solutions. Default parameters have used in all the CPLEX software packages unless otherwise stated. The parameter settings in the CP-CG approach for all problems are given in Table 6.1. The total computational time is set as one hour, the same as that of existing methods in the literature (see Table 6.4). Other parameters are set based on observations of the approach on small problems Gpost and Valouxis by a number of initial tests. To control the size of the master problem solved by Branch-and-Bound after column generation, the maximum number of columns is set as 10000.

Table 6.1 Parameter settings for the CP-CG approach

Parameters	Values
Total CPU time limit	1 hour
Maximum CPU time for CP solver per iteration	60(sec)
Maximum number of iterations	50
Maximum number of columns in LP	10000

6.4.2 Performance of strategies in CP-CG

First, we comment on the number of columns processed. Table 6.2 presents the number of columns generated by DDS with and without the cost threshold. The settings of these threshold values for different problems are based on the weights of the soft constraints shown in the Appendix. These weights decide the original cost c_{il} of columns l of each problem, see the equation $c_{il} = E(l) = \sum_{C_e \in C} w_{C_e} \mu^{C_e}(l)$ in section 6.2.2.Without the cost threshold, a large number of columns with quite large cost can be generated. However, a large part of these columns has no contribution to the restricted master problem. The last column in Table 6.2 demonstrates that with the help of cost threshold, a large

number of columns with costs above the threshold can be discarded to provide good columns at each iteration without the loss in the solution quality. It can be clearly seen from the faster convergence of DDS with cost threshold in Fig. 6.3 that "good" columns make the real contribution to the search procedure. Fig. 6.3 presents the decrease of the objective function value over the iterations of column generation. Since the maximum CPU time per iteration in the CP solver is settled as shown in Table 6.1, Fig. 6.3 can also illustrate the relationship between the objective value and CPU time.

Table 6.2 DDS with and without cost thresholds in CP-CG. "up to limit" indicates that the search stopped after reaching the maximum number of columns 10000, as shown in Table 6.1.

Problem	**Without threshold**	**With adaptive threshold**		
	total no. of columns	threshold values	no. of columns	no. of discarded columns
Gpost	5862	2	2567	7433
Valouxis	8562	40	3860	6140
ORTEC	up to limit	100	4586	5414

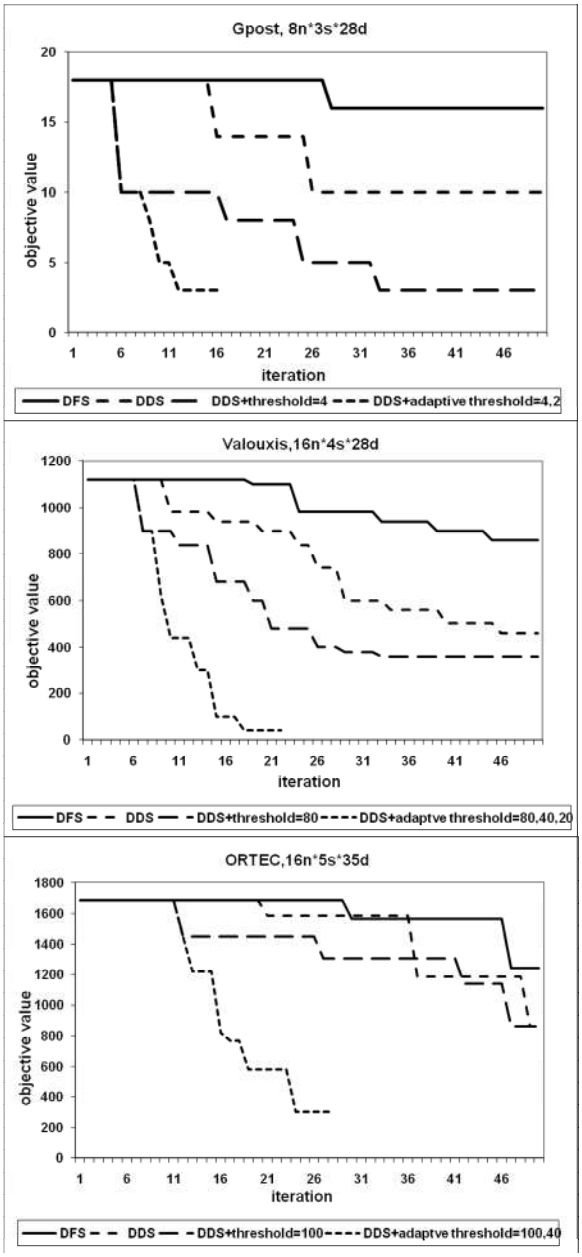


Fig. 6.3 The decrease of objective function value over iterations of CP-CG with different search strategies for the three problems. "16n*5s*35d" denotes problem with 16 nurses, 5 shift types and 35 days.

Next we comment on the decrease of Linear Program lower bound in CP-CG with four search strategies, namely DFS, DDS, DDS with static threshold and DDS with adaptive threshold as shown in Fig. 6.3. DDS with static cost threshold continuously decrease the Linear Program lower bound comparing with DFS and pure DDS. The performance is further improved by the threshold tightening strategy which adaptively updated the

threshold according to information collected during the search. With the adaptive cost threshold, less number of columns is being processed, so less iterations of CG are executed comparing with DFS and pure DDS.

To provide an in-depth analysis of the search strategies in CP-CG on the improvement of the Linear Program lower bound and integer solution obtained after Branch-and-Bound, Table 6.3 compares detailed numerical results of DFS, DDS and DDS with the adaptive threshold. For each strategy, objective values of solutions after the initialization, Linear Program relaxation and Branch-and-Bound are presented. In terms of Linear Program lower bound, DDS with the adaptive cost threshold makes the most improvement to initial solutions, although both DFS and DDS strategies can also improve initial solutions to a certain scope. In terms of integer solution after Branch-and-Bound, for the small problem Gpost, integer solutions can be found by all the three strategies, where the optimal solution has been only found by using DDS with adaptive cost threshold. However, for larger problems, integer solutions can only be obtained by DDS or DDS with adaptive cost threshold within the time limit, of which the latter obtained much better results for both problems.

Table 6.3 Numerical results of CP-CG with DFS, DDS and DDS + adaptive cost threshold. Optimal results are shown in bold. Avg CPU(sec): the average time of a single iteration of column generation by CP; Z_{IN}: the objective value of the initial solution; Z_{LP}: the objective value of the best solution of the LP relaxation of the master problem ob3tained at the end of column generation procedure; Z_{IP}: the objective value of the best integer solution obtained after applying B&B on generated columns.

Problem	Strategy	Z_{IN}	Z_{LP}	Z_{IP}	Avg CPU(sec)
Gpost	DFS	18	16	16	5.62
	DDS	18	10	14	3.58
	DDS + adaptive cost threshold	18	3	**3**	3.68
Valouxis	DFS	1120	860	--	8.21
	DDS	1120	460	540	4.20
	DDS + adaptive cost threshold	1120	40	60	4.28
ORTEC	DFS	1686	1240	--	18.75
	DDS	1686	860	--	9.24
	DDS + adaptive cost threshold	1686	300	401	8.78

6.4.3 CP-CG compared with existing approaches in the literature

We finally evaluate our CP-CG approach on the most constrained ORTEC problem. In the literature, 12 instances of the problem of different scheduling periods, i.e. 28, 30 or 31 days of 12 months, have been widely tested. To evaluate the CP-CG as a *general* approach, we first model the problem as of 5-week length and produce 5-week columns in one go for *all* the instances rather than implementing the algorithm for each instance (denoted as CP-CG I). Based on these 5-week columns the real solutions of 28-31 days for each month have been obtained by removing the extra days at the beginning / end of the schedule.

We compare our CP-CG I approach with current existing approaches in the literature on the 12 real-world instances in Table 6.4. In all the other approaches in Table 6.4, meta-heuristic algorithms (e.g. genetic algorithms and Variable Neighourhood Search) have been either delicately designed using domain knowledge to solve the problem [115, 125], or used as a part to improve the solutions obtained by Integer Programming or CP [123, 133]. It is interesting to see that both the hybrid Integer Programming [123] and hybrid CP [133] have combined Variable Neighourhood Search to the solutions obtained from exact mathematical methods in a sequential manner. Our CP-CG approach does not employ any advanced meta-heuristic algorithm sequentially, but embeds the heuristic search DDS more closely within the column generation procedure.

Table 6.4 Existing approaches on ORTEC benchmarks in the literature, best results shown in bold. P.d. stands for the percentage deviations from the best results. The threshold is set as $\tilde{c}$ = 40 in the search of CP-CG. Hybrid GA: a genetic algorithm hybridised with local search. It has been developed in the commercial software HarmonyTM, developed at ORTEC, and was compared with the hybrid VNS in [115]. Hybrid VNS: a hybrid Variable Neighourhood Search (VNS) algorithm. Hybrid IP: IP solutions improved by a VNS with four neighbourhoods. Hybrid CP: a CP approach followed a VNS with two neighbourhoods. Original results in chapter 4 were obtained within 0.5 hours and cited here. CP+LNS: a CP integrated with LNS presented in chapter 5.

instances	**Hybrid GA [125]**		**HybridVNS [115]**		**Hybrid IP[123]**		**Hybrid CP [133]**		**CP+LNS**		**CP-CG I**		**CP-CG II**	
	1 hour		**1 hour**		**1 hour**		**½ Hour**		**½hour**		**1 hour**		**1 hour**	
	Obj	**p.d.**	**Obj**	**p.d.**	**Obj**	**p.d.**	**Obj**	**p.d.**	**Obj**	**p.d.**	**Obj**	**p.d.**	**Obj**	**p.d.**
Jan	775	157%	735	144%	460	53%	616	105%	395	31%	**301**	**0%**	**301**	**0%**
Feb	1791	42%	1866	48%	1526	21%	1736	38%	**1261**	**0%**	**1261**	**0%**	**1261**	**0%**
Mar	2030	19%	2010	17%	**1713**	**0%**	2766	61%	1831	7%	3111	82%	1975	15%
Apr	612	57%	457	17%	**391**	**0%**	956	145%	731	87%	621	59%	621	59%
May	2296	29%	2161	21%	2090	17%	**1786**	**0%**	2111	18%	1941	8%	1941	8%
Jun	9466	52%	9291	49%	8826	42%	8700	40%	**6201**	**0%**	6231	**0%**	6231	**0%**
Jul	781	84%	481	13%	**425**	**0%**	650	53%	751	77%	751	77%	751	77%
Aug	4850	123%	4880	125%	3488	61%	2171	2%	**2121**	**0%**	5901	172%	2171	2%
Sep	615	86%	647	96%	**330**	**0%**	1300	294%	851	158%	1811	449%	401	22%
Oct	736	145%	665	121%	445	48%	616	105%	395	31%	**301**	**0%**	**301**	**0%**
Nov	2126	34%	2030	28%	1613	1%	1620	2%	**1136**	**0%**	2251	42%	1590	40%
Dec	625	54%	520	28%	**405**	**0%**	496	22%	1101	172%	1676	314%	450	11%

By CP-CG I, we obtain the best solution by for 4 out 12 instances (with 0% deviation from the best results). For some particular instances, i.e. Mar, Aug, Sep, Nov and Dec, solutions of CP-CG I are much worse than that of other methods. This is due to the underlying problem characteristics that violations of constraints occur at the boundaries of the scheduling period. As stated above, this CP-CG I generates 5-week columns in one run for all 12 instances. The removing of extra days in the schedule for each month easily leads to violations of constraints which are related to specific weekdays at the boundaries of the month. For example, removing a Sunday shift at the end of 5-week columns may leave a single Saturday shift for the weekend in the current calendar month, thus causes a large cost in the roster.

The above mentioned problem can be easily resolved by generating columns for each individual month. We run CP-CG for those instances (Mar, Aug, Sep, Nov and Dec), and the results are presented under column CP-CG II in Table 6.4. Within the same computational effort, results obtained by the CP-CG II are good, especially considering that the CP-CG approach is built upon the complete hybrid model formulating all constraints, and solution quality does not rely on advanced meta-heuristic algorithns. In our CP-CG approach, no meta-heuristic improvement algorithms have been applied afterwards. The idea is to provide a clear indication of the guaranteed effectiveness of the pure CP-CG approach. Hybridizations of our CP-CG approach with meta-heuristics can be investigated in our future work, and is out of the scope of this chapter.

6.5 Conclusions

In this chapter, we investigate a hybrid column generation approach, where constraint programming (CP) is integrated to solve the highly constrained real-world nurse rostering problems. The work has been submitted to Computers & Operations Research, see List of Publications.

The complex nurse rostering problems have been modeled based on the column generation scheme, where the master problem is formulated as an Integer Program

problem and the pricing subproblem is modeled and solved in the CP paradigm.

In the standard column generation procedure, the quality of columns is only measured by reduced cost. Those columns which satisfy constraints of the pricing subproblem enter the restricted master problem. This usually leads to the slow convergence in the column generation. In this chapter, we propose two strategies which aim to generate good and diverse columns. A cost threshold has been introduced, and is adaptively tightened during the search to choose those columns of good quality, i.e. only columns with a cost below the threshold and is negative reduced cost enter the master problem. Depth bounded discrepancy search have been used in the CP procedure to produce diverse columns. Experimental results demonstrate that a much less number of columns is processed by using DDS compared with DFS. What is more, even with these less columns, the reduction of objective value of linear relaxation by applying DDS in the pricing subproblem is faster than applying DFS. Further speed up of convergence of linear relaxation has also been obtained by applying DDS with the cost threshold.

The effectiveness and efficiency of our CP-CG approach have been demonstrated by a set of comprehensive experiments on three real-world benchmark nurse rostering problems with different profiles. Comparison results against several existing approaches have demonstrated and justified the adopted strategies based on the analysis of the strength of different approaches on the benchmark nurse rostering problems tested.

The main focus of this work is to design efficient search strategies which speed up the Linear Program relaxation convergence while also try to satisfy the integrality request of the master problem. In this work, the Branch-and-Bound search is applied at the root node within the CP-CG approach to produce integer solutions with a certain gap. Given more computational time, our CP-CG may be plugged at each node of the tree to derive optimal integer solutions to the problem, i.e. by using the Branch-and-Price algorithm. Other future research directions include investigations on hybridizations of the CP-CG approach with more advanced search algorithms such as meta-heuristics with problem specific neighourhoods and move strategies, etc., as well as more efficient cost propagation in solving the CP pricing subproblem.

Chapter 6 CP based column generation approach to nurse rostering problems

CP-CG integrates CP with column generation to find solutions to the problems. It not only demonstrates how to solve the problem with exact methods (instead of relying on meta-heuristic algorithms), which makes a valuable contribution to the research community, but also provides a quality measure of the solution obtained. That is, the lower bound of the solution can be obtained from the optimal solution of the LP relaxation. Since the optimal solution of the nurse rostering problem is unknown due to its computational complexity, this quality measure is very important. This lower bound provides us with some knowledge of how far the obtained solution is away from the optimal one.

Chapter 7 A Layered Branch-and-Bound algorithm to portfolio selection problems with real-world constraints

7.1 Introduction

In this chapter and chapter 8, we investigate another real-world combinatorial optimisation problem: the portfolio selection problem.

As we introduced in section 3.3.1, Markowitz's mean–variance model (MV) of portfolio selection is one of the best-known models in modern finance. The basic MV model selects the composition of assets which either achieves a predetermined level of expected return while minimizing the risk, or achieves the maximum expected return within a pre-defined level of risk.

From a practical point of view, however, the MV model is often considered to be too basic, as it ignores many constraints, such as trading limitations and size of the portfolio, etc, faced by real-world investors. Adding such constraints into the basic MV formulation results into a nonlinear mixed integer quadratic programming problem (MIQP), which is considerably more difficult to solve than the basic model.

In this chapter, these real-world constraints are considered *simultaneously* in one single model, which leads to a MIQP model with both binary variables and general integer variables. Comparing with the relevant models [112-114] in the literature, our model consists of all three constraints thus more closely reflects the need of investors. However, this obviously leads to a more complex model than those in [112-114], and thus demands more efficient solution approach for the MIQP model. As observed by many researchers, the efficiency of Branch-and-Bound (B&B) highly relies on the branching rule heuristic (to choose which variable to be branched on) and the node

selection heuristic (to choose which value to be traversed first)[1]. The MIQP model of the portfolio selection problem has different variables of different features. This motivates our work to apply different branching rules and node selection rules according to the different features of these variables.

In this chapter, we try to solve the portfolio selection problems by a multi-level exploitation in the search tree. This proposed layered B&B algorithm can be seen as a decomposition approach, where variables are decomposed according to their different features and layered into certain levels/layers in the search tree. The search is performed layer by layer in the tree. Several benefits can be achieved. Firstly, search will be performed intensively on those variables with a higher priority (at the higher layer). Intuitively, this means we focus on the core variables of problem first, and then deal with the rest of the variables. Secondly, a heuristic which works well for one subset of variables of the problem may not be appropriate for the other variables. By layering the tree (decomposing the variables of the problem), we can easily devise different efficient heuristics to different layers. Thirdly, search is more easily manipulated within the given time limit by aborting it at each layer accordingly. Of course, the optimality of solution will be sacrificed, but the quality of the solution can still be measured by the gap between the incumbent solutions and the optimal solution.

7.2 Problem formulation

In the basic version of the Markowitz MV model, we have a given set of n assets $A = \{a_1, \ldots, a_n\}$. Each asset a_i is associated with an expected return (per period) r_i, and each pair of assets $< a_i, a_j >$ has a covariance σ_{ij}. The covariance matrix $\sigma_{n\times n}$ is symmetric and each diagonal element σ_{ii} represents the variance of asset a_i. A positive value R represents the expected return.

[1] The tightness of the lower bound and the upper bound also plays a key role in B&B, but this issue is not discussed here. We apply the optimal solution of continuous relaxation as the lower bound and the incumbent solution as the upper bound.

Chapter 7 A Layered Branch-and-Bound algorithm to portfolio selection problems with real-world constraints

In the modern MV portfolio theory, the variance of asset represents the risk of investing asset a_i; while the covariance σ_{ij} represents the correlated risks between pairs of assets. To obtain the expected return, rational investors should pick combination of diversified assets, i.e. a portfolio, to reduce the risk which is measured by the covariance of combined portfolios.

A portfolio can be represented by a set $W = \{ w_1, \ldots, w_n \}$, where w_i represents the percentage wealth invested on asset a_i. The value $\sum_{i=1}^{i=n}\sum_{j=1}^{j=n}\sigma_{ij}w_i w_j$ represents the variance of the portfolio, and is considered as the measure of the risk associated with the portfolio [7, 103]. Consequently, the portfolio selection problem can be defined as to minimize the overall variance, while ensuring the expected return R. The formulation of the basic problem can thus be defined as the following.

$$\begin{aligned} &\min \sum_{i=1}^{i=n}\sum_{j=1}^{j=n}\sigma_{ij}w_i w_j \\ &s.t. \sum_{i=1}^{i=n} r_i w_i = R \\ &\sum_{i=1}^{i=n} w_i = 1 \\ &0 \le w_i \le 1, i = 1,\ldots n \end{aligned} \tag{7-1}$$

The covariance σ_{ij} is positive semi-definite, so the minimization problem (7-1) is a convex optimisation problem. More specifically, it is a convex Mixed Integer Quadratic Programming (MIQP) problem.

The quadratic programming problem is NP-complete [142], but nowadays can be solved optimally by using some existing commercial tools[2]. Plotting the risk of a portfolio by solving (7-1) for each corresponding expected return R, we obtain the so-called

[2] There are a lot of state-of-art solvers for the quadratic programming, such as IBM CPLEX, Matlab computational libraries, and NAG's routines [www.nag.co.uk].

unconstrained efficient frontier that provides for each expected return the minimal associated risk.

In our formulation, we consider the following additional three types of constraints.

By using the basic MV model, an obtained optimal portfolio may contain very small investments in a (large) number of assets [7]. However, it is found that such small positions have very limited impact on the overall performance of the portfolio. What's more, in practice, it is quite costly to establish and maintain these small positions [143] due to the cost of tracking, monitoring, and brokerage, etc [7, 143]. In our model, we introduce the buy-in constraint that prevents investors from holding very small positions by defining a prescribed proportion limit w_{min} of the available capital. That is, holding a position strictly less than w_{min} in a portfolio is forbidden. To model such a constraint, we first introduce n extra binary variables z_i, $z_i = 1$ if the investor holds asset a_i (i.e. $w_i > 0$), $z_i = 0$ otherwise:

$$w_i \leq z_i \tag{7-2}$$

Then small positions in portfolios are forbidden by introducing the following buy-in constraint:

$$w_{\min} z_i \leq w_i \tag{7-3}$$

Investors can also put a limit on the number of assets, k, that compose the portfolio, named as cardinality constraint, and expressed as the follows:

$$\sum_{i=1}^{i=n} z_i = k \tag{7-4}$$

Real-world investors usually purchase large blocks of individual financial assets. This is not only because such blocks are more easily traded than the smaller holdings, but also for liquidity reasons, investors want to avoid the risk of getting stuck in a small, poor liquid holding of an asset [7]. Another reason to buy stocks by lots of large quantity is that brokers require a premium for lot trades. So, it is very important to develop an approach that effectively handles the lot constraint within the optimisation procedure. In

our portfolio optimisation model, we define the lots constraint that requires the purchase of assets by batches or lots of M.

To each risky asset a_i, we associate a general integer variable lot_i, and define the following lots constraint:

$$x_i = lot_i M \tag{7-5}$$

imposing that the amount x_i of asset a_i in the portfolio is a multiple of M (a real number). By denoting the face value of asset a_i by p_i and the total available capital by K (a real number), we have $x_i = \frac{w_i K}{p_i}$. By replacing x_i in (7-5) with $\frac{w_i K}{p_i}$, we reformulate (7-5) as the follows:

$$w_i = \frac{p_i lot_i M}{K} \tag{7-6}$$

Problem (7-1) thus becomes:

$$\begin{aligned}
&\min \sum_{i=1}^{i=n}\sum_{j=1}^{j=n}\sigma_{ij} w_i w_j \\
&s.t. \sum_{i=1}^{i=n} r_i w_i = R \\
&\sum_{i=1}^{i=n} w_i = 1 \\
&w_i \le z_i \\
&w_{\min} z_i \le w_i \\
&\sum_{i=1}^{i=n} z_i = k \\
&w_i = \frac{p_i lot_i M}{K} \\
&0 \le w_i \le 1, i = 1,...n \\
&z_i \in \{0,1\}, i = 1,...n \\
&lot_i \in Z^+, i = 1,...n
\end{aligned} \tag{7-7}$$

In model (7-7), three additional constraints are considered simultaneously which leads to a MIQP model with three types of variables, the continuous variables w_i, the binary variables z_i and the general integer variables lot_i.

7.3 The Layered Branch-and-Bound algorithm

7.3.1 The Branch-and-Bound algorithm

As we introduced in section 2.3, The B&B algorithm is a general technique for finding optimal solutions of various optimisation problems, especially in integer and combinatorial optimisation. It systematically enumerates all candidate solutions through a tree search. The main idea of B&B is to prune large subsets of unpromising candidates (branches) by using estimated upper and lower bounds of the objective function to be optimised [8].

The B&B algorithm generally consists of two main procedures [144]. The first one, called *branching*, is a splitting procedure that creates child nodes from the parent node in the tree. The other procedure, called *bounding*, computes the upper and lower bounds of the objective function during the search.

We calculate the lower bound in our convex MIQP by replacing the integer variables with continuous ones. This relaxation of the MIQP is Quadratic Programming with linear constraints and its quadratic matrix is positive semi-definite. Therefore, it is easy to solve with various solution methods (such as extensions of the Simplex method and interior point method, etc). We denote the optimal solution values of this continuous relaxation as $\tilde{x}$.

If $\tilde{x}$ are integer values, then $\tilde{x}$ represents the optimal solution and the problem is solved. Otherwise, those integer variables x_i with non-integer values are chosen for branching. One of the most common ways of branching is to create two subproblems (or nodes) on the floor value $\lfloor x_i \rfloor$ (the largest integer smaller than x_i) and the ceiling value $\lceil x_i \rceil$ (the smallest integer larger than x_i), respectively. These two subproblems are stored in a list of open nodes. Then, at each subsequent iteration of the algorithm, a subproblem is chosen, and the continuous relaxation of the current node is solved, providing a lower

bound. The enumeration at the current node stops if any of the three following conditions are met:

- The continuous relaxation is infeasible (pruning by infeasibility);
- The optimal solution of the continuous relaxation (lower bound) is not better than the value of the best integer feasible solution found so far (upper bound) (pruning by bounds);
- The optimal solution of the continuous relaxation is integer (pruning by optimality).

If the optimal solution of the continuous relaxation cannot be pruned, one of the integer variables with infeasible values is then chosen for branching, and two new subproblems are then added to the list of open nodes. This iterative process in the search tree continues until the list of open subproblems is empty. Fig. 7.1 presents the B&B algorithm.

```
1: Incumbent := ∞ ; Open :={ (P0, ∞ )}; // set the upper bound as ∞ , Open: the list of open nodes
2: Repeat until Open = ∅ ;
3:     Select the node P from Open to be processed; Open := Open \ {P};
4:     Bound P: LBP := g(P)  // function g(P) calculates the lower bound of P
5:         If LBP = f(X) for a feasible solution X and f(X) < Incumbent then
6:                 Incumbent := f(X); Solution := X;
7:                 go to EndBound;
8:         If LBP ≥ Incumbent then prune P
9:         Else Branch on P, generating P1…, Pk;
10:                 Open := Open ⋃ {(Pi; LBP)}, i = 1, …, k;
11:     EndBound;
12: OptimalSolution := Solution; OptimumValue := Incumbent;
```

Fig. 7.1 The Branch-and-Bound algorithm [8]

The pseudo code in Fig.7.1 describes how B&B algorithm exploits the nodes in the tree by the branching and bounding procedures. There are various branching and bounding procedures investigated in research. These branching heuristics [145] and bounding methods [146] are plugged in the B&B algorithm in Fig. 7.1, i.e. line 9 and line 4 correspondingly.

7.3.2 The Layered Branch-and-Bound algorithm

One of the key factors in the B&B procedure is the branching rule, i.e. which variable to branch first. The MIQP model we build contains additional constraints defined by three different groups of variables: continuous variables w_i, binary variables z_i and general integer variables lot_i. To each asset a_i, z_i indicates if the asset is held; lot_i indicates how much to hold. We can thus solve the problem by firstly deciding which assets will be held, and then deciding how much of each asset should be held. This motivates the devise of our proposed layered B&B algorithm.

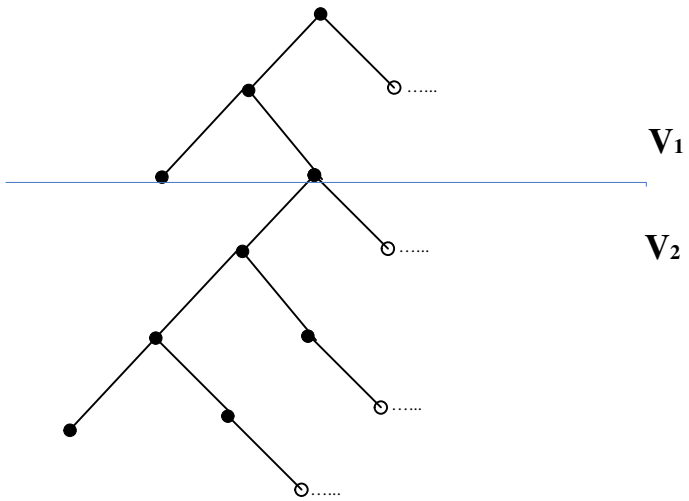


Fig. 7.2 Illustration of layered B&B algorithm. Spots without descendant nodes represent the leaf nodes and circles represent the open nodes as shown in Fig. 7.1

The idea of the layered B&B is that we split the B&B tree into certain layers which correspond to subsets of variables V_1, V_2 …. The variables with the same feature are solved within the same layer of the tree, and different ones are processed in different layers. Fig. 7.2 illustrates this idea, where the first layer consists of the binary variables z_i, and continuous variables w_i. The general integer variables lot_i is added to the second layer. The B&B search will perform on the top layer firstly to instantiated the binary variables z_i, and continuous variables w_i. Based on the instantiations of z_i, and w_i, the

search goes down to the second layer where the general integer variables lot_i are present. By layering the tree, we actually perform an implicit decomposition on the process of solving the original problem. The idea is that we focus on the core variables of the problem first, and perform intensive search on them before we deal with the rest of the variables.

The pseudo-code of the layered B&B is outlined in Fig. 7.3. The first three lines define the branch and node selection heuristics which are applied to different layers of the search tree. Firstly, model V_1 (corresponds to layer one) is built that only consists of binary variables z_i and continuous variables w_i. The problem is solved with the help of branch and node selection heuristics (line 5). When the stopping condition is met, (i.e. a feasible or the optimal solution of problem V_1 is obtained), the search on the top layer is terminated and the incumbent solution is saved. Next, problem V_1 is modified by adding variables on the second layer, the general integer variables lot_i, to build problem V_2. The search is continued on the second layer for problem V_2 with its own heuristics (line 9). Note that the search performed on the second layer does not start from scratch, but is based on the solution obtained from the previous layer (line 8).

There is certain restriction in creating this layered B&B for the problem at hand. The variable in the objective function (i.e. w_i) must be all instantiated in the top layer. This is ensured by building the model V_1 on the top layer on binary variables z_i and continuous variables w_i. After the search on the top layer, the incumbent solution (i.e. value assignments to z_i and w_i) is saved and fed to the second layer to continue the search.

```
1: void L-BranchRule();
2: void L-NodeSelection();
3: void L-RoundHeuristic();

4: model(V1);
5: if (B&B(V1, L-BranchRule(),L-NodeSection()) == feasible or optimal
6:     solutionVector := getIncumbentSolution();
7:     V2 = modify model(V1);
8:     setSolution(solutionVector);
9:     B&B (V2, L-RoundHeuristic())
```

Fig. 7.3 The pseudo-code of the layered B&B algorithm

7.3.3 Branching rules and the node selection heuristic

Finding good branching strategies is the core of successful B&B algorithms and for solving the MIQP [144]. Based on the most popular and common branching rules in MIP, we propose the branching rules employed in our layered B&B for the MIQP problem.

a. The Most Infeasible Branching Rule

The most infeasible branching rule is still a very common branching rule in MIP. It chooses the variable with the largest fraction (closest to 0.5) to branch first. The reason behind this is that we should firstly select a variable for which the least tendency can be recognized. This is similar to choosing the most constrained variable first in solving constraint satisfaction problems.

b. The Pseudocost Branching Rule

Pseudocost branching is a more sophisticated rule proposed in [144]. It estimates the changes of the objective function value caused by fixing the fractional variable to its floor value or ceiling value, which is represented by the following:

$$p_j^- = \frac{\Delta z}{x_j - \lfloor x_j \rfloor}, p_j^+ = \frac{\Delta z}{\lceil x_j \rceil - x_j}.$$

The initial value of p_j^-, p_j^+ for the integer variable x_j is set as the coefficient c_i in the objective function. Different strategies can be used to update p_j^-, p_j^+ if x_j is branched on for more than once. More details of this pseudocost branch rule can be found in [144].

c. Branching Rules in the Layered B&B Algorithm

The most infeasible branching rule focuses on the integer infeasibility of variables. It tries to firstly branch on the variable with the most integral infeasible value. The pseudocost branching rule tries to predict which variable will improve the objective

function value the most. Our first branching rule L-BranchRule() for binary variables z_i combines the most infeasible rule and the pseudocost rule.

We choose the variable with the largest objective coefficient among those with the largest integer infeasibility. The integer infeasibility is measured by the difference between the fractional value x_i and its floor value $\lfloor x_i \rfloor$ or ceiling value $\lceil x_i \rceil$. For example, in the case that we have two fractional values $x_1 = 0.2$ and $x_2 = 0.4$ with their corresponding objective coefficients 0.6 and 0.8, we will choose x_2 to branch first. This branching rule is similar to the static branching rule in [114] that considers the objective coefficient. The difference is that in our study we also take the integer infeasibility into account.

For the general integer variables lot_i, the branching rule L-BranchRule() may not perform so well. For example, assume we have two fractional values $x_1 = 50.8$ and $x_2 = 1245.1$. Because of the different scales, it is difficult to say which one has larger integer infeasibility. We therefore do not branch on these variables but apply a round heuristic to the variable in our L-RoundHeuristic().

In [145] the author reviews most of the existing round heuristics. Experimental results on general MIP problems show that among these round heuristics, the Simple Rounding heuristic is the fastest. The Simple Rounding heuristic looks at each fractional integer variable of a given primal feasible point, and round down or round up the fractional value to its floor or ceiling value. It operates on the fractional integer variables within their feasible domains. Another efficient round heuristic, the ZI round heuristic in [147], attempts to round each fractional integer variable to integer value while using row slacks to maintain the feasibility of the constraints. In this chapter, we propose a L-RoundHeuristic() which combines the Simple Round heuristic and the idea of ZI round. We apply the row slack to measure if a variable can be rounded down or up while maintaining the feasibility of the constraints.

To explain the L-RoundHeuristic() rule, we first define some terms for a MIP in the following form:

$$\begin{aligned} &\min CX \\ &\text{subject to } AX \le B \\ &X \ge 0 \\ &X \in Z \end{aligned}$$

For each constraint $\sum_j a_{ij}x_j \le b_i$, a row slack s_i, defined as $\sum_j a_{ij}x_j + s_i = b_i$, indicates how much the value $\sum_j a_{ij}x_j$ can be changed while maintaining the feasibility of this constraint.

The L-RoundHueristic() first calculates the range that a variable x_j can be changed while maintaining the feasibility of the constraint. x_j cannot be changed to over the bound $\min_i\{\frac{s_i}{a_{ij}}, a_{ij} > 0\}$. Likewise, x_j cannot be changed to below $\min_i\{\frac{-s_i}{a_{ij}}, a_{ij} < 0\}$. After we obtained these bounds, for each variable x_j with a fractional value, we check if its ceiling value $\lceil x_i \rceil$ or flooring value $\lfloor x_i \rfloor$ is within the range defined by the above bounds. If such a variable is found, we then set it to its corresponding $\lceil x_i \rceil$ or $\lfloor x_i \rfloor$ to improve the integer feasibility while maintaining the feasibility of constraints. If such variable cannot be found, we abort the heuristic.

After branching on the variables with fractional values, two subproblems are created and inserted to the list of open nodes. Next, we decide how to choose the node to process in our node selection heuristic L-NodeSelection(). Within our layered B&B, the search is based on the depth first search but with the consideration of integer infeasibility. In the list of open nodes, the node that is the deepest in the tree and with the maximal integer infeasibility is chosen.

The search of the overall layered B&B can be controlled by deciding how much search effort to put on each layer. As presented in line 5 of the pseudo-code of the layered B&B in Fig. 7.3, we can terminate the search on the top layer as soon as we find the first feasible solution. The search then dives into the second layer based on the feasible solution obtained from the top layer. Alternatively, we can terminate the search on the top layer until the optimal solution is found (under the condition that solving the top layer problem is not too time-consuming). The solution obtained when the top layer is solved to optimality may be closer to the optimal solution of the original problem comparing with the solution obtained when the top layer is solved to be feasible. We investigate the performance of the layered B&B under both settings of the stopping criteria in our experimental study.

7.4 Experimental results

7.4.1 Test problems

To build the testing data for our algorithm, we extended the portfolio optimisation instances, publicly available in the OR Library [148] at http://people.brunel.ac.uk/~mastjjb/jeb/info.html, by adding the three additional real life constraints (e.g. cardinality, buy-in threshold and lots) to the benchmark.

Table 7.1 presents the extended five datasets. In the original problem datasets in the OR Library, the expected return rates and covariance matrices of assets have been provided. We set the minimum proportion of the wealth w_{min} to be invested on an asset to 1% for the small instances Hang Seng (Hong Kong) and Dax (Germany), and 0.1% for the rest of the instances. The available capital K is set to 10,000k. The minimum trading amount of asset M is set to 5, 10, and 50 arbitrarily, and the face values p_i for assets are set within the range of 3.50-53.50 arbitrarily without loss of generality. We test different values of k in the cardinality constraint, ranging from 10 to 90 with respect to different sizes of the portfolio. In total, 13 instances have been built to test the algorithm. Note

that the values of K, M and p_i will not affect the difficulty of the problem but the value of cardinality k does affect the selection of portfolios. This will be discussed in the following subsections.

Table 7.1 Properties of problem instances for portfolio selection problems

Dataset	Instances	No. of assets	Minimum proportion w_{min}	Cardinality k
Hang Seng	Port 11	31	0.01	10
	Port 12	31	0.01	28
	Port 13	31	0.01	31
DAX	Port 21	85	0.01	50
	Port 22	85	0.01	65
FTSE	Port 31	89	0.001	60
	Port 32	89	0.001	29
S&P	Port 41	98	0.001	64
	Port 42	98	0.001	80
Nikkei	Port 51	225	0.001	20
	Port 52	225	0.001	60
	Port 53	225	0.001	10
	Port 54	225	0.001	90

7.4.2 Evaluation of the Layered Branch-and-Bound algorithm

In our experiments, we compare the results obtained from the standard B&B algorithm in CPLEX10.0 solver with those from our layered B&B. The layered B&B is implemented in C++ with the concert technology in CPLEX on top of CPLEX10.0. All experiments have been carried out on an Intel(R) Core(TM) 2CPU 1.86GHz machine with 1.97GB memory.

7.4.2.1 Evaluation of the Performance without Layering the B&B Tree

First, we test the branching heuristic L-BranchRule() and the node selection heuristic

L-NodeSelection() without decomposing the problems into layers in the B&B tree. Table 7.2 reports the results obtained from the three solution approaches listed below on the 13 problem instances based on model (7-7).

- Default B&B in CPLEX: all the parameters in B&B are set to default values. B&B itself will choose the node selection and branching direction. We refer this approach as Def hereinafter.
- B&B with our branching rule, without layering the tree: our L-BranchRule() is applied to all the integer variables in the model. We refer this approach as BR hereinafter.
- B&B with our branching rule and node selection heuristic, and without layering the tree. Our proposed L-BranchRule() and L-NodeSelection() are applied to all the integer variables in the model. We refer this approach as BR&NS hereinafter.

The same computational time, 60 seconds, is set for all approaches. For each problem instance and solution approach, Table 7.2 reports:

- The node of the B&B tree at which the first feasible solution is obtained and its corresponding optimality gap.
- The node at which the best feasible solution (optimal solution) is obtained and its corresponding optimality gap.
- The computing time (in CPU seconds) needed to solve the problem to obtain the best solution.

For example, in Table 7.2, for problem instance Port 11 solved by the default B&B in CPLEX, the first feasible solution is obtained at node 20 with a 4.80% optimality gap. The best feasible solution is obtained at node 40 with a 0.00% optimality gap, and the computing time is 0.03 seconds.

First, we comment on which method can find the first feasible solution by traversing the least number of nodes. From Table 7.2 we can see that Def can find the first feasible

solution by exploring the least number of nodes for 6 out of 13 instances, while BR can find the first feasible solution for 4 out of 13 instances by searching the least number of nodes and BR&NS can find 6 out of 13. One thing should be mentioned here is about the feature of instance Port 13. The cardinality constraint k in Port 13 is the same as the total number of assets, which means that the binary variables z_i play no role in the model (all variables z_i will take the value of 1). This has been shown by the fact that all three methods obtain the same results, as all variables take the value of 1 automatically thus our branching rule will not be applied.

We then comment on the quality of these first feasible solutions found by different approaches. It can be seen that although BR finds the first feasible solution later than Def, the quality of the first solution found by BR is better than Def for most of problem instances e.g. Port 1-4. For problem instances Port 5, although Def found the first feasible solution later than BR, the quality is better than that of BR.

Table 7.2 Branch rule and node selection heuristics without layer the B&B tree

Instance	Default CPLEX B&B (Def)					With branch rule (BR)					With branch rule and node selection heuristic (BR&NS)				
	First node	Gap	Best node	Gap	Time	First node	Gap	Best node	Gap	Time	First node	Gap	Best node	Gap	Time
Port 11	**20**	4.80%	**40**	**0.00%**	**0.03**	52	**0.00%**	52	**0.00%**	**0.03**	40	9.02%	4679	0.13%	1.0
Port 12	**19**	**2.20%**	**1267**	**0.00%**	**0.3**	**19**	**2.20%**	1276	0.06%	**0.3**	**19**	**2.20%**	4204	**0.00%**	0.9
Port 13	26	0.00%	26	0.00%	0.02	26	0.00%	26	0.00%	0.02	26	0.00%	26	0.00%	0.02
Port 21	**53**	2.16%	**84**	0.42%	**0.21**	165	**1.86%**	698	**0.30%**	1.09	2927	2.52%	16971	1.01%	60
Port 22	**204**	0.39%	1461	**0.12%**	0.99	434	**0.18%**	**434**	0.18%	**0.55**	222	1.80%	2681	0.22%	2.89
Port 31	**37**	1.83%	**117**	0.04%	**0.26**	82	**0.84%**	184	**0.00%**	0.38	92	12.29%	8777	0.13%	7.45
Port 32	**36**	**4.01%**	**123**	**0.14%**	**0.25**	46	7.12%	2545	0.37%	2.24	50	21%	39708	0.16%	28.4
Port 41	594	**0.02%**	594	**0.02%**	1.30	**81**	6.27%	**205**	0.57%	**0.55**	**81**	6.27%	4591	1.91%	60
Port 42	1019	0.08%	**1019**	0.08%	**2.00**	1511	**0.04%**	1511	**0.04%**	4.29	**247**	0.70%	1590	0.44%	4.02
Port 51	764	2.01%	853	1.08%	**4.50**	776	1.80%	2414	1.01%	16.2	**701**	**0.89%**	**701**	**0.89%**	6.9
Port 52	240	**0.10%**	**240**	**0.10%**	**2.7**	**114**	4.97%	4647	0.30%	18.0	2361	5.96%	9107	5.03%	60
Port 53	26	**0.93%**	**145**	0.33%	**0.6**	**25**	6.77%	499	**0.00%**	1.8	**25**	6.77%	2795	0.27%	4.7
Port 54	344	**3.22%**	**1825**	**0.04%**	**5.3**	5615	7.78%	13273	1.58%	60	**227**	5.01%	9619	0.09%	30

Comparing the best solution (the optimal solution with a certain optimality gap) found by these three methods, we can see that Def can find the best solution more quickly as well as with a better quality.

Considering all the factors on how fast the first and the best solution can be found and how good these solutions are, we can conclude that the performance of BR and BR&NS is not consistently good. This means that our proposed branching rule which is designed to certain type of variables does not work efficiently for all of the variables in the model.

7.4.2.2 Evaluation of the Layered B&B

In this subsection, we apply the branching rule and node selection heuristics in the layered B&B where the tree has been layered according to different groups of variables.

Table 7.3 reports the results of Def and the layered B&B on the 13 problem instances as follows:

- Def: Default B&B in CPLEX, same as the above
- Layered B&B: our proposed layered B&B with L-BranchRule() and L-NodeSelection() applied to binary variables z_i and L-RoundHueristic() applied to general integer variables lot_i

The evaluation criteria reported in Table 7.3 are the same as those in Table 7.2, i.e. the node at which the first and best solutions obtained and their corresponding quality.

As we introduced in section 7.3, in addition to applying different branching rules and heuristics to different groups of variables, we can also easily control the search by putting appropriate search effort on different layers of the tree. In Table 7.3, the layered B&B is terminated after the first feasible solutions are found at the top layer.

Table 7.3 Layered B&B, search aborts after finding the first feasible solution at the top layer

Instance	Default B&B					Layered B&B (feasible at the top layer)				
	First node	Gap	Best node	Gap	Time	First node	Gap	Best node	Gap	Time
Port 11	20	4.80%	40	**0.00%**	**0.03**	164	**1.02%**	164	1.02%	0.1
Port 12	19	2.20%	1267	0.00%	0.3	19	2.20%	1267	0.00%	0.3
Port 13	26	0.00%	26	0.00%	0.02	26	0.00%	26	0.00%	0.02
Port 21	53	**2.16%**	84	**0.42%**	**0.21**	2136	4.52%	2136	4.52%	3.82
Port 22	204	0.39%	1461	**0.12%**	**0.99**	3443	**0.26%**	3443	0.26%	1.99
Port 31	37	1.83%	117	0.04%	**0.26**	143	**0.00%**	143	**0.00%**	0.88
Port 32	36	4.01%	123	**0.14%**	**0.25**	45	**3.63%**	45	3.63%	0.30
Port 41	594	**0.02%**	594	**0.02%**	1.30	46	0.14%	46	0.14%	**0.90**
Port 42	1019	**0.08%**	1019	**0.08%**	2.00	24	0.22%	24	0.22%	**0.80**
Port 51	764	2.01%	853	1.08%	4.50	21	**0.06%**	21	**0.06%**	**0.40**
Port 52	240	**0.10%**	240	**0.10%**	2.7	249	3.78%	249	3.78%	**2.0**
Port 53	26	0.93%	145	**0.33%**	0.6	0	**0.56%**	0	0.56%	**0.1**
Port 54	344	3.22%	1825	**0.04%**	5.3	127	**0.11%**	127	0.11%	**1.3**

We highlight the first feasible solutions obtained by two methods in Table 7.3. It can be seen that the layered B&B can obtain the first feasible solutions with better quality for 7 out of 11 instances. This indicates that the layered B&B with our proposed heuristics can find better first feasible solutions than Def. This is quite satisfied result because we devote a much less search effort on the layered B&B (i.e. the search is terminated as soon as the first feasible solution is found). For the same reason, not surprisingly, Def obtains better quality optimal solutions than the layered B&B. In Table 7.3, the same performance is achieved for two instances Port 12 and Port 13 due to the properties of these instances. The cardinality constraint k in Port 13 is the same as the total number of assets, which means that the binary variables z_i play no role in the model (all variables z_i will take the value of 1). This is shown by the fact that Default B&B and Layered B&B obtain same results. Instance Port 12, the cardinality constraint k is 28, which is very close to the size of portfolio 31. The Layered B&B can produce the result at the top layer of the tree without applying L-RoundHueristic() to the general integer variables lot_i. Therefore, the results obtained by Layered B&B are the same as those obtained by Default B&B.

More search effort can be put on the top layer of the layered B&B to achieve improved best solutions. That is, the subproblem on the top layer can be solved to optimal before the search dives into the second layer. The optimal solution from the top layer is saved, based on which the search on the second layer is performed. From the results reported in Table 7.4, we can see that the layered B&B found better solutions in term of optimality gap for all instances except Port 12 and Port 31 (we set the optimality gap tolerance as 0.01%).

Note that when the initial solutions are fed to the B&B, CPLEX will apply repairing heuristics on the solutions to derive feasible solutions before starting the tree search. In Table 7.4, we can see that the repairing heuristic succeeds for 2 problem instances (Port 21 and Port 22) thus the tree search does not need to continue to layer two.

Table 7.4 Layered B&B. search aborts after obtaining the optimal solution

Instance	Layered B&B (feasible top layer)					Layered B&B (optimal top layer)						
						Heuristic repair succeed		Heuristic repair failed				
	First node	Gap	Best node	Gap	Time	Node	Gap	First node	Gap	Best node	Gap	Time
Port 11	164	1.02%	164	1.02%	**0.1**	--	--	67	**0.19%**	80	**0.00%**	0.15
Port 12	19	2.20%	1267	**0.00%**	**0.3**	--	--	0	**0.01%**	0	0.01%	0.45
Port 13	26	0.00%	26	0.00%	**0.02**	--	--	0	0.00%	0	0.00%	0.03
Port 21	2136	4.52%	2136	4.52%	3.82	0	**0.00%**	--	--	--	--	**0.2**
Port 22	3443	0.26%	3443	0.26%	1.99	0	**0.22%**	--	--	--	--	**0.32**
Port 31	143	**0.00%**	143	**0.00%**	0.88	--	--	20	0.20%	20	0.20%	**0.87**
Port 32	45	3.63%	45	3.63%	**0.30**	--	--	30	**0.32%**	30	**0.30%**	0.50
Port 41	46	**0.14%**	46	0.14%	**0.90**	--	--	50	0.87%	190	**0.04%**	1.90
Port 42	24	**0.22%**	24	0.22%	**0.80**	--	--	30	0.68%	70	**0.14%**	1.20
Port 51	21	0.06%	21	0.06%	**0.40**	--	--	0	**0.00%**	0	**0.00%**	0.7
Port 52	249	3.78%	249	3.78%	2.0	--	--	0	**0.01%**	0	**0.01%**	**1.7**
Port 53	0	0.56%	0	0.56%	**0.1**	--	--	0	**0.14%**	0	**0.14%**	1.1
Port 54	127	0.11%	127	0.11%	**1.3**	--	--	0.	**0.10%**	0	**0.10%**	1.4

Based on the results presented in Tables 7.2, 7.3 and 7.4 we can draw the following conclusions:

- The branching rule and node selection heuristic do play an important role on the performance of B&B algorithms
- Applying the branching rule and node selection heuristic on different types of variables at different layers of the tree greatly improves the solutions obtained
- The search can be more easily controlled in the layered B&B
- Based on the solutions of subproblems, better or optimal solutions can be easily obtained within similar computational time compared with standard B&B.

7.4.3 The efficient frontier

In this section, we compare of the performance of the standard B&B and layered B&B algorithms by examining their performance on obtaining the efficient frontiers of the five problems. We plot the efficient frontiers for the five instances Port 11, Port 21, Port 31, Port 41 and Port 51. For each instance, we have computed the mean-variance frontier by setting the expected return from 0.2% to 0.8% with the step of 0.012 (50 portfolios are thus plotted for each efficient frontier). Linear interpolation is used to plot the intermediate values.

To obtain the exact efficient frontier for the constrained problem, model (7-7) needs to be solved optimally for each expected return R. However, as we stated before, this is not achievable due to the introduction of the integer constraints. We therefore use the CPLEX 10.0 solver to compute an approximate efficient frontier for model (7-7) by running the algorithm for a long computational time. For each point on the efficient frontier, the running time limit is set as 60 seconds. We denote this as the default efficient frontier.

Fig. 7.4 illustrates the default efficient frontiers calculated by the default CPLEX solver with a long running time, and the efficient frontiers obtained from the layered B&B.

Here, in the layered B&B, we solve the subproblem at the top layer to optimal before going down to the second layer. The time spent on each point on the efficient frontier is given in Table 7.4, i.e. around 0.02 to 2 seconds for each point. This is much less than the time limit for the default CPLEX solver which is 60 seconds.

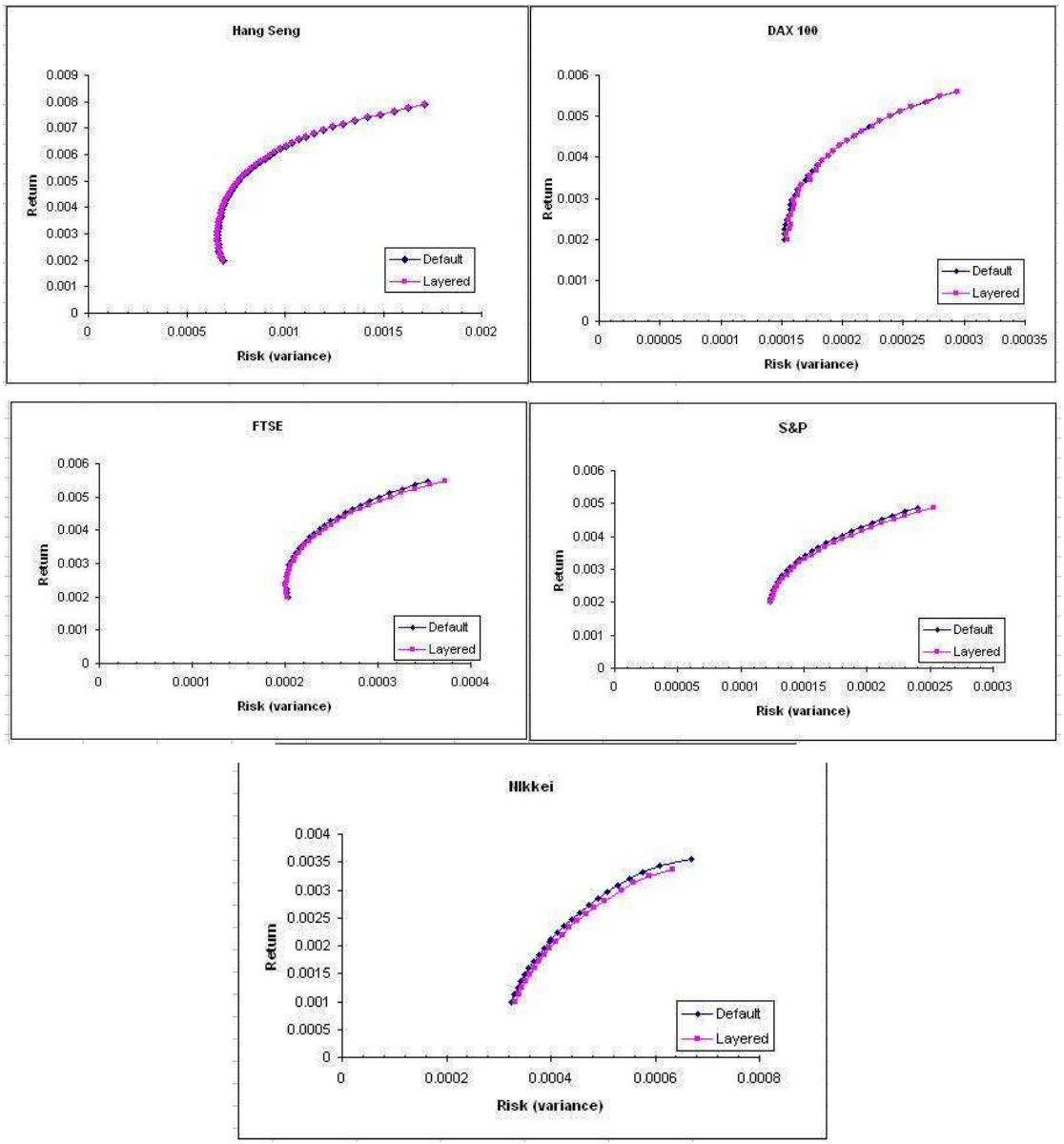


Fig. 7.4 Efficient frontiers from the default B&B and layered B&B

From Fig. 7.4 we can see that the quality of solutions from the layered B&B is extremely good. For instance Hang Seng, the average gap between the default efficient frontier and our layered B&B efficient frontier is under 0.01%. For instance DAX, the

gap for portfolios with lower risk is larger than those of with higher risk, but the overall gap is still quite small. The gaps for instances FTSE and S&P are quite small as well. However, for large instances Nikkei, the gap is relatively larger.

Due to the additional constraints introduced in our problem formulations, it is difficult to compare the results against those from other approaches in the literature. Even for approaches that are tested on the same OR Library instances, a fair comparison is still difficult to conduct as, to the best of our knowledge, all the existing approaches in the literature analyze the results of problems with only a single additional constraint. Our approach considers the model with all three constraints simultaneously for the first time.

In order to more accurately compare our results against the default frontier obtained by B&B with a long running time, we compute the percentage deviations of each portfolio from the default frontier by calculating the distance between the risk obtained by our layered B&B algorithm and that on the default frontier. This measure represents the deviation of the obtained solution from an approximation of the exact solution, and is also used in [109] [112] to evaluate the quality of results and provide an indication of the performance of the algorithm. Table 7.5 presents the comparisons between heuristic approaches.

In Table 7.5, the genetic algorithm, tabu search and simulated annealing methods investigated in [109] are pure metaheuristic methods. Pooled (GA, TS, SA) method, also investigated in [109], combines the three sets of non-dominated points given by the three algorithms into one set. Those points which are dominated are eliminated from this new set. In [112], the integer restart method applies the previous integer solution of QP as the first feasible solution and the upper bound of the following QP.

In Table 7.5, the percentage deviations of our method are quite small compared with other approaches. One thing to note is that the problems solved by the integer restart method are formulated in three independent models; each consisting of one of the three additional constraints investigated in this chapter, while our model consists of all the

three additional constraints. Compared with these simpler models, the percentage deviations of our solutions are still highly competitive.

Table 7.5 Comparisons of the layered B&B with existing approaches in the literature

Instance	Solution method	Percentage deviation
Hang Seng	GA	0.94570
	TS	0.99080
	SA	0.98920
	Pooled (GA,TS,SA)	0.93320
	Integer restart	0.01415
	Layered B&B	0.00008
DAX	GA	1.9515
	TS	3.06350
	SA	2.42990
	Pooled (GA,TS,SA)	2.19270
	Integer restart	0.01399
	Layered B&B	0.00992
FTSE	GA	0.87840
	TS	1.39080
	SA	1.13440
	Pooled (GA,TS,SA)	0.77900
	Integer restart	0.01141
	Layered B&B	0.02074
S&P	GA	1.71570
	TS	3.16780
	SA	2.69700
	Pooled (GA,TS,SA)	1.31060
	Integer restart	0.01586
	Layered B&B	0.03079
Nikkei	GA	0.6431
	TS	0.9891
	SA	0.6370
	Pooled (GA,TS,SA)	0.5690
	Integer restart	0.00618
	Layered B&B	0.03901

7.5 Conclusions

In this chapter, we extend the Markowitz mean-variance portfolio selection problems with three additional real-world trading constraints *simultaneously* in a single model. The resulting formulation, which is a Mixed Integer Quadratic Programming problem, thus has different features corresponding to different groups of integer variables (i.e. binary variable and general integer variable). These features motivate the development of a decomposition approach, layered Branch-and-Bound (layered B&B) algorithm, for solving the problem which we are concerned with. The work has been submitted to Journal of Heuristics, see List of Publications.

In the B&B search tree, sets of variables are layered (decomposed) according to their different features, and search is performed on one layer before another in sequence. Two tailored branching heuristics and one node selection heuristic are applied to individual layers of the B&B tree in order to speed up the search for optimal solutions. The performance of the layered B&B is analyzed and compared based on the extended instances in the OR Library with all three additional constraints. The efficient frontiers are plotted for each instance to provide a graphic illustration of the results. It can be seen that the quality of solutions from the layered B&B algorithm is extremely good, with a much less computational time, compared with the default B&B.

The layered B&B algorithm can be seen as firstly searching on the top layer of the tree (subproblem of a set of variables) then *diving* into a particular region of the search space in order to explore it intensively. In this chapter, both layers are searched by the B&B, and with different tailored branching rules to the corresponding layer, i.e. these branching rules are tailored explicitly to the features of the variables.

The layered B&B for the portfolio selection problems with integer constraints is proposed based on the features of different variables in the problem, i.e. the B&B tree is layered to multi-levels accordingly to the binary variables and general integer variables.

For problems without this specific feature, the layered B&B technique can still be generalized and applied to solve the problem accordingly.

One possible generalization of the layered B&B is to apply it in solving problems represented as decomposed constraint graphs. Constraint graph has been widely used in the literature, especially in constraint satisfaction, where nodes represent variables and edges represent constraints. By decomposing the constraint graph (e.g. using the clique partition), the problem can be partitioned into subproblems (e.g. cliques). Search can then be applied on each layer corresponding to each subproblem. This generalised framework of layered B&B provides the possibility of applying different search methods on each layer. It is not necessary to perform the B&B on all layers. In the next chapter, different algorithms including local search will be introduced into the layered B&B algorithm, with the exact B&B search, to solve different decomposed subproblems in the layered tree.

Chapter 8 A hybrid local search and Branch-and-Bound approach to constrained portfolio selection problems

8.1 Introduction

In this chapter, we consider the single-period portfolio selection problem, taking into account transaction costs and a set of trading constraints. In the problem which we are concerned with, a number of transactions can be carried out to adjust the portfolio during the trading period. The goal of the problem is to minimize the risk of the adjusted portfolio with the presence of transaction costs, while satisfying a set of trading constraints in feasible portfolios.

If the transaction cost function is linear, then the problem is generally easy to solve. However, a function which better reflects realistic transaction costs is usually non-convex [149]. Some research show that realistic transaction costs usually include a fixed fee, and thus the cost is relatively higher when the amount of transaction is smaller [149, 150]. The transaction cost is thus usually represented by linear piecewise concave function, leading to non-convex optimisation problems which are more difficult to solve.

A common approach to handle a linear piecewise concave cost function is to introduce a number of additional binary variables and solve the resulting mixed Integer Programming (MIP) [151]. However, due to the introduction of these additional variables, the problem becomes larger and is difficult to solve when the portfolio consists of a large number of assets.

There has been a considerable progress recently in the development of exact methods for solving general MIP. To certain extent, this progress is mainly due to the integration of heuristics into the B&B method. There are a large variety of heuristics proposed for

general MIP in the literature (see [8, 152, 153]). The heuristics can be applied as branching rules and node selection heuristics, etc, to accelerate the search in the tree [145, 154, 155]. Heuristics can also be applied as improvement methods which start from one or more feasible solutions and try to find feasible solutions with a better objective value [49, 156-158].

In this chapter, we propose a new hybrid approach which integrates local search into the B&B algorithm to solve the non-convex portfolio selection problem *heuristically*. In the integrated B&B, we propose a new branching scheme which applies local search. We thus name this branching scheme *local search branching*. Instead of branching on a single variable, the local search branching scheme branches on a set of core binary variables of the problem iteratively to generate a sequence of subproblems. The subproblems are then solved in sequence by the default B&B in a general solver. The best solution among them approximates optimal solution to the original problem.

Our main contribution is the tight integration of local search to B&B. The idea is to identify a set of core variables in the problem, perform computationally inexpensive search on the *surface* of these core variables, and then explore the subproblems defined by variable fixings to completion. The inherent similar structures of the subproblems facilitate efficient and successful solution information reusing in solving the subproblems. It is well known that the sooner a tight upper bound can be found, the more efficient a B&B search will be [8]. Therefore, Local Search Branching B&B search can be further improved by a heuristic which identifies the subproblem who has a tight upper bound.

8.2 Problem description

8.2.1 Portfolio selection problem with transaction cost

In financial practice, the transaction cost has significant effects on the optimal portfolio in portfolio selection. It has been shown in [159] that ignoring the transaction cost

results into inefficient portfolios. This has also been justified by the experimental study in [160].

The linear transaction cost function, which leads to a convex optimisation problem, is relatively easy to solve, while the non-convex optimisation problem is more challenging. Some researchers [106, 149-151, 160-162] investigate the portfolio selection problem with non-convex transaction cost functions under the linear risk measure. In [151], the piecewise linear concave cost function is approximated by the convex envelop function, and the convex optimisation problem is solved by B&B. In [161, 162], the reduced costs of assets are used as a heuristic to reduce the mixed integer Linear Program model. Assets with the highest reduced cost are selected to construct a reduced problem which is solved by B&B. In [106], the reduced cost of the model is also used as a heuristic measure to decompose the problem into subproblems. The assets with reduced costs which are under a certain threshold are selected to construct a subproblem P_1 and the rest of them forms the other subproblem P_2. The subproblems are them solved separately by B&B.

Solution approaches to the portfolio selection problem with transaction costs where covariance is used as risk measure are less applied than those using linear risk measures. To the best of our knowledge, there are only limited research attention on the problem with non-convex cost function based on the MV model [163, 164]. In [163], a Lagrangian relaxation is applied to derive the lower bound in B&B. In [164], a convex envelop cost function and an iterative heuristic method are applied to approximate the non-convex cost function and solve the problem by B&B.

In this chapter, we investigate the MIQP portfolio selection problem based on the MV model. The quadratic term in the objective function (i.e. covariance) makes the problem a Quadratic Program, which can be easily solved by any current solver (through extended Simplex, etc.) So the difficulty of the MIQP problem lies on the discrete aspect of the problem. In this chapter, we thus focus on the aspects of algorithm design and performance analysis of our proposed new local search branching B&B to tackle

the MIQP problem. More specifically, we focus on the discrete optimisation aspects of the problem, instead of a theoretical study on risk measures, which is not in the scope of this work.

8.2.2 Problem formulation

Consider that an investor is holding a portfolio that consists of a set of n assets. To respond to changes in the market, the investor must review its current portfolio, with the view to carry out a number of transactions. It is assumed that the new portfolio will be held for a fixed time period. The investor's goal is to minimize both the transaction costs occurred and the risk of the assets in the portfolio at the end of the investment period, while satisfying a set of constraints. These constraints typically include meeting the target return, the minimum position size (bounds on the amount of each asset), and the minimum trading size (bounds on the amount of the transaction occurred on each asset).

Let w_i be the percentage of capital invested in asset i. We shall use a weight vector $w^0 = (w_1^0, w_2^0, ..., w_n^0)^T$ to denote a portfolio. The amount transacted in each asset is specified by $x = (x_1, x_2 \ldots, x_n)^T$, $x_i < 0$ means selling and $x_i > 0$ means buying. After the transaction, the adjusted portfolio is $w = w^0 + x$, and is held for a fixed period of time. At the end of the investment period, we denote the return of asset i at the end of the investment period as r_i. We denote its variance in return as σ_{ii}. We further define $\phi(x)$ as the sum of individual transaction costs associated with each x_i and σ_{ij} as the covariance between assets i and j. Based on the basic MV model, the portfolio selection problem with transaction costs can thus be modeled as follows:

$$\min \sum_{i=1}^{i=n} \sum_{j=1}^{j=n} \sigma_{ij} w_i w_j + \phi(x) \tag{8-1}$$

$$s.t. \sum_{i=1}^{i=n} r_i w_i = R \tag{8-2}$$

$$w = w^0 + x \in F \tag{8-3}$$

where objective (8-1) is to minimize the risk of the portfolio and the transaction costs incurred. F in (8-3) represents the set of feasible portfolios subject to all the related constraints detailed next.

The transaction cost

The transaction cost is the sum of the transaction cost associated with each asset traded:

$$\phi(x) = \sum_{i=1}^{i=n} \phi_i(x_i)$$

As shown in [149, 150, 160], in practice, the transaction cost usually includes a fixed fee, and the function can be linear piecewise concave as the cost decreases relatively when the trading amount increases. In this chapter, we consider a model that includes a fixed fee plus a linear cost, thus leads to a non convex function. Our method is readily extendable to handle more complex transaction costs. The fixed fee charged for buying and selling asset i is denoted as β_i^+ and β_i^-, and the cost rates associated with buying and selling asset i are denoted as α_i^+ and α_i^-. The transaction cost function is given in (8-4), and shown in Fig. 8-1:

$$\phi_i(x_i) = \begin{cases} 0, x_i = 0; \\ \beta_i^+ + \alpha_i^+ x_i, x_i > 0; \\ \beta_i^- - \alpha_i^- x_i, x_i < 0; \end{cases} \tag{8-4}$$

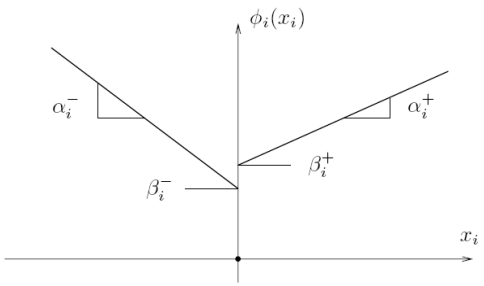


Fig. 8.1 The transaction cost function

As we stated before, the amount transacted in each asset is specified by $x = (x_1, x_2 \ldots, x_n)^T$. $x_i < 0$ means selling and $x_i > 0$ means buying. To denote the transaction, we

introduce new variables $x_i^{buy} \geq 0$ and $x_i^{sell} \geq 0$, and thus $x_i = x_i^{buy} - x_i^{sell}$. We assume that the investor does not invest additional capital during the transaction process, i.e. we have constraint

$$\sum_{i=1}^{i=n} w_i + \sum_{i=1}^{i=n} \phi_i(x_i) = 1 \tag{8-5}$$

Trading constraints

The minimum position constraint prevents investors from holding very small positions by introducing a prescribed proportion $w_{\min}$ of the available capital. That is, holding a position strictly less than $w_{\min}$ is forbidden. To model such constraint, we first introduce n extra binary decision variables z_i^{hold} to indicate if an asset is held or not. z_i^{hold} =1 if the investor hold asset i (i.e. $w_i > 0$), $z_i^{hold} = 0$ otherwise. The below constraint is firstly introduced:

$$w_i \leq z_i^{hold} \tag{8-6}$$

Then small investments can be forbidden by introducing the following constraint:

$$w_{\min} z_i^{hold} \leq w_i \tag{8-7}$$

Investors can also put a limit on the number of assets, k, that compose the portfolio, named as cardinality constraint. It can be expressed as the following:

$$\sum_{i=1}^{i=n} z_i^{hold} = k \tag{8-8}$$

In addition, the minimum trading constraint is also used to prevent investors from trading very small amount of assets by $x_{\min}$. Same as the above way of modelling the minimum position constraint, we introduce additional binary variables z_i^{buy} and z_i^{sell} to represent the buying or selling of the corresponding asset *i*. The minimum trading constraint can then be expressed as the follows:

$$x_{\min} z_i^{buy} \leq x_i^{buy} \tag{8-9}$$

$$x_{\min} z_i^{sell} \leq x_i^{sell} \tag{8-10}$$

We can add additional constraints to define the relation between binary variables and continuous variables as the follows:

$$x_i^{sell} \le z_i^{sell} \tag{8-11}$$

$$x_i^{buy} \le z_i^{buy} \tag{8-12}$$

We also need the exclusive constraint to prevent buying and selling the same asset at the same time:

$$z_i^{buy} + z_i^{sell} \le 1 \tag{8-13}$$

Problem model with transaction cost and trading constraints

With these additional constraints which define a feasible portfolio *F*, we have the complete problem model (PSP) as follows:

i the number of assets

Parameter

w^0	Current position of portfolio
σ_{ij}	Covariance between *i* and *j*
R	Expected return
β_i^+, β_i^-	Fixed cost for buying or selling asset *i*
α_i^+, α_i^-	Variable cost rate for buying or selling asset *i*
$w_{\min}$	Minimum hold position
k	Number of assets in portfolio after transaction

Variable		Feature	Type	Domain	Core variable
w_i	Position of portfolio after transaction	Decision variable	Continuous	[0,1]	No
x_i^{buy}	Amount of buying asset *i*	Decision variable	Continuous	[0,1]	No
x_i^{sell}	Amount of selling asset *i*	Decision variable	Continuous	[0,1]	No
z_i^{hold}	Hold asset *i* or not	Dependent variable	Binary	{0,1}	Yes
z_i^{buy}	Buy asset *i* or not	Dependent variable	Binary	{0,1}	No
z_i^{sell}	Sell asset *i* or not	Dependent variable	Binary	{0,1}	No

$$\min \sum_{i=1}^{i=n}\sum_{j=1}^{j=n}\sigma_{ij}w_i w_j + \sum_{i=1}^{i=n}\phi_i(x_i) \qquad \text{PSP}$$

$$s.t.$$

$$w_i = w_i^0 + x_i^{buy} - x_i^{sell}$$

$$\sum_{i=1}^{i=n} r_i w_i = R$$

$$\sum_{i=1}^{i=n} w_i + \sum_{i=1}^{i=n}\phi_i(x_i) = 1$$

$$w_i \le z_i^{hold}$$

$$w_{\min} z_i^{hold} \le w_i$$

$$\sum_{i=1}^{i=n} z_i^{hold} = k$$

$$x_i^{buy} \le z_i^{buy}$$

$$x_{\min} z_i^{buy} \le x_i^{buy}$$

$$x_i^{sell} \le z_i^{sell}$$

$$x_{\min} z_i^{sell} \le x_i^{sell}$$

$$z_i^{buy} + z_i^{sell} \le 1$$

$$0 \le w_i \le 1, i = 1,...n$$

$$0 \le x_i^{buy} \le 1, i = 1,...n$$

$$0 \le x_i^{sell} \le 1, i = 1,...n$$

$$z_i^{hold}, z_i^{buy}, z_i^{sell} \in \{0,1\}, i = 1,...n$$

There two group of variables in the formulation of the problem, as denoted by the "feature" column. w_i , x_i^{buy} , x_i^{sell} are decision variables. z_i^{hold} , z_i^{buy} , z_i^{sell} are dependent variables which are used to formulate the constraints. The column "core variable" denotes which decision variables are core variables. We will describe it in the following sections.

8.3 Related work of hybrid local search with B&B

The B&B algorithm is an exact method to find optimal solutions for various optimisation problems, especially in integer and combinatorial optimisation [8].

Recently, there have been successive improvements in B&B, such as Branch-and-Cut [165], Branch-and-Price [166] and Lagrangian relaxation [8], etc. However, many large-scale MIP problems still cannot be solved within a reasonable time limit by these exact methods. Consequently, heuristics and local search have attracted great research attention as possible complements to the exact methods.

The simplest method which integrates heuristics and exact methods is to employ heuristics as branching rules, node selection heuristics and cut-off bound to improve the B&B tree search. This showed to significantly improve the efficiency of current B&B algorithms.

More closely integrated approaches apply the idea of local search within B&B [49, 156, 158]. These researchers have tried to bring the idea of local search to B&B. More specifically, the way that local search explores the neighbourhood of a solution can be adapted in B&B to explore the nodes of the tree. In this section, we review several main and important success in the area, including the local branching [156], relaxation induced neighbourhood search [157] and several other related approaches.

8.3.1 Local Branching

Local branching [156] is a branching strategy for exploring an explicit neighbourhood of a MIP solution. The idea of local branching is based on the observation that the neighbourhood of an integer feasible solution often contains valid or better solutions of the problem.

The neighbourhood x' of a current incumbent x can be defined by introducing an additional constraint $H(x, x') < r$, where r represents a neighbourhood radius parameter, and $H(x, x')$ is a generalized notion of Hamming distance, $H(x, x') = \sum_{j \in B} \left| x_j - x'_j \right|$. In another word, local branching defines the neighbourhood of x by adding a linear constraint $H(x, x') < r$. All the solutions which satisfy this constraint

are the neighbourhood of x. This can be also seen as a soft variable fixing [49], and the neighbourhood is actually a sub-MIP. Local branching combines local search with a generic MIP solver for solving exactly the MIPs.

8.3.2 Relaxation Induced Neighbourhood Search (RINS)

Another innovation which brings the idea of local search to explore the neighbourhood in MIP is Relaxation Induced Neighbourhood Search (RINS) [157]. RINS defines and explores the neighbourhood in MIP based on two solutions: the incumbent solution and the solution of continuous relaxation.

An incumbent solution during search is feasible with respect to the integerality constraint but it is often not optimal until the global optimal integer solution has been found. On the other hand, a solution of continuous relaxation at the current node is very often not an integer solution, but its objective value is always better than or the same as that of the incumbent [157]. Thus, the incumbent solution and the solution of continuous relaxation each achieves one and fails the other of the following conflicting goals: integrality and optimisation of the objective value. While some variables clearly take different values in the solutions of the incumbent and relaxation, it is important to note that many take the same values, as observed in [157]. The analysis of these two solutions showed that a small neighbourhood of the incumbent is likely to contain better feasible solutions.

Based on the above idea, the RINS strategy is thus simple. At a node of the B&B tree, RINS performs the following procedure: (1) fix the values of the variables which are the same in the current continuous relaxation and the incumbent integral solution; (2) set the objective cut-off value (bound) to the objective value of the current incumbent solution; and (3) solve the sub-MIP on the remaining variables.

8.3.3 Other approaches

Recently, some approaches have emerged following the similar idea of applying local search to approximately explore the nodes of B&B. Variable Neighourhood Search Branching (VNSB) [158] is a heuristic search for solving the general MIP problem, using the general-purpose MIP solver CPLEX. It can be seen as a special variant of local branching. Compared with local branching, in VNSB, the neighbourhoods are changed according to the rules of the general VNS in a more systematic manner.

Variable Neighbourhood Decomposition Search (VNDS) [49] is a hybrid heuristic for solving MIP. It combines Variable Neighourhood Search with a general-purpose MIP solver, e.g. CPLEX in [49]. VNDS also performs systematic variable fixing, and can be seen as an improved version of VNSB. The variables to be fixed are chosen based on a non-decreasing order of the difference between their values in the LP-solution and the incumbent solution, i.e. they are chosen according to the distance of their values to those in the corresponding linear relaxation solution. Subproblems are obtained by successively fixing a certain number of variables in the order obtained. The subproblem thus consists of the remaining free variables (uninitiated variables) which are the furthest from their linear relaxation values. These subproblems are then solved exactly or within the CPU time limit by B&B in CPLEX.

8.4 Local search branching B&B algorithm

8.4.1 Framework of Local search branching B&B

In this chapter, we propose a new hybrid approach, named Local Search Branching B&B, the pseudo-code of which is presented in Fig. 8.2. Instead of branching on just one variable at a time (as it is done in standard branching schemes) to create two subproblems, the local search branching scheme branches on a set of variables at the same time.

First, in the initialization phase of the approach (line 1), the original problem is decomposed into subproblems by variable fixing on the core variables z_i^{hold} . Variables of a subset of z_i^{hold}, *S*, are assigned values 1. The exclusive set of *S*, denoted as *S'*, comprises the rest of variables. The variables in *S'* are assigned values 0 (see section 8.4.3 and pseudo code in Fig. 8.3). Next, the lower bound of the subproblem P_{subi} is computed by a general LP solver, which relaxes the subproblem to be a continuous problem (line 3). The default B&B algorithm in the MIQP solver in CPLEX10.0 is applied to solve the subproblems to optimality. The objective value of the feasible solution to the concerned subproblem P_{subi} serves as the upper bound of the original problem. The objective value of the optimal solution to the relaxed continuous subproblem P_{subi} serves as the lower bound (line 3). If the lower bound of a subproblem is above the current upper bound found so far, we can prune this subproblem during the search (line 4). Otherwise, these subproblems are solved exactly by a standard B&B in an IP solver (line 7). The solutions to the subproblems together with the assignments of core variables constitute feasible solutions to the complete original problem (line 7).

Local search is next performed on this set of core variables to generate new value assignment for each variable z_i^{hold} (line 9) (see section 8.4.4 and pseudo code in Fig. 8.4). The elements of subsets *S* and *S'* are updated (line 10). Each move of the local search updates the subsets *S* and *S'* thus a sequence of subproblems is created (line 11). These subproblems are solved in sequence and the best solution among them approximates the optimal solution to the original problem (line 13). The whole procedure terminates by terminating the local search on the z_i^{hold}. Therefore, the search is an incomplete search. It cannot guarantee optimality of the solution due to the nature of the local search on core variables z_i^{hold}.

Each of the subproblems itself is still a MIQP problem due to the presence of binary variable z_i^{buy} and z_i^{sell}. However, due to the assignment to variable z_i^{hold} by the variable fixing (see section 8.4.2), the size of the subproblem is much smaller compared to the original one. Therefore, subproblems can be handled by the default B&B.

What is more, the inherent similar structures of the subproblems enable a very successful reuse of solution information, so the repairing heuristics embedded in the solveB&B (line 7) are evoked to improve the search (see section 8. 4.3).

Local Search Branching B&B is further improved by a heuristic which identifies the subproblem which has a tight upper bound at the early stage to help pruning more nodes in the tree (see initialization phase, section 8.4.3).

In the hybrid local search and B&B approach, the neighbourhood defined is usually a sub-MIQP, intensively searched by B&B. The default B&B in a general solver can usually be used as a black box solver to solve the subproblems. In our work, we also apply the default B&B to solve the subproblem as some other researchers have done [158] [49] . We place our focus on the identification of tight upper bounds and the reusing of solution information which contributes to the success of repairing heuristics.

Local search branching B&B

LB: lower bound;
UB: upper bound;
(h, x,w,z): a solution (x,w,z) of the problem with a corresponding objective value h;
solveB&B: the default B&B solver in CPLEX 10.0;
S and S': two *exclusive subsets* of Z, i.e. $S \cup S' = Z$: set of z_i^{hold};
P_{org}: orginal problem defined by model (PSP);
P_{subi}: subproblem defined by variable fixing;

1: **Initialization phase** // see section 3.2.3
2: while (stop condition is not met)
3: If ($LB\ (P_{subi}) \geq UB$)
4: prune the subproblem P_{subi}; // see section 3.2.3
5: go to line 13;
6: Else
7: (h, x,w,z) = solveB&B(P_{subi}) $\bigcup$ ($z_i^{hold} = 1$), $z_i^{hold} \in S$;
8: set $UB = \infty$;
9: perform **Local search phase** on the space of Z ; // see Fig. 4
10: update elements in set S;
11: generate subproblems by variable fixing: $P_{subi} = P_{org} \cup$ ($z_i^{hold} = 1$), $z_i^{hold} \in S$; // see section 3.2.2
12: go to line 2;
13: set (x^*, w^*, z^*) as the best solution among all (x, w, z) and h^* be the corresponding objective value;

Fig. 8.2 Local Search Branching B&B algorithm for minimization

8.4.2 Notations and definitions of variable fixing

(Hard) variable fixing or diving has been used in MIP context to obtain a reduced problem [48]. It assigns given values to a subset of variables of the original problem. A formal description of variable fixing is given in [48, 49]. In this work, variable fixing is applied to decompose the problem. Firstly, we define the original problem P_{org} on model PSP in section 8.2 .2 as follows:

$$
\begin{aligned}
&P_{org} : \min c^T x \\
&s.t. Ax \le b; \\
&x_j = \{0,1\}, \forall j \in B \ne \varnothing \\
&x_j = [0,1], \forall j \in C
\end{aligned}
$$

where x is the vector of variables which are partitioned into two subsets: B corresponds to binary variables and C corresponds to continuous variables.

We denote S and S' as two exclusive subsets of B, i.e. $S \bigcup S' = B$. The original problem P_{org} is thus decomposed into two subproblems P_{sub1} and P_{sub2} by fixing variables in subsets S and S' to 1 and 0, respectively:

$$
\begin{aligned}
&P_{sub1} : \min c^T x \\
&s.t. Ax \le b; \\
&x_j = 1, \forall j \in S \subseteq B \ne \varnothing \\
&x_j = [0,1], \forall j \in C
\end{aligned}
\qquad\qquad
\begin{aligned}
&P_{sub2} : \min c^T x \\
&s.t. Ax \le b; \\
&x_j = 0, \forall j \in S' \subseteq B \ne \varnothing \\
&x_j = [0,1], \forall j \in C
\end{aligned}
$$

For the problem modeled by PSP in section 8.2.2, we apply variable fixing on the core binary variables z_i^{hold}.

8.4.3 Solution information reusing and cut-off bound

The local search branching scheme creates a sequence of subproblems which have very similar structures. They only differ in the coefficient or the right-hand side of constraints which are related to z_i^{hold}. When solving this sequence of subproblems, the solution information such as the basis list and basis factors from its simplex tableau (i.e.

we apply the extended tableau simplex algorithm in the default MIQP solver) for the current problem are stored, and can be retrieved and applied to the successive subproblems. This means the solution information (i.e. basis list and basis factors) of the problem P_{subi} can thus be reused to obtain solution to $P_{subi'}$, so that $P_{subi'}$ do not need to be solved again from scratch. This solution information reusing thus can evoke the repairing heuristics embedded in the default IP solver (see section 8.5.2). This solution information reusing has been shown to be extremely efficient in the algorithm in the following experiment section.

Another important benefit we obtain with this approach is that, the subproblems generated by the local search are solved in sequence so the objective value h_i of problem P_{subi} can be used as a cut-off upper bound for the subsequent problems $P_{subi'}$. Therefore, it is important in our approach to delicately sequence the subproblems to obtain efficient upper bounds as soon as possible. The earlier a subproblem with a tight upper bound is obtained, the more subproblems can be pruned by applying the upper bound.

Therefore, the selection of the first appropriate set S to construct the first subproblem P_{sub} is a critical step in the algorithm. We aim to find a good incumbent as early as possible to reduce the number of nodes (subproblems) to be explored in Local Search Branching B&B. This means the first subproblem should, with a high probability, contain the subset of assets which are selected in the optimal solution.

In [106], it has been shown that, the set of assets selected by the continuous relaxation problem often contain the assets included in the integer optimal solution. Based on this, we propose a heuristic to select the first subset S to construct the first subproblem P_{sub} as the following. This heuristic is illustrated in Fig. 8.3:

Initialization phase

R: linear relaxation of the problem;
solveLP: a Linear Programming solver in CPLEX 10.0;
k : the number of assets allowed in the portfolio, as defined in the model (PSP);

1: solve the continues relaxation problem R(PSP): solveLP(R(PSP));
2: sort the assets according to a sort rule;
3: consider the sorted assets to generate subproblems:
4: select the first k assets and add them into set S;
5: set $S'=Z/S$;
6: generate subproblems by variable fixing:
7: $P_{subi}= P_{org} \bigcup (z_i^{hold} = 1), z_i^{hold} \in S$;
8: $P_{subi'}= P_{org} \bigcup (z_i^{hold} = 0), z_i^{hold} \in S'$;
9: obtain lower bound of subproblem: LB (P_{subi})= solveLP(P_{subi});
10: set $UB = \infty$;

Fig. 8.3 Initialization phase of Local Search Branching B&B approach

1. Relaxed problem solution: solve the continuous relaxed problem. Save the solution vector w;
2. Sort on the assets: sort the assets in a non-decreasing order of the reduced cost of the continues relaxation to model (PSP);
3. Select the assets: select the first k assets of the solution vector w. These assets form the first subproblem.

This heuristic is applied in line 1 in the algorithm presented in Fig. 8.2 to ensure a proper selection of the first subproblem.

8.4.4 Local search techniques

Our local search branching scheme performs on the binary variables z_i^{hold}. In this chapter, we apply a variation of Variable Neighourhood Search (VNS) [124] to carry out the search on z_i^{hold}. Theoretically, any local search technique can be applied to search on z_i^{hold}.

Local search phase
z: current assignment of z_i^{hold}

1: Select the set of neighbourhood structures N_l, $l = 1,\ldots,l_{max}$;
2: Provide an initial solution vector z (z represents the assignment of z_i^{hold})
3: *Repeat* the following steps for it_1 iterations:
4: set l=1;
5: *Repeat* the following steps for it_2 iterations:
6: *Exploration of the neighbourhood N_l of z with the aim to update the assignment of z_i^{hold}* : Find the first improved neighbour $z'$of z;
7: Move or not. If the new solution z' is better than z, set z=z' ; otherwise, set l= l+1;

Fig.8.4 Steps of VNS local search

Three neighbourhood structures N_l are employed in the algorithm. N_1 swaps one pair of elements in S and S'. N_2 and N_3 swaps two and three pairs of elements, respectively. For each current neighbourhood structure N_l, a given number of it_2 iterations are carried out before the search moves to the next neighbourhood structure N_{l+1}. This procedure terminates after it_1 iterations. Therefore, Local Search Branching B&B is an incomplete search. It aims to seek near optimal solutions in a limited computational time.

8.5. Experimental results

8.5.1 Test problems

The extended model PSP based on the MV model in section 8.2.2 is concerned with properties of a real-world portfolio selection problem derived from Société Générale Corporate & Investment Bank. The problem takes a set of side constraints as well as transaction costs into consideration.

To test our algorithm on more general benchmark instances, we also test in this paper the portfolio optimisation instances publicly available in the OR library [148], with additional constraints derived from the above real problem. Table 8.1 presents the properties of these 6 instances tested in this work.

Table 8.1 Properties of problem instances. n represents the total number of assets available; k represents the number of assets to be held; w^0 represents the current positions. We assume that the starting portfolio has capital equally invested in all assets; therefore w^0 is set to 1/n.

Instance	n	k	w^0
Société Générale	23	10	1/23
Hang Seng	31	20	1/31
DAX	85	40	1/85
FTSE	89	50	1/89
S&P	98	60	1/98
Nikkei	225	150	1/225

We set the minimum proportion of wealth to be invested in an asset, $w_{\min}$, to 0.01% and the minimum transaction amount, $x_{\min}$, to 0.01%. We also set the parameters in the transaction cost function α_i to 0.005 and β_i to 0.0001 (see section 8.2.2). Other values of k in the cardinality constraint have been tested, ranging from 10 to 150 for different size of portfolios.

8.5.2 Evaluations on the local search branching B&B algorithm

In our experiments, we analyze different aspects of the proposed approach, including subproblem solving and overall problem solving, etc. The B&B algorithm with local search branching scheme is implemented in C++ with concert technology in CPLEX on top of CPLEX10.0 solver. All experiments have been carried out on an Intel Core 1.86GHz machine with 1.97GB memory.

8.5.2.1 The size of the original problem and subproblems

Firstly, we compare the size of the original problem and subproblems after the local search branching. The purpose of this comparison is to assess the effectiveness of problem decomposition by using the local search branching. Both of the original and subproblems are MIQP problems. In Table 8.2, it can be seen that after fixing the values for variable z_i^{hold} by the local search branching, the resulting subproblems are much smaller than the original ones (reduces up to 70% of the number of rows, columns and nonzeros). It has been observed that the sizes of the subproblems are similar (Table 8.2 presents the average size of subproblems). Due to the smaller size, the MIQP

subproblems are all solvable within in seconds by the default B&B in CPLEX 10.0 in our experiments. Note that in this work we aim to reduce the computational time of solving the problem by heuristically decompose the original problem. The subproblems produced still present to be NP-hard. However, as the problem size is significantly reduced by using variable fixing on z_i^{hold}, the subproblems can be easily solved efficiently.

Table 8.2 Size of the original MIQP problem and MIQP subproblems

Instance	Original problem			Subproblem		
	No. of rows	No. of columns	No. of nonzeros	No. of rows	No. of columns	No. of nonzero
Société Générale	164	115	401	52	40	124
Hang Seng	220	155	557	102	80	259
DAX	598	425	1529	202	160	519
FTSE	626	445	1600	525	200	649
S&P	689	490	1763	302	240	779
Nikkei	1578	1125	4048	752	600	1948

8.5.2.2 Details of the subproblem solving

In this section, we analyze the deferent behaviors (i.e. CPU time spend) of subproblem solving. The deferent behaviors of subproblem solving can demonstrate the effectiveness of information reusability we claimed in section 8.4.3.

In Local Search Branching B&B, each neighbourhood of the current solution is evaluated by solving the corresponding subproblem by B&B. That is, after fixing values for variables z_i^{hold} by the local search branching scheme, the resulting MIQP subproblem is created. It is solved in subsequently by the default B&B in CPLEX10.0. When these subproblems are processed, four possible situations could emerge: (1) a subproblem could be solved by B&B to optimality; (2) (2) the repairing heuristic mechanism [74] imbedded in CPLEX could be evoked and applied to a subproblem to obtain a feasible solution heuristically; (3) a subproblem could be pruned; this will happen if the optimal solution under LP relaxation is larger than the current upper bound; and (4) the solution of a subproblem could be infeasible.

Table 8.3 illustrates the behavior of the above four situations during the processing of subproblems. The total CPU time of the algorithm is dependent upon the CPU time needed for each situation.

Table 8.3 Information of subproblem processing

Instance	Total CPU time	subproblem solved		subproblem repaired		subproblem pruned		subproblem infeasible	
		Number	Avg CPU time per subp	Number	Avg CPU time per subp	Number	Avg CPU time per subp	Number	Avg CPU time per subp
Société Générale	3.16	56	0.01	398	0.006	86	0	60	0
Hang Seng	3.09	184	0.01	178	0.005	120	0	118	0
DAX	9.00	296	0.02	121	0.01	112	0.01	71	0
FTSE	11.44	79	0.08	102	0.025	127	0.02	292	0
S&P	13.55	286	0.04	114	0.01	77	0	123	0
Nikkei	76.97	89	0.40	21	0.36	221	0.08	269	0.06

Table 8.3 clearly indicates that the CPU time for identifying infeasibility is negligible. The CPU time for pruning the inferior subproblem (by calculating its optimal solution of the LP relaxation) is quite efficient. Therefore, the more nodes pruned, the more efficient the search is. It can be interpreted from Table 8.3 that solving subproblems with repairing heuristics is quite efficient. These repairing heuristics are the results of solution information reuse in B&B solver. Solving subproblem exactly is the most time consuming situation comparing with other three situations.

The solution information shown in Fig. 8.5 can further demonstrate that the solution information reusing makes the search procedure more efficient. Fig. 8.5 is a partial CPLEX log file. It records the objective value of relaxation, objective value of integer solution, and gap, etc on each node of the tree of the subproblems solved by B&B. In Fig. 8.5, an asterisk (*) on the left-most column for any node indicates that an integer feasible solution has been found. It also logs the successful application of repairing heuristics on the node, and denotes by + the node where an integer feasible solution has been generated by the heuristics. It can be seen that the integer feasible solution with a small gap usually can be obtained at the root node of the tree with negligible CPU time.

```
Tried aggregator 1 time.
MIQP Presolve eliminated 147 rows and 66 columns.
…
…
Reduced MIQP has 52 rows, 40 columns, and 124 nonzeros.
Presolve time =    0.00 sec.
Root relaxation solution time =    0.00 sec.

   Node      Objective      Best Integer      Gap
      0         0.0627           0.0627
*     0+          0.0651           0.0627        3.67%

…
…
Reduced MIQP has 52 rows, 40 columns, and 125 nonzeros.
Presolve time =    0.00 sec.
MIP emphasis: balance optimality and feasibility.
Root relaxation solution time =    0.00 sec.

   Node      Objective    Best Integer     Gap

      0          0.0558
*     0+                      0.0568       1.67%
*     0+         0.0558       0.0558       0.05%
…
…
```

Fig. 8.5 Part of a log file in CPLEX for solving a MIQP subproblem for Société Générale

8.5.2.3 The cut off bound in the local search branching B&B

The initialization heuristic proposed in section 8.4.3 is applied to construct the first subproblem in Local Search Branching B&B. In this section, we assess the effectiveness of the initialization heuristics.

Table 8.3 and Fig. 8.5 have shown that the CPU time of the search by pruning the inferior subproblems (by calculating its optimal solution of LP relaxation) is less than that of heuristics repairing, and much less than solving the subproblems exactly. This demands an efficient heuristic to detect the first subproblem and provide a good bound, as the more subproblems pruned, the more efficient the algorithm is.

Fig. 8.6 presents the comparison between the heuristic initialization and a random initialization. It plots the decreasing objective function values over the iterations of Local Search Branching B&B. The total number of iterations corresponds to the total number of subproblems being solved, either exactly or heuristically. That is, the total number of iterations is the sum of subproblems solved and subproblems repaired.

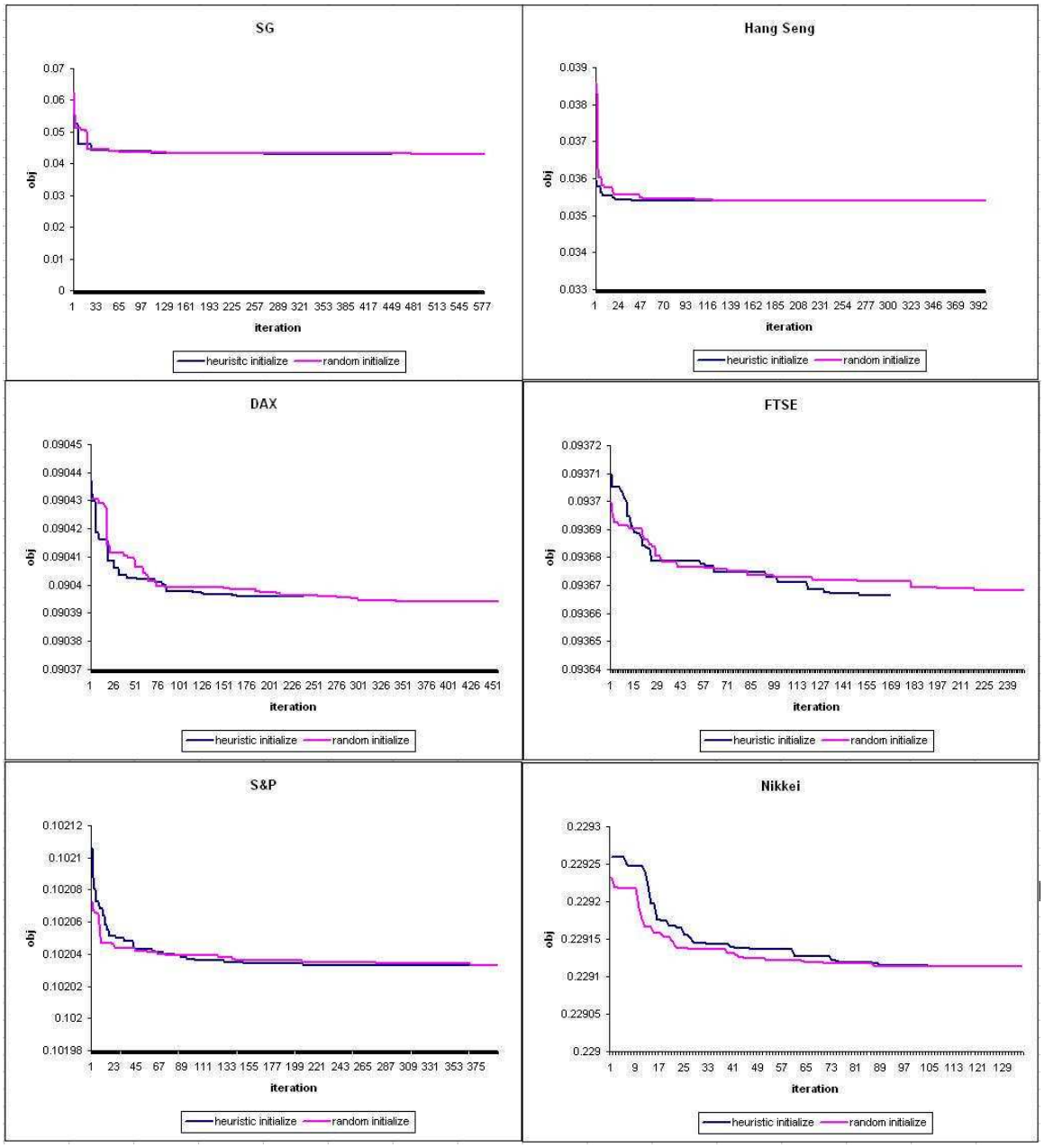


Fig. 8.6 The local search branching B&B with heuristic initialization and random initialization

Fig. 8.6 demonstrates that for most of the instances, Local Search Branching B&B with heuristic initialization converges faster than that of with random initialization. For instances DAX, FTSE and S&P, although the initial objective value of heuristic initialization is larger than that of random initialization, the decreasing rate of objective values of heuristic initialization is higher than that of random initialization. The only exception is the Nikkei instance. This may be due to that the cardinality constraint in Nikkei requires the selection of a relatively large portion, 150 assets out of 225 assets. Therefore, the random initialization has a higher chance of including more appropriate assets in the optimal portfolio.

What is more important drown from Fig. 8.6 is that, Local Search Branching B&B with heuristic initialization processes less number of subproblems than the one with random initialization. This can be demonstrated by that the number of iterations of Local Search Branching B&B with heuristic initialization is less than the one with random initialization. The heuristic initialization speeds up Local Search Branching B&B.

8.5.2.4 Comparisons with the default B&B in CPLEX

Portfolio selection problem is one of the most studied topics in finance. A wide range of models have been proposed to tackle the problems. Several variable definition, objective functions, constraints, and data sets have been proposed. For this reason, fair and exhaustive comparison of all the published papers cannot be performed. What is more, to our best knowledge, our model with non-convex transaction cost formulation is first time presented in the literature. In order to evaluate the quality of the solutions we obtained from Local Search Branching B&B, we compare it against the optimal solution to the problem.

It is worth noting that Local Search Branching B&B is a heuristic approach to the problem. It cannot prove optimality of the solution due to the nature of the local search on core variables z_i^{hold}, although the subproblems can be measured by the optimality gap. In order to evaluate the quality of the solutions we obtained from Local Search Branching B&B, we compare it against the optimal solution to the problem. It is

however very difficult, if not impossible, to obtain and prove the optimal solution to the problems. We therefore calculate the *approximate* optimal solution to the problem by running the default B&B algorithm in CPLEX10.0 for an extensive amount of time.

We compare Local Search Branching B&B with the default B&B in Table 8.4 in terms of the following criteria:

- The number of nodes being processed in B&B to obtain the best integer feasible solution;
- The gap between optimality and the quality of the best feasible solution;
- If the repairing heuristic is evoked and succeed;
- The total CPU time required.

From Table 8.4 we can see that by decomposing the problem through variable fixing on z_i^{hold}, the repairing heuristics succeed in Local Search Branching B&B approach. The repairing heuristics cannot be evoked by the default B&B while solving the original problem.

Without the decomposition, the default B&B needs to explore a much larger number of nodes in the tree to obtain feasible solutions, while Local Search Branching B&B with decomposition requires much less time, shown in Table 8.4. For example, for the largest instance Nikkei, more than 35500 nodes have been explored in the default B&B to obtain a feasible solution with a gap of 0.44%.

The optimality gap of solution obtained by Local Search Branching B&B is calculated by gap = $(f_{LS}-f_{LP})/ f_{LS}$, where f_{LS} is the objective value obtained by Local Search Branching B&B, and f_{LP} is the objective value of LP relaxation. Table 8.4 shows that, to achieve solutions of similar quality (as measured by the optimality gap), the CPU time needed by the default B&B is much greater than that required by Local Search Branching B&B (e.g. 600 CPU seconds as opposed to 76.97 seconds for Nikkie).

The comparison of Local Search Branching B&B with default B&B can be more clearly illustrated in Fig. 8.7, which plots of the objective values of Local Search Branching B&B and the approximate optimal values obtained by the default B&B with extensive runtime (600 seconds).

Table 8.4 Comparisons of default B&B and local search branching B&B. + denotes that the repairing heuristics are succeed. All the CPU time is measured in second.

		Société Générale	Hang Seng	DAX	FTSE	S&P	Nikkei
Default B&B (original problem)	No. of nodes processed	30	50	150200	147100	130800	35500
	Optimality Gap	0.22%	1.06%	4.66%	3.65%	2.74%	0.44%
	Repair success	No	No	No	No	No	No
	CPU time			180			
	No. of nodes processed	60	80	541800	486700	365800	105000
	Optimality Gap	0.1%	0.29%	4.66%	3.63%	2.74%	0.43%
	Repair success	No	No	No	No	No	No
	CPU time			600			
LS branching B&B (subproblem)	No. of nodes processed	0+	0+	50+	30+	30+	50+
	Repair success	Yes	Yes	Yes	Yes	Yes	Yes
	Optimality gap*	0.22%	1.07%	4.65%	3.67%	2.75%	0.44%
	CPU time total*	3.16	3.09	9.00	11.44	13.55	76.97

It can be seen that Local Search Branching B&B converges very well for instances Société Générale, Hang Seng and Nikkei, where the gap between the objective values of Local Search Branching B&B and approximate optimal is very small. For instance DAX, the best solution of Local Search Branching B&B is even better than the approximate optimal value. For instances FTSE and S&P, the gap is slightly larger. However, it should be noted that Local Search Branching B&B spends significantly less time (3-79 seconds) than the default B&B (180 and 600 seconds).

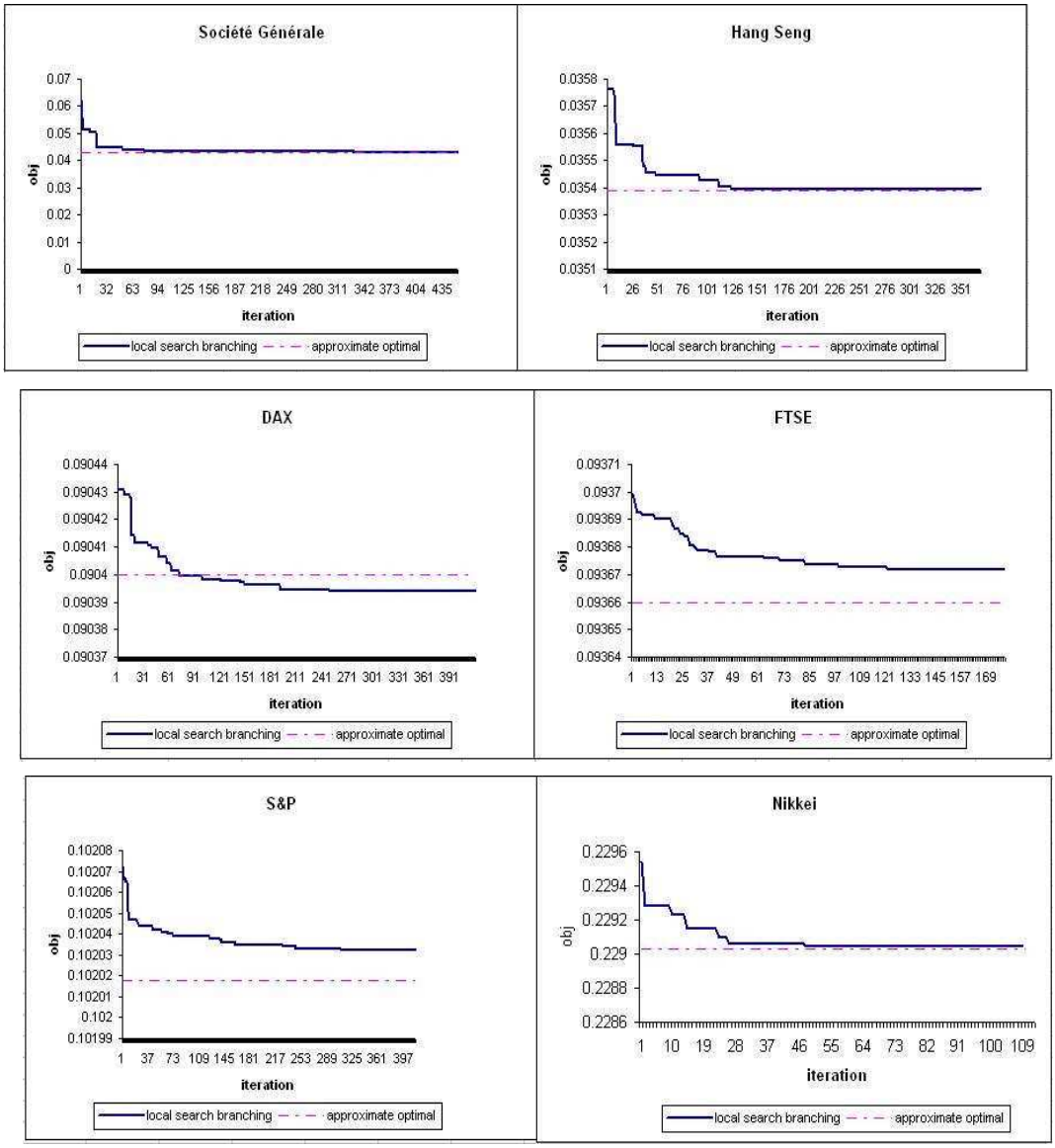


Fig. 8.7 The gap between the local search branching B&B and approximate optimal by default B&B

8.6 Conclusions

In this chapter, we investigate a hybrid method named ***Local Search Branching B&B***, which can be seen as a decomposition method for the PSP. This new hybrid approach effectively integrates local search into the B&B algorithm to implement an incomplete search which aims to seek near optimal solution heuristically in a limit computational time. A set of core variables of the problem are first selected according to the property

of the problem. *Variable fixing* is applied to these core variables to define the subproblems. A new *local search branching* strategy is proposed and performed on these core variables to decompose the problem into a sequence of subproblems. The default B&B search then solves these restricted subproblems optimally due to their reduced size comparing to the original one. Due to the inherent similar structures of the subproblems, the reusability of solution information evokes the repairing heuristics in the default B&B. This thus accelerates the B&B solving procedure of the subproblems. The tight upper bound identified at early stage of the search can prune more nodes (subproblems) in the tree. This speeded up Local Search Branching B&B search to the optimal solution to the original problem.

In this chapter, we apply variable fixing to define the subproblems. As we introduced in chapter 2 section 2.8.2, variable fixing assigns values to *a selected restricted subset of variables* of the original problem. Therefore, we can reduce the analysis of the whole solution space to a promising region. We apply variable fixing as a decomposition approach to the problem in this chapter. Benders' decomposition is an approach that solves certain optimisation problem efficiently by inferring information from its dual problem. The first step of Benders' decomposition also consists of *fixing certain amount of variables* in the original problem, hereby making the resulting subproblem easy to solve. The essence of Benders' decomposition lies in determining which variables to be fixed so that results in easy to solve subproblem and the strongest bound can be derived from its dual problem. The information derived from dual problem is added as Benders' cut to the master problem. In this chapter, we do not utilize the information from the dual problem. Choosing whether or not to impose this solving method depends heavily on the knowledge of the potential simplicity of certain"easy" subproblems. The Benders' decomposition and Branch-and-Cut algorithm will be investigated in our future work.

There is some similarity between our proposed local search branching scheme with the existing schemes (e.g. local branching, RINS, VNSB and VNDS). The similar basic mechanism can be concluded as: (1) the decomposition of the original problem due to

the variables fixing; and (2) the local search performed on specifically defined neighbourhoods. In our approach, we decompose the problem by fixing core variables z_i^{hold}, and the local search is performed on z_i^{hold} to generate sequence of subproblems.

The main contributions of our hybrid approach are: (1) In all the existing approaches (local branching, RINS, VNSB and VNDS) in the literature, the definition of neighbourhoods is based on the incumbent solution of the problem. That is, an integral feasible solution is required, and these approaches can be seen as improvement heuristic search methods. Our approach is executed without the request of feasible solution. It works as a constructive search method (i.e. the subproblems are solved by B&B) as well as an improvement search method (i.e. using local search to obtain the best solution). (2) Our hybrid approach emphasizes the solution information reusing, demonstrated by the succeed of repairing heuristics in the experiments to speed up the subproblems solving; and (3) Our hybrid approach extends the basic mechanism by heuristically identifying the subproblem which can provide a tight upper bound to prune more subproblems thus significantly improve the efficiency of the algorithm on solving the complex portfolio optimisation problem with real life trading constraints.

Chapter 9 Conclusions and future work

9.1 Conclusions

9.1.1 Research overviews

As we stated in section 1.3, the scope and the aims of this thesis are to investigate how to efficiently integrate Constraint Programming, Operational Research techniques and heuristic search methods to solve two combinatorial optimisation problems from real-world applications, taking the advantages of each well developed component. We identify problem/subproblem features and correlated suitable algorithms firstly. Then we exploit several hybrid algorithms for each problem.

We demonstrate these integration methods on two real-world application problems. Firstly, we apply three hybrid algorithms on nurse rostering problems, i.e. hybrid CP with VNS approach in chapter 4, constraint-directed Large Neighbourhood Search in chapter 5 and CP based column generation in chapter 6.

These three hybrid algorithms integrate different techniques with respect to the specific features of the problem. Each hybrid algorithm emphasizes specific features of the problem and correlated suitable algorithms. The hybrid CP algorithm in chapter 4 emphasizes the feasibility reasoning of CP. Chapter 5 emphasizes the usage of information about constraints during the local search. The neighbourhood structure can be generally defined by the constraints. The local search can be guided by the evaluation functions that are the violation of the constraints. In chapter 6, the feasibility reasoning is still handled by CP while the relaxation and optimality reasoning is handled by IP/LP in the form of CP based column generation. Besides the complementation benefits gained from CP and column generation, another benefit we gain is that we derive a lower bound for the problem which cannot be obtained by pure local search or

meta-heuristic methods. With this lower bound we can have some knowledge of how far away the current solution we obtained is from the optimal one.

We investigate two hybrid algorithms on the second application problem - portfolio selection. The basic formulation of the problem is usually solved by Branch-and-Bound algorithm. The additional feature of the problem, i.e. discrete feature due to the presence of the side constraints requires the hybridization of heuristics and local search with Branch-and-Bound. Experiments show the effectiveness of heuristics, i.e. branching rule and node selection heuristic in the Branch-and-Bound algorithm in chapter 7. In chapter 8, local search works as branching rule for Branch-and-Bound so that these two techniques are integrated and interplayed more closely.

9.1.2 Research contributions

In chapter 4, a new decomposition approach to nurse rostering problems based on good quality solution blocks is proposed to solve the problems successfully. This chapter proposes a new solution route as following for nurse rostering problems. This solution approach has the potential to be applied to other complex and large optimisation problems with similar features to those of the nurse rostering problem:

1. Decompose the problem into subproblems according to the feature of the problem. The subproblems may be easy to handle by certain techniques efficiently, e.g. CP. In this chapter, the problem is decomposed by constraints. A CSP model is built to generate feasible solutions to the subproblems.
2. Obtain the feasible solutions to the subproblems by CP, and then merge the solution to the subproblems to get feasible solution to the complete problem. In this chapter, we apply iterative forward search to merge the solution to the subproblem into the solution to the complete problem.
3. Apply local search methods on the feasible solutions to get improved solution.

In chapter 5, a constraint-directed local search is proposed and successfully solves the nurse rostering problems. The contributions can be concluded as following:

1. The neighbourhood in local search is defined in a more general way by constraints. The local search approach asks for large-scale neighbourhood, e.g. changes of chain of variables. The neighbourhoods we design in this chapter cover all these types of neighbourhoods.
2. The search of neighbourhood is done by CP. Two benefits we can gain are: firstly, we can take the advantage of CP's search. Secondly, the search is easy to implement.
3. We apply the violation measure of constraints as evaluation functions during the search which makes the search more informative.

In chapter 6, a CP based column generation to nurse rostering problems is proposed and demonstrated. The contribution is listed as following:

1. Another decomposition method, column generation integrated with CP is applied to nurse rostering problems. In this hybrid decomposition approach, we use CP to solve the pricing subproblem and column generation to handle the master Integer Program problem.
2. Two strategies which aim to generate good and diverse columns we proposed have been demonstrated by the experimental results. These efficient search strategies speed up the Linear Program relaxation convergence and satisfy the integrality request of the master problem.
3. This approach provides a lower bound for the problems. Hence we can know how far the current solution falls short of the optimal solution.

In chapter 7, we develop a decomposition approach, layered Branch-and-Bound (layered B&B) algorithm, for solving the problem we investigate. In the B&B search tree, sets of variables are layered (decomposed) according to their different features, and search is performed on one layer before another in sequence. The layered B&B algorithm can be seen as firstly searching on the top layer of the tree (subproblem of a set of variables) then *diving* into a particular region of the search space in order to explore it intensively. Several benefits are achieved in chapter 7:

1. Search is performed intensively on those variables with a higher priority (at the higher layer). Intuitively, this means we focus on the core variables of the problem first, and then deal with the rest of the variables.
2. A heuristic which works well for one subset of variables of the problem may not be appropriate for the other variables. By layering the tree (decomposing the variables of the problem), we can easily devise different efficient heuristics to different layers.
3. Search is more easily manipulated within the given time limit by aborting it at each layer accordingly. Of course, the optimality of solution will be sacrificed, but the quality of the solution can still be measured by the gap between the incumbent solutions and the optimal solution.

In chapter 8, we propose a new hybrid approach which integrates local search into the B&B algorithm. In this integrated B&B, we propose a new branching scheme which applies the idea of local search. Instead of branching on a single variable, the local search branching scheme branches on a set of core binary variables of the problem iteratively to generate a sequence of subproblems. These subproblems are then solved in sequence by the default B&B in a general solver and the best solution among them is the approximate optimal solution of the original problem. The contribution can be concluded as following:

1. The main contribution is the tight integration of local search with B&B. The idea is to perform certain efficient and computational cheap search by local search branching on the *surface* of the problem which consists of core variables, and then dive into a particular region of the search space and explore it more intensively.
2. The proposed decomposition approach generates subproblems with similar structures. The inherent similar structures of the subproblems facilitate efficient and successful solution information reusing in solving the subproblems.
3. The local search branching B&B search is further improved by a heuristic that identifies the subproblem which has a tight upper bound to help prune more nodes (subproblems) in the tree.

9.2 Future work

In this section, we discuss the future research directions in two perspectives: firstly, from the specific application problems perspective; and secondly, from the solution techniques perspective.

9.2.1 Future research directions for nurse rostering problems

Continuity

An important issue in nurse rostering is the continuity from one rostering period to the next. The nurse rostering benchmark instances we tested in this thesis are designed only to produce rosters for an isolated period, applying penalties in accordance with the convention that all potential violations are counted at the beginning of the period, and ignored at the end. We recognise that the benchmark instances are intended as a basis for comparison between alternative rostering methodologies, and that the consideration of an isolated rostering period serves this purpose. However, in a practical environment, information relating to one rostering period is carried forward to the next, creating additional issues of ‘‘continuity”.

For example, although the rostering period is one month in length, the constraints do not primarily relate to a one month period. In those constraints which relate to periods of time, some relate to one week, others relate to a rolling 5-week period, or even a rolling 13-week period. Effective approaches need to be designed to handle the constraints relating to various time periods.

Rerostering

In this thesis, we construct a deterministic personnel roster that determines the line-of-work for each nurse member. However, administration systems in hospital typically have to operate in a dynamic and uncertain environment where unexpected events may occur. The rerostering problem is a scheduling type of problem that most hospitals

confront. When the unexpected events lead to schedule disruptions and infeasibilities, rerostering is necessary to update the activity schedule. Therefore, rerostering is a very important and interesting topic in the personnel scheduling environment. Decision support systems that adequately react to unexpected events should be developed.

Multi-criteria problems

For many optimisation problems it is unclear what exactly should be optimised. For the nurse rostering problems we tested in this thesis, the objectives and constraints are extracted from real-world cases and preserved with the essential characteristics. However, it is common, especially in the industrial context, that the problem may have conflicting goals. The constraints are typically preferences rather than necessary requirements. Therefore, Pareto-optimal solutions against different criteria are expected. What is more, if the goal is to perform collaborative work on the problem, a more practical approach is needed. For example, in exploring the "what-if" scenarios, different set of solutions should be provided with different situations.

9.2.2 Future research for portfolio selection problems

Multi-period problem

In this thesis, we tackle the portfolio selection problem in a single period. Of many other possible extensions, most worthy of mentioning are those with a multi-period setting or continuous time. These are significantly more complex problems due to the stochastic dynamics. The desirability of a trade in a given stock must then take into account the alternative of delaying the trade. The challenge is to develop effective numerical methods for the (approximate) solution of the resulting stochastic programming (or optimal stopping) problems.

Risk measures

Applying which term to measure the risk associated with the portfolio, to a certain extent, determines the complexity of the model. Besides applying covariance as the risk measure of the portfolio, several other risk measures have been investigated in the literature and in practice, such as mean absolute deviation, and mean absolute semi-deviation etc. More recently, some researchers focus on other risk measures where quantise and tail of the distribution of the return, such as value at risk and conditional worst expectation, are used. Different risk measures which capture the features of the market can be designed and applied in the portfolio selection model to reflect the requirements of the investors more accurately.

9.2.3 Future research for hybrid algorithms

Integrate CP with other OR techniques

MIP offers several ideas that can benefit search in general. The majority among them is the use of a relaxation, usually a continuous LP relaxation to guide the search. The optimal solution of the LP relaxation is applied as the lower bound during the search to prune unpromising parts of the search tree. Stronger relaxation makes it worth to invest more processing time at each node of the search tree. Therefore, different relaxations are worthy of investigation in future work.

The second lesson that can be learned from MIP is the use of duality. The LP dual, Largrangean and many other duals can be used to construct a nogood or Benders cut that directs the search away from poor solutions. CP based Bender decomposition allows us to apply CP and OR techniques to different parts of the problem. The CP search can learn from past experience by accumulating Benders cuts (in a form of nogood).

Integrate CP or IP with heuristics

Applying meta-heuristics within exact methods can help to gain robustness and constrained-CPU-time effectiveness. The research topic such as (conflict/heuristic) information learning in the design of branching rule, node selection rule etc. in Branch-and-Bound algorithm will be investigated in our future work.

In chapters 4 and 5, CP search is integrated with heuristic local search. In our future work, more information can be inferred from the CP search. For example, by applying specific designed heuristics, a "good" value can be associated with a variable. If all the variables have heuristic values, then the further heuristic information can be derived, such as conflicts between these values. This will enable more efficient propagation on group of constraints. More efficient propagation algorithms which work on group of constraints will be investigated in our future work.

In chapter 6, the generation of column is purely done by CP search. However, heuristic construction method can be applied to gain more efficiency in the procedure of pricing subproblem solving.

Based on the work that has been done in chapters 7 and 8, heuristic information can be applied to tree search of IP to improve the efficiency of the search. For example, a linear solver can be used to find specific values for the variables at which the linear relaxation of the problem has an optimal solution.

Apply information of OR to meta-heuristics

The existing literature has demonstrated the possibility of using effective algorithmic schemes, such as meta-heuristics, for solving hard optimisation problems. However, current meta-heuristics make very limited use of explicit mathematical tools. We will investigate the possibility of embedding sound mathematical techniques into robust meta-heuristic approaches to optimisation. For example, some information obtained

from linear solver can be actually applied in the designing of meta-heuristics. For example, reduced cost measures the influence on the optimal cost of changing the value assigned to certain variables. This can be used in domain pruning and search heuristics. This enables the information that the mathematical programming solvers extract from the cost function to be exploited by other solvers or the search.

The investigation of the possibility of embedding sound mathematical techniques into robust meta-heuristic approaches to optimization is also essential idea of "matheurisitcs". The hybrid methods investigated in this thesis have close relationship with "matheuristics", since both of them seek efficient integration of mathematic methods with meta-heuristics. We will continue the research on this promising topic.

Appendix

Hard and Soft Constraints in the Nurse Rostering Benchmarks

Hard constraints	Category	Details
Gpost	One shift per day	One shift per day (D, N, O)*
	Coverage (no over/under cover)	Weekday: 3D 1N; Weekend: 3D 1N
	Working time	Full time: 18 shifts; Part time: 10 shifts
	Shift patterns	Maximum consecutive working days: 6
		Maximum consecutive N shifts: 3
		Maximum consecutive working weekends: 3
		After a series of work, at least 2 days off
		Complete weekends, i.e. free or work on both days
		After N shifts, at least 2 days off
Valouxis	One shift on one day	One shift one day (D, E, N, O)*
	Coverage (no over/under cover)	Weekday: 4D 4E 2N; Weekend: 3D 3E 2N
	Working time	18 shifts
	Shift patterns	Maximum consecutive working days: 5
		Maximum consecutive N shifts: 3
		Maximum consecutive working weekends: 3
		After a series of work, at least 2 days off
		Complete weekends, i.e. free or work on both days
		After N shifts, at least 2 days off
ORTEC	One shift one day	One shift on one day (E, L, D, N, O)*
	Coverage (no over/under cover)	Weekday: 3E 3D 3L 1N; Weekend: 2E 2D 2L 1N
	Working time	Group 1: 36 hours/week; Group 2: 32 hours/week; Group 3: 20 hours/week
	Shift pattern	Maximum consecutive working days: 6
		Maximum consecutive N shifts: 3
		Maximum consecutive working weekends: 3
		After a series of work, at least 2 days off
		Complete weekends, i.e. free or work on both days
		After N shift, at least 2 days off

*D: day shift; E: evening shift; L: late shift; N: night shift; O: day off.

Appendix

Soft constraints	Category	Details	Weights	Violation measure
Gpost	Balanced workload	Full time: [4,5] shift/week Part time: [2,3] shift/week	1	*Difference between the no. of shifts received and the acceptable no. of shifts per week
		Full time: series of shifts length [4,6] Part time: series of shifts length [2,3]	1	*Difference between the no. of shifts received and the acceptable series length
	Pattern preference	No stand alone shift, i.e. single day on	100	Number of isolated shifts
		No one shift over a weekend	100	Number of incomplete weekends
		No one day off between shift series	10	Number of one day off
Valouxis	Balanced workload	No. of D shifts: [5, 8] in the schedule	100	Difference between the no. of shifts received and the acceptable no. of shifts
		No. of E shifts: [5, 8] in the schedule	100	Difference between the no. of shifts received and the acceptable no. of shifts
		No. of N shifts: [2, 5] in the schedule	100	Difference between the no. of shifts received and the acceptable no. of shifts
	Pattern preference	No stand alone shift, i.e. single day on	1000	Number of isolated shifts
		No one shift over a weekend	1000	Number of incomplete weekends
		A D after E should be avoided	1000	Number of D shifts after E shift
		A E after N should be avoided	1000	Number of E shifts after N shift
		A D after N should be avoided	1000	Number of D shifts after N shift
		At least 2 days off between shift series	100	Number of one day off
		Series of D/E/N shift length:3	40	Difference between the series length and the acceptable length
		Series of D/E/N shift length: 3	20	Difference between the series length and the acceptable length
ORTEC	Balanced workload	Group 1: [4,5] shifts/week Group 2: [4,5] shifts/week Group 3: [2,3] shifts/week	10	*Difference between the no. of shifts received and the acceptable no. of shifts per week

		Group1: length of shift series [4,6] Group2: length of shift series [4,6] Group3: length of shift series [2,3]	10	*Difference between the no. of shifts received and the acceptable series length
	Pattern preference	No stand alone shift, i.e. single day on	1000	Number of isolated shifts
		No one shift at a weekend	1000	Number of incomplete weekends
		Length of a series of N shifts: [2,3]	1000	Difference between the series length and the acceptable length
		At least 2 days off between shift series	100	Number of one day off
		Length of a series of E shifts: [2,3]	10	Difference between the series length and the acceptable length
		Length of a series of L shifts: [2,3]	10	Difference between the series length and the acceptable length
		A E after D should be avoided	5	Number of E shifts after D shift
		A N after E should be avoided	1	Number of N shifts after E shift

* In order to have same evaluation functions for the solutions with other approaches in the literature, the constraints denoted by * is measured by quadratic function. That is, the violation measure squared and multiplied by the corresponding weight

List of Publications

The research presented in this thesis has been published (or is currently under review) as follows:

1. **F. He,** R. Qu, *A Constraint Programming based Column Generation Approach to Nurse Rostering Problems*, under review at Computers & Operations Research, 2011
2. **F. He,** R. Qu, *A Layered Branch-and-Bound Algorithm to Portfolio Selection Problems with Real-world Constraints*, under review at Journal of Heuristics, 2011
3. **F. He**, R. Qu, *A hybrid local search and Branch-and-Bound approach to constrained portfolio selection problem*, under review at INFORMS Journal on Computing, 2011
4. F. He, R. Qu, *A Constraint-directed Local Search Approach to Nurse Rostering Problems*, Proceeding of the 6th International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS'09) at the 15th International Conference on Principles and Practice of Constraint Programming (CP'09), Lisbon, Portugal, 20th-24th September, 2009
5. R. Qu, **F. He**, E. K. Burke, *Hybridizing Integer Programming Models with an Adaptive Decomposition Approach for Exam Timetabling Problems*, Proceeding of 4th Multidisciplinary International Scheduling Conference (MISTA2009), Dublin, Ireland, 10th-12th August, 2009
6. R. Qu, **F. He***, A Hybrid Constraint Programming Approach for Nurse Rostering Problems*, Allen T., Ellis R. and Petridis M. (eds.) Applications and Innovations in Intelligent Systems XVI. The Twenty-eighth SGAI International Conference on Artificial Intelligence (AI-2008), 211-224, Cambridge, England, 9th-11th December, 2008

References

1. Burke, E.K., et al., *The state of the art of nurse rostering.* Journal of Scheduling, 2004. **7**(6): p. 441-499.
2. Osogami, T. and H. Imai, *Classification of Various Neighbourhood Operations for the Nurse Scheduling Problem* Lecture Notes in Computer Science, 2000. **1969**: p. 72- 83.
3. Apt, K.R., *Principles of Constraint Programming*. 2003: Cambridge University Press.
4. Rossi, F., P.v. Beek, and T. Walsh, *Handbook of Constraint Programming*. Foundations of Artificial Intelligence, ed. J.Hendler, H.Kitano, and B.Nebel. 2006: Elsevier.
5. Hooker, J.N., *Integrated Methods for Optimization*. 2007: Springer.
6. Glover, F. and G.A. Kochenberger, *Handbook of Meta-Heuristics*. 2003: Kluwer.
7. Maringer, D.G., *Portfolio Management with Heuristic Optimization*. 2005: Springer.
8. Wolsey, L.A. and G.L. Nemhauser, *Integer and Combinatorial Optimization*. 1999: Wiley.
9. Machworth, A.K., *Consistency in networks of relations.* Artificial Intelligence, 1977. **8**: p. 99-118.
10. Machworth, A.K. *On reading sketch maps*. in *Proceedings IJCAI'77*. 1977: Cambridge MA.
11. R.Mohr and T.C.Henderson, *Arc and path consistency revisited.* Artificial Intelligence, 1979. **28**: p. 225-233.
12. R.Mohr and G.Masini. *Good old discrete relaxation*. in *Proceedings ECAI'88*. 1988: Munchen, FRG.
13. van Hoeve, W.J. and I. Katriel, *Global Constraints*, in *Handbook of Constraint Programming*, F. Rossi, P.v. Beek, and T.Walsh, Editors. 2006, Elsevier B.V. p. 169-208.
14. Harvey, W.D. and M.L. Ginsberg, *Limited Discrepancy Search*, in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. 1995. p. 607- 613.
15. Bartak, R. *Constraint Propagation and Backtracking-based Search.* **Volume**,
16. Regin, J.C., *Generalized arc consistency for global cardinality constraint*, in *National Conference on Artificial Intelligence*. 1996, AAAI Press. p. 209-215.
17. Quimper, C.G., et al. *Improved algorithms for the global cardinality constraint*. in *Principles and Practice of Constraint Programming*. 2004.
18. Katriel, I. and S. Thiel. *Fast bound consistency for the global cardinality constraint*. in *Principles and Practice of Constraint Programming*. 2003: Lecture Notes in Computer Science.
19. Schiex, T., H. Fargier, and G. Verfaillie. *Valued Constraint Satisfaction Problems: Hard and Easy Problems*. in *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. 1995.
20. Bistarelli, S., U. Montanari, and F. Rossi, *Semiring-based Constraint Satisfaction and Optimization.* Journal of the ACM, 1997. **44**(2): p. 201-236.
21. Larrosa, J. *Node and Arc Consistency in Weighted CSP*. in *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth*

Conference on Innovative Applications of Artificial Intelligence (AAAI /IAAI). 2002: AAAI Press / The MIT Press.

22. Larrosa, J. and T. Schiex. *In the quest of the best form of local consistency for Weighted CSP*. in *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. 2003: Morgan Kaufmann.
23. Schiex, T. *Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints ?"* in *In Proceedings of the 8th Annual Conference on Uncertainty in Artificial Intelligence*. 1992: Morgan Kaufmann.
24. Dubois, D., H. Fargier, and H. Prade. *The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction*. in *In Proceedings of the Second IEEE International Conference on Fuzzy Systems*. 1993.
25. Fargier, H., J. Lang, and T. Schiex. *Selecting preferred solutions in fuzzy constraint satisfaction problems*. in *In Proceedings of the first European Congress on Fuzzy and Intelligent Technologies*. 1993.
26. Ruttkay., Z. *Fuzzy constraint satisfaction*. in *In Proceedings of the First IEEE Conference on Evolutionary Computing*. 1994.
27. Regin, J.C., et al. *An Original Constraint Based Approach for Solving over Constrained Problems*. in *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*. 2000: Springer.
28. Beldiceanu, N. and T. Petit. *Cost Evaluation of Soft Global Constraints*. in *Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2004: Springer.
29. Hoeve, W.-J.v., G. Pesant, and L.M. Rousseau, *On global warming: Flow-based soft global constraints.* Journal of Heuristics, 2006. **12**: p. 347- 373.
30. Burke, E.K. and G.Kendall, *Search Methodologies. Introductory Tutorials in Optimisation and Decision Support Techniques*. 2005: Springer.
31. Reeves, C., *Modern Heuristic Techniques for Combinatorial Problems*. 1995, NY, USA: John Wiley&Sons, Inc.
32. Rardin, R.L., *Optimization in Operations Research*. 1998: Prentice Hall, Inc.
33. *IBM ILOG User's Manual.*
34. Bacchus, F. and T. Walsh, *Propagating Logical Combinations of Constraints*, in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*. 2005. p. 35-40.
35. *Oxford Dictionary of Computing*. 1997: Oxford University Press.
36. Talbi, E., *Metaheuristics, from design to implementation*. 2009.
37. Osman, I.H. and J.P. Kelly, *Meta-Heuristics: Theory and Applications*. 1996: Kluwer Academic Publishers.
38. Blum, C. and A. Roli, *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison.* ACM Computing Surveys, 2003. **35**: p. 268-303.
39. Gilmore, P.C. and R.E. Gomory, *A linear programming approach to the cutting-stock problem.* Operations Research, 1961. **9**: p. 849- 859.
40. Mason, A.J. and M.C. Smith. *A Nested Column Generator forsolving Rostering Problems with Integer Programming*. in *International Conference on Optimisation: Techniques and Applicaitons*. 1998. Perth, Australia.

41. Jaumard, B., F. Semet, and T. Vovor, *A generalized linear programming model for nurse scheduling.* European Journal of Operational Research, 1998. **107**: p. 1-18.
42. Junker, U., et al., *A Framework for Constraint Programming Based Column Generation*, in *Principles and Practice of Constraint Programming*. 1999. p. 261- 275.
43. Yunes, T.H., A.V. Moura, and C.C. de Souza, *Solving very large crew scheduling problems to optimality*, in *ACM symposium on Applied computing*. 2000. p. 446- 451.
44. Demassey, S., G. Pesant, and L.-M. Rousseau, *A Cost-Regular Based Hybrid Column Generation Approach.* Constraints, 2006. **11**(4): p. 315- 333.
45. Fahle, T., et al., *Constraint Programming Based Column Generation for Crew Assignment.* Journal of Heuristics, 2002. **8**(1): p. 59-81.
46. Barnhart, C., et al., *Branch-and-price: Column generation for solving huge integer programs.* Operations Research, 1998. **46**: p. 316- 329.
47. Lubbecke, M.E. and J. Desrosiers, *Selected topics in column generation.* Operations Research, 2002. **53**: p. 1007- 1023.
48. Bixby, R., et al., *MIP:Theory and practice--closing the gap.* system modelling and optimization:methods,theory and applications, 2000. **174**: p. 19-49.
49. Lazic, J., et al., *Variable neighbourhood decomposition search for 0-1 mixed integer programs.* Computers &Operations Research, 2009. **37**(6): p. 1055-1067.
50. Puchinger, J., G. Raidl, and U. Pferschy. *The core concept for the multidimensional knapsack problem*. in *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006*. 2006.
51. Angelelli, E., R. Mansini, and G.S. M., *Kernel search: a new heuristic framework for portfolio selection.* Computational Optimization and Applications, 2009(DOI: 10.1007/s10589-010-9326-6).
52. Aickelin, U. and K. Dowsland, *Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem.* Journal of Scheduling, 2000. **3**(3): p. 139-153.
53. Burke, E.K. and J.P. Newall, *Solving examination tinetabling problems through adaptation of heuristic orderings.* Annals of Operations Research, 2004. **129**: p. 107-134.
54. Ikegami, A. and A. Niwa, *A Subproblem-centric Model and Approach to the Nurse Scheduling Problem.* Mathematical Programming, 2003. **97**(3): p. 517-541.
55. Aickelin, U. and K. Dowsland, *An indirect genetic algorithm for a nurse scheduling problem.* Journal of Operations Research Society, 2003. **31**(5): p. 761-778.
56. Brucker, P., et al., *A decomposition, construction and postprocessing approach for a specific nurse rostering problem*, in *Multidisciplinary International Scheduling Conference*. 2005: New York, USA. p. 397-406.
57. Raidl, G. and J. Puchinger, *Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization*, in *Hybrid Metaheuristics*. 2008. p. 31-60.

58. Focacci, F., F. Laburthe, and A. Lodi. *Local Search and Constraint Programming* in *Metaheuristics Interational Conference*. 2001.
59. Wallace, M., *Hybrid Algorithms in Constraint Programming*, in *Recent Advances In Constraints*. 2007, LNCS. p. 1-32.
60. Chandru, V. and J. Hooker, *Optimization Methods for Logical Inference*. 1999: Wiley.
61. Milano, M., *Constraint and Integer Programming: Toward a Unified Methodology (Operations Research/Computer Science Interfaces, 27)*. 2003: Kluwer Academic Publishers.
62. Gabteni, S. and M. Grönkvist, *A Hybrid Column Generation and Constraint Programming Optimizer for the Tail Assignment Problem* in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2006. p. 89-103.
63. Easton, K., G. Nemhauser, and M. Trick, *Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach*, in *Practice and Theory of Automated Timetabling IV*. 2003. p. 100-109.
64. Gualandi, S. and F. Malucelli, *Constraint Programming based Column Generation: a Survey.* 4OR, 2009.
65. Yunes, T.H., A.V. Moura, and C.C. de Souza, *Hybrid Column Generation Approaches for Urban Transit Crew Management Problems.* Transportation Science, 2005. **39**(2): p. 273- 288.
66. Grönkvist, M., *Using Constraint Propagation to Accelerate Column Generation in Aircraft Scheduling*, in *Principles and Practice of Constraint Programming*. 2002. p. 37- 48.
67. Grönkvist, M., *A Constraint Programming Model for Tail Assignment*, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2004. p. 142- 156.
68. Pisinger, D. and M. Sigurd, *Using Decomposition Techniques and Constraint Programming for Solving the Two-Dimensional Bin-Packing Problem.* INFORMS J. on Computing, 2007. **19**(1): p. 36- 51.
69. Sellmann, M., et al., *Crew Assignment via Constraint Programming: Integrating Column Generation and Heuristic Tree Search.* Annals of Operations Research, 2002. **115**(1): p. 207- 225.
70. Hansen, J. and L. Tomas, *Group Construction for Airline Cabin Crew: Comparing Constraint Programming with Branch and Price*, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2005. p. 228- 242.
71. Rousseau, L.M., et al., *Solving VRPTWs with constraint programming based column generation.* Annals of Operations Research, 2004. **13**(1): p. 199–216.
72. Gendron, B., H. Lebbah, and G. Pesant, *Improving the Cooperation Between the Master Problem and the Subproblem in Constraint Programming Based Column Generation*, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2005. p. 217- 227.
73. Rousseau, L.-M., *Stabilization Issues for Constraint Programming Based Column Generation*, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2004. p. 402- 408.

74. Gualandi, S., *Enhancing Constraint Programming-based Column Generation for Integer Programs*. 2008.
75. Jourdan, L., M. Basseur, and E.G. Talbi, *Hybridizing exact methods and metaheuristics: A taxonomy.* European Journal of Operational Research, 2009. **199**: p. 620-629.
76. Li, H., A. Lim, and B. Rodrigues, *A hybrid AI approach for nurse rostering problem*, in *ACM symposium on Applied computing*. 2003. p. 730- 735.
77. Rousseau, L.-M., G. Pesant, and M. Gendreau, *A General Approach to the Physician Rostering Problem.* Annals of Operations Research, 2002. **115**(1): p. 193- 205.
78. Lin, Y., *Directed Annealing Search in constraint satisfaction and optimazition*. 1997.
79. Pesant, G. and W. Nuijten, *A Constraint Programming Framework for Local Search Methods.* Journal of Heuristics, 1999. **5**: p. 255- 279.
80. Shaw, P., *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, in *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*. 1998, Springer-Verlag. p. 417-431.
81. Hentenryck, P.V. and L. Michel, *Constraint-Based Local Search*. 2005: The MIT Press.
82. Voudouris, C. and E.P.K. Tsang, *Guided local search*, in *Handbook of metaheuristics*, F. Glover, Editor. 2003, Kluwer. p. 185-218.
83. Cotta, C., et al., *Hybridizing genetic algorithms with branch and bound techniques for the resolution of the tsp*, in *Artificial Neural Nets and Genetic Algorithms*, D.W. Pearson, N.C. Steele, and R.F. Albrecht, Editors. 1995, Springer-Verlag. p. 277-280.
84. Jahuira, C.A.R. *Hybrid genetic algorithm with exact techniques applied to tsp*. in *Second International Workshop on Intelligent Systems Design and Application*. 2002: Dynamic Publishers.
85. Ahuja, R.K., et al., *A survey of very large-scale neighbourhood search techniques.* Discrete Appl. Math., 2002. **123**(1-3): p. 75-102.
86. Bent, R. and P.V. Hentenryck, *A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows.* Transportation Science, 2004. **38**(4).
87. Augerat, P., et al., *Separating capacity constraints in the CVRP using tabu search.* European Journal of Operational Research, 1998. **106**(2-3): p. 546-557.
88. Maniezzo, V., T. Stützle, and S. Voß, *Matheuristics - Hybridizing Metaheuristics and Mathematical Programming*, in *Annals of Information Systems*. 2010.
89. Cheang, B., et al., *Nurse rostering problems--a bibliographic survey.* European Journal of Operational Research, 2003. **151**(3): p. 447 - 460.
90. Ernst, A.T., et al., *Staff scheduling and rostering: A review of applications, methods and models.* European Journal of Operational Research, 2004. **153**(1): p. 3 - 27.
91. Curtois, T., *Novel Heuristic and Metaheuristic approaches to the Automated Scheduling of Healthcare Personnel*. 2007, Unversity of Nottingham.

92. Warner, D.M. and J. Prawda, *A Mathematical Programming Model for Scheduling Nursing Personnel in a Hospital.* Management Science, 1972. **19**(4): p. 411-422.
93. Bailey, J., *Integrated days off and shift personnel scheduling.* Computing and Industrial Engineering, 1985. **9**(4): p. 395-404.
94. Bard, J.F. and H.W. Purnomo, *Preference scheduling for nurses using column generation.* European Journal of Operational Research, 2005. **164**(2): p. 510 - 534.
95. Bard, J.F. and H.W. Purnomo, *A column generation-based approach to solve the preference scheduling problem for nurses with downgrading.* Socio-Economic Planning Sciences, 2005. **39**(3): p. 193 - 213.
96. Maenhout, B. and M. Vanhoucke, *Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem.* Journal of Scheduling, 2010. **13**(1): p. 77-93.
97. Darmoni, S.J., et al., *Horoplan: computer-assisted nurse scheduling using constraint-based programming.* Journal of Society for Helth Systems, 1995. **5**: p. 41-54.
98. Weil, G.K., et al., *Constraint Programming for nurse scheduling.* IEEE Engineering in Medicine and Biology Magazine, 1995. **14**(4): p. 417-422.
99. Cheng, B.M.W., J.H.M. Lee, and J.A.C.K. Wu, *A nurse rostering system using constraint programming and redundant modelling.* IEEE Transactions on information technology in biomedicine, 1997. **1**(1): p. 44- 54.
100. Metivier, J.-P., P. Boizumault, and S. Loudni, *Solving Nurse Rostering Problems Using Soft Global Constraints* in *Principles and Practice of Constraint Programming*. 2009, Lecture Notes in Computer Science. p. 73-87.
101. Wong, G.Y.C. and A.H.W. Chun, *Constraint-based rostering using meta-level reasoning and probability-based ordering.* Engineering Applications of Artificial Intelligence, 2004. **17**: p. 599- 610.
102. Hofe, H.M.A.m., *Nurse rostering as constraint satisfaction with fuzzy constraints and inferred control strategies*, in *DIMACS workshop on on Constraint programming and large scale discrete optimization*. 2001, American Mathematical Society: Rutgers Univ., Piscataway, New Jersey, United States. p. 67-101.
103. Markowitz, H.M., *Portfolio Selection.* J. Finance, 1952. **7**: p. 77-91.
104. Hiroshi, K. and Y. Hiroaki, *Mean-absolute deviation portfolio optimization model and its applications to Tokyo stock market.* Management Science, 1991. **37**(5): p. 519-531.
105. Speranza, M.G., *Linear programming models for portfolio optimization.* Finance, 1993. **14**: p. 107-123.
106. Mansini, R. and M.G. Speranza, *An exact approach for portfolio selection with transaction costs and rounds.* IIE Transactions, 2005. **37**: p. 919-929.
107. Rockafellar, R.T. and S. Uryasev, *Optimization of conditional value-at-risk* Journal of risk, 2000. **2**: p. 21-41.
108. Crama, Y. and M. Schyns, *Simulated annealing for complex portfolio selection problems.* European Journal of Operational Research, 2003. **150**(3): p. 546-571.
109. Chang, T.J., et al., *Heuristics for cardinality constrained portfolio optimisation.* Computers & Operations Research, 2000. **27**(13): p. 1271-1302.

110. Schaerf, A., *Local Search Techniques for Constrained Portfolio Selection Problems.* Computational Economics, 2002. **20**(3): p. 177-190.
111. Di Gaspero, L., et al., *Hybrid Local Search for Constrained Financial Portfolio Selection Problems*, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2007. p. 44-58.
112. Jobst, N.J., et al., *Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints* Quantitative Finance 2001. **1**(5): p. 489-501.
113. Mitra, G., F. Ellison, and A. Scowcroft, *Quadratic programming for portfolio planning: Insights into algorithmic and computational issues* J of Asset Manag, 2007. **8**(3): p. 200-214.
114. Bonami, P. and M.A. Lejeune, *An Exact Solution Approach for Portfolio Optimization Problems Under Stochastic and Integer Constraints.* OPERATIONS RESEARCH, 2009. **57**(3): p. 650-670.
115. Burke, E.K., et al., *A hybrid heuristic ordering and Variable Neighourhood Search for the nurse rostering problem.* European Journal of Operational Research, 2008. **188**(2): p. 330-341.
116. Hooker, J.N., *Dictionary of Constraints*, in *Integrated Methods for Optimization*. 2007, Springer. p. 414-447.
117. Regin, J.C. and J.F. Puget., *A filtering algorithm for global sequencing constraint*, in *Principles and Practice of Constraint Programming*. 1997, Springer. p. 32-46.
118. Bourdais, S. and P. Galinier, *HIBISCUS: A Constraint Programming Application to Staff Scheduling in Health Care*, in *Principles and Practice of Constraint Programming*. 2003. p. 153-167.
119. Focacci, F., A. Lodi, and M. Milano, *Cost-Based Domain Filtering*, in *Principles and Practice of Constraint Programming – CP'99*. 1999. p. 189-203.
120. Burke, E.K., et al., *A Time Pre-defined Variable Depth Search for Nurse Rostering.* Accepted at INFORMS Journal on Computing, 2011.
121. Burke, E.K., et al., *A Scatter Search for the Nurse Rostering Problem.* Journal of Operational Research Society, 2010. **61**: p. 1667-1679.
122. Muller, T., R. Bartak, and H. Rudova, *Iterative forward search algorithm: combining local search with maintaining arc consistency and a conflict-based statistics*, in *Lecture Notes in Computer Science*. 2004, Springer: Berlin. p. 802-817.
123. Burke, E.K., J. Li, and R. Qu, *A Hybrid Model of Integer Programming and Variable Neighourhood Search for Highly-constrained Rostering Problems.* European Journal of Operational Research, 2009. **203(2)**: p. 484-493.
124. Hansen, P. and N. Mladenovic, *Variable Neighbourhood Search: Principles and Applications.* European Journal of Operational Research, 2001. **130**: p. 449-467.
125. Fijn van Draat, L., et al., *Harmonious personnel scheduling.* Medium Econometrische Toepassingen, 2006. **14**: p. 4-7.
126. Kilby, P., P. Prosser, and P. Shaw, *A Comparison of Traditional and Constraint-based Heuristic Methods on Vehicle Routing Problems with Side Constraints.* Constraints, 2000. **5**(4): p. 389-414.
127. Dowsland, K.A., *Nurse scheduling with tabu search and strategic oscillation.* European Journal of Operational Research, 1998. **106**(2-3): p. 393-407.

128. Nareyek, A., *Using global constraints for local search*, in *DIMACS workshop on on Constraint programming and large scale discrete optimization*. 2001, American Mathematical Society: Rutgers Univ., Piscataway, New Jersey, United States. p. 9-29.
129. Perron, L., P. Shaw, and V. Furnon, *Propagation Guided Large Neighbourhood Search*, in *Principles and Practice of Constraint Programming – CP 2004*. 2004. p. 468-481.
130. Agren, M., P. Flener, and J. Pearson, *Revisiting constraint-directed search.* Inf. Comput., 2009. **207**(3): p. 438-457.
131. Applegate, D. and W. Cook, *A computational study of the job-shop scheduling problem.* ORSA Journal on Computing 1991. **3**: p. 149-156.
132. Pisinger, D. and S. Ropke, *A general heuristic for vehicle routing problems.* Computers & Operations Research, 2007. **34**(8): p. 2403-2435.
133. Qu, R. and F. He. *A Hybrid Constraint Programming Approach for Nurse Rostering Problems*. in *Applications and Innovations in Intelligent Systems XVI*. 2008.
134. Morris, J.G. and M.J. Showalter, *Simple approaches to shift, days-off, and tour scheduling programs.* Management Science, 1983. **29**: p. 942-950.
135. Billionnet, A., *Integer programming to schedule a hierarchical workforce with variable demands.* European Journal of Operational Research, 1999. **114**(1): p. 105 - 114.
136. Beaumont, N., *Scheduling staff using mixed integer programming.* European Journal of Operational Research, 1997. **98**(3): p. 473 - 484.
137. Walsh, T. *Depth-bounded discrepancy search*. in *Proceedings of IJCAI-97* 1997.
138. Vanderbeck, F. and L.A. Wolsey, *An exact algorithm for IP column generation.* Operations Research Letters, 1996. **19**(4): p. 151-159.
139. Sol, M. and M.W.P. Savelsbergh, *Column generation techniques for pickup and delivery problems*, in *Eindhoven University of Technology*. 1994.
140. Focacci, F., A. Lodi, and M. Milano, *Optimization-Oriented Global Constraints.* Constraints, 2002. **7**(3): p. 351-365.
141. Petit, T., J.-C. Régin, and C. Bessière, *Specific Filtering Algorithms for Over-Constrained Problems*, in *Principles and Practice of Constraint Programming — CP 2001*. 2001. p. 451-463.
142. Bienstock, D., *Computational study of a family of mixed-integer quadratic programming problems.* Math. Program., 1996. **74**(2): p. 121-140.
143. Byrne, P. and S. Lee, *Different risk measures: Different portfolio compositions?* J. Property Investment & Finance, 2004. **22**(6): p. 501-511.
144. J. T.Linderoth and M.V.P. Savelsbergh., *computational study of branch and bound search strategies for mixed integer programming.* INFORMS J. Comput., 1999. **11**: p. 173-187.
145. Achterberg, T., *Constraint Integer Programming*. 2007, University of Berlin.
146. Gao, Z., S. Zhang, and X. Sun, *Matrix decomposition and Lagrangian dual method for discrete portfolio optimization under concave transaction costs.* Journal of Shanghai University (English Edition), 2009. **13**(2): p. 119-122.
147. Wallace, C., *ZI round, a MIP rounding heuristic.* Journal of Heuristics, 2010. **16**: p. 715-722.

148. Beasley, J.E., *OR-Library: distributing test problems by electronic mail.* Journal of the Operational Research Society, 1990. **41**(11): p. 1069-1072.
149. Konno, H. and A. Wijayanayake, *Portfolio optimization problem under concave transaction costs and minimal transaction unit constraints.* Mathematical Programming, 2001. **89**(2): p. 233-250.
150. Konno, H. and A. Wijayanayake, *Portfolio optimization under D.C. transaction costs and minimal transaction unit constraints.* Journal of Global Optimization, 2002. **22**(1): p. 137-154.
151. Konno, H. and R. Yamamoto, *Global Optimization Versus Integer Programming in Portfolio Optimization under Nonconvex Transaction Costs.* Journal of Global Optimization, 2005. **32**(2): p. 207-219.
152. Balas, E., S. Schmieta, and C. Wallace, *Pivot and shift - a mixed integer programming heuristic.* Discrete Optimization, 2004. **1**(1): p. 3-12.
153. Balas, E. and C.H. Martin, *Pivot-and-Complement: A Heuristic for 0-1 Programming.* Management Science, 1980. **26**(1): p. 86-96.
154. Johnson, E.L., G.L. Nemhauser, and M.V.P. Savelsbergh., *Progress in linear programming-based algorithms for integer programming: An exposition.* INFORMS J. Comput., 2000. **12**(1): p. 2-23.
155. Linderoth, J.T. and M.V.P. Savelsbergh, *computational study of branch and bound search strategies for mixed integer programming.* INFORMS J. Comput., 1999. **11**: p. 173-187.
156. Fischetti, M. and A. Lodi, *Local branching.* Mathematical Programming, 2003. **98**(1): p. 23-47.
157. Danna, E., E. Rothberg, and C.L. Pape, *Exploring relaxation induced neighbourhoods to improve MIP solutions.* Mathematical Programming, 2005. **102**(1): p. 71-90.
158. Pierre, H., M. Nenad, and U. Dragan, *Variable neighbourhood search and local branching.* Computer & Operations Research, 2006. **33**(10): p. 3034-3045.
159. Arnott, R.D. and W.H. Wagner, *The measurement and control of trading costs.* Financial analusts journal, 1990. **4**(6): p. 73-80.
160. Yoshimoto, A., *The mean variance approach to portfolio optimization subject to transaxtion costs.* journal of the operational research society of Japan, 1996. **39**(1): p. 99-117.
161. Mansini, R. and M.G. Speranza, *Heuristic algorithms for the portfolio selection problem with minimum transaction lots.* European Journal of Operational Research, 1999. **114**(2): p. 219-233.
162. Kellerer, H., R. Mansini, and M.G. Speranza, *Selecting Portfolios with Fixed Costs and Minimum Transaction Lots.* Annals of Operations Research, 2000. **99**(1): p. 287-304.
163. Gao, Z.-x., S.-t. Zhang, and X.-l. Sun, *Matrix decomposition and Lagrangian dual method for discrete portfolio optimization under concave transaction costs.* Journal of Shanghai University (English Edition), 2009. **13**(2): p. 119-122.
164. Lobo, M., M. Fazel, and S. Boyd, *Portfolio Optimization with Linear and Fixed Transaction Costs.* Annals of Operations Research, 2007. **152**(1): p. 341-365.
165. Padberg, M. and G. Rinaldi, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems.* Journal SIAM Review, 1991. **33**(1): p. 60-100.

166. Savelsbergh, M.W.P., *A Branch-and-Price Algorithm for the Generalized Assignment Problem.* Operations Research, 1997. **45**: p. 831-841.