AN INVESTIGATION OF NOVEL APPROACHES FOR OPTIMISING RETAIL SHELF SPACE ALLOCATION

by

Ruibin Bai, BEng, MSc

Thesis Submitted to The University of Nottingham
for the Degree of Doctor of Philosophy

School of Computer Science & Information Technology
The University of Nottingham, Nottingham, UK

September 2005

CONTENTS

List of Figures

List of Tables

Abstract

This thesis is concerned with real-world shelf space allocation problems that arise due to the conflict of limited shelf space availability and the large number of products that need to be displayed. Several important issues in the shelf space allocation problem are identified and two mathematical models are developed and studied. The first model deals with a general shelf space allocation problem while the second model specifically concerns shelf space allocation for fresh produce. Both models are closely related to the knapsack and bin packing problem.

The thesis firstly studies a recently proposed generic search technique, hyper-heuristics, and introduces a simulated annealing acceptance criterion in order to improve its performance. The proposed algorithm, called simulated annealing hyper-heuristics, is initially tested on the one-dimensional bin packing problem, with very promising and competitive results being produced. The algorithm is then applied to the general shelf space allocation problem. The computational results show that the proposed algorithm is superior to a general simulated annealing algorithm and other types of hyper-heuristics. For the test data sets used in the thesis, the new approach solves every instance to over 98% of the upper bound which was obtained via a two-stage relaxation method.

The thesis also studies and formulates a deterministic shelf space allocation and inventory model specifically for fresh produce. The model, for the first time, considers the freshness condition as an important factor in influencing a product's demand. Further analysis of the model shows that the search space of the problem can be reduced by decomposing the problem into a nonlinear knapsack problem and a single-item inventory problem that can be solved optimally by a binary search. Several heuristic and meta-heuristic approaches are utilised to optimise the model, including four efficient gradient based constructive heuristics, a multi-start generalised reduced gradient (GRG) algorithm, simulated annealing, a greedy randomised adaptive search procedure (GRASP) and three different types of hyper-heuristics. Experimental results show that the gradient based constructive heuristics are very efficient and all meta-heuristics can only marginally improve on them. Among these meta-heuristics, two simulated annealing based hyper-heuristic performs slightly better than the other meta-heuristic methods.

Across all test instances of the three problems, it is shown that the introduction of simulated annealing in the current hyper-heuristics can indeed improve the performance of the algorithms. However, the simulated annealing hyper-heuristic with random heuristic selection generally performs best among all the other meta-heuristics implemented in this thesis.

This research is funded by the Engineering and Physical Sciences Research Council (EPSRC) grant reference GR/R60577. Our industrial collaborators include Tesco Retail Vision and SpaceIT Solutions Ltd.

Acknowledgements

Sincere gratitude to my supervisor, Dr. Graham Kendall, for his trust, patience, kind help, support and valuable guidance throughout my PhD. I am also extremely grateful to him for giving this opportunity to allow me to pursue a PhD at the University of Nottingham.

Deep appreciation to Professor Edmund Burke and all the members in ASAP group for their help and assistance.

Dedicated to my wife, Jumei, for her support and encouragement throughout the three years of my PhD.

Dedicated to my parents and brothers, for without their encouragement and inspiration I would not have come to the UK to pursue my research.

I am highly grateful for the financial funding from the EPSRC and the School of Computer Science, University of Nottingham. It would have not been possible for me to take this PhD programme without their support. Many thanks to our industrial collaborators for their valuable discussions with regard to the project.

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature:

CHAPTER 1. INTRODUCTION

1.1 Background and Motivations

The retailing sector in the UK is an extremely competitive arena. We only need to consider some high profile companies to see that this is the case. A particular example is provided by the recent decline of Marks and Spencer, who used to be the leading high street retailer. A further example is given by C&A's decision to close all of its high street retail outlets. Yet another example is the decline of J Sainsburys from its position as the leading food retailer in the UK in the 1990's (in 1996, Tesco opened up a 2% lead over their rivals and continue to maintain an advantage). Asda, after merging with Wal-Mart, increased its market share dramatically and overtook Sainsbury's as the second biggest supermarket in the UK. In July 2003, Asda had gained a 17% market share, while Sainsbury's had slipped from 17.1% to 16.2%. Tesco retains the top spot with 27% of the overall market (BBC Business, 2003). This trend is continuing with Tesco's market share increasing further to 29% with a total of £29.5 billion of domestic sales in 2004. However, Morrisons, since taking over Safeways in 2004, has been struggling to lift their sales and profits (BBC Business, 2005).

This level of competitiveness is unlikely to decline. On the contrary, the high street (or more likely, out of town shopping centres) is likely to become even more competitive.

Retailers are keen to do everything possible to make their systems more efficient, whilst maximising their profit. Several tactics are used to influence consumers' purchases, including product assortment (deciding which merchandise to sell), store layout and space planning, merchandise pricing, services offered, advertising and other promotional programs (Levy and Weitz, 1992). Store layout and space planning focuses on the improvement of the visual effect of the shopping environment and space productivity.

Shelf space allocation also uses the term *planograms*. A planogram is a retailer's product map or blueprint which shows exactly where and how many items should physically be displayed on the shelves or fixtures (see figure 1-1 for an example).

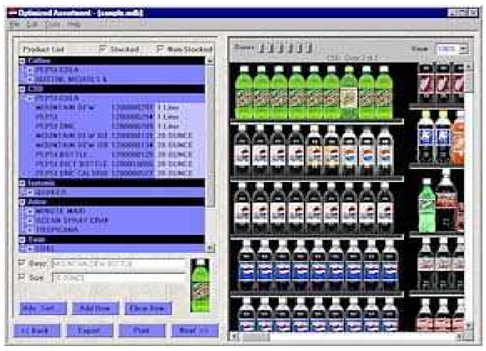


Figure 1-1: An example of a simple planogram

If customers are completely loyal to the products that they buy and all purchases are planned before a visit to the shop, shelf space manipulation, both in terms of volume and the location where a product displayed, would not be able to boost sales as long as 'out-of-stock' issues do not occur. However, unplanned (occasional) purchases are very common. An attractive layout of the products could increase impulse purchases. Previous research shows that unplanned purchases make up about one third of all transactions in many retail stores (Buttle, 1984). Therefore, shelf space allocation is an area worthy of investigation in which retailers have the opportunity to increase their sales.

However, allocating shelf space to hundreds or even thousands of products is challenging. On one hand, shelf space is an expensive and scarce resource for retailers. They would prefer not to increase the store size due to the high costs of construction as well as maintenance. Dreze et al. (Dreze et al., 1994) reported a shelf cost of $20/square foot for dry shelf space, and rising to over $50/square foot for dairy shelves. The costs can even rise to about $70/square foot for frozen food. On the other hand, many supermarkets are selling upwards of thousands of different products on a daily basis and this number is continuing to rise as retailers strive to diversify their product lines to more and more non-food products (Dreze et al., 1994; Yang and Chen, 1999). For example, a general Tesco store carries about 30,000 different products or stock-keeping units (SKU) and a Tesco hypermarket sells more than 50,000 different items. This poses a real dilemma for the supermarkets. The space allocation has to balance the conflict of thousands of products to display, versus the limited amount of space at their disposal.

Research and practice reveals that planograms, especially computer-based planograms, are one of the most important aspects used to improve the financial performance of a retail outlet and can also be used for inventory control and vendor relation improvement (Levy and Weitz, 1992; Yang and Chen, 1999). However, generating planograms is a challenging and time-consuming process because the simplest form of planogram problem (ignoring all marketing and retailing variables) is already a multi-knapsack problem, a well-known NP-Hard problem (Martello and Toth, 1990a) which is very difficult to solve. The difficulty is further increased when we consider other merchandise, such as fresh food, clothing and frozen food. This is due to their special display requirements and the fact that they do not use standard shelf fitments. Currently, producing planograms is largely a manual process (there is software assistance available (e.g. Galaxxi, Spaceman) but most are drag-and drop procedures or semi-automated processes which involve significant human interaction) and the shelf space allocation is mainly based on some simple rules. Examples of the rules include allocating space proportional to a product's market share, historical sales, profit or a combination of these (Corstjens and Doyle, 1981b). However, these simple approaches may lose substantial sales according to (Borin et al., 1994).

Yang and Chen (Yang and Chen, 1999) conducted a survey of the area. This work highlighted the lack of academic work that has been conducted in this domain. Only twelve references were cited. Five of these date back to the 1970's, four were drawn from the 1980's and only three were from the 1990's. It seems timely that this area should receive research attention given the recent advances in AI search techniques.

1.2 Scope and Aims

The thesis is based on a research proposal which is funded by EPSRC (GR/R60577), in collaboration with three other industrial collaborators: Tesco, Retail Vision and SpaceIT Solutions Ltd. Throughout this project, we have had several meetings with them, which has proven to be very useful and valuable. The software provided by SapceIT Solutions Ltd allowed us to recognise the shortcomings of current planogram software. The conversations with John Ibbotson from Retail Vision helped us to understand the key issues of the problem, while the conversations with Tesco helped us direct our research attention to a more interesting problem (fresh produce) in the latter stage of the project.

Overall, the aim of this research is to develop models and algorithms that can be used in the next generation of planogram systems. The software should be able to not only produce automated planograms but also provide optimised shelf space allocation solutions for the given requirements. Specifically, we want to:

1. Identify potential important issues in the shelf space allocation problem;
2. Formulate a practical model that captures the main characteristics of various shelf space allocation problems and one that can be used in practice;

3. Specifically investigate a fresh food inventory control and shelf space allocation problem, which is of particular interest to retailers.
4. Identify the relationship between the shelf space allocation problems and other space allocation problems and investigate potential optimisation techniques for those problems;
5. Investigate meta-heuristics, especially a simulated annealing hyper-heuristic for the optimisation the model formulated in 2 and 3.

1.3 Overview of the Thesis

The thesis is organised as follows: chapter two introduces and discusses the shelf space allocation problem. Several important issues are identified and discussed. Previous research of the shelf space allocation problem is then reviewed and several other related space allocation problems are briefly discussed. Chapter three overviews the optimisation techniques that can be used for combinatorial optimisation problems. Chapter four specifically studies a recently emerging generic search technique, hyper-heuristics. A simulated annealing acceptance criterion is proposed that can be included in the hyper-heuristic framework in order to further improve its performance. The resulting algorithms are initially tested on the well-known bin packing problem. Chapter five proposed a practical model for a general shelf space allocation problem. To have a better measure of the solution quality and algorithm performance, an upper bound of the problem is obtained by a two-stage relaxation. Several hyper-heuristic approaches are implemented and applied to the problem and their performances are analysed and compared on two simulated data sets. The advantages of simulated annealing hyper-heuristics are discussed in comparison to two conventional simulated annealing algorithms and other types of hyper-heuristic algorithms.

Chapter six and seven investigate a shelf space allocation problem specifically for fresh produce. This problem differs from the general shelf space allocation problem in that the products deteriorate continuously over time and their freshness plays a vital role in influencing customers' demand. In chapter six, a practical shelf space allocation and inventory control model is proposed which, for the first time, uses the concept of freshness condition to formulate the fresh food demand function. Further analysis shows that the proposed model is an extension of the non-linear bounded knapsack problem. A generalised reduced gradient algorithm (GRG) is proposed and extended in order to optimise the problem. Chapter seven investigates several heuristic and meta-heuristic approaches for the problem model formulated in chapter six. Four efficient gradient based heuristics are firstly proposed and several meta-heuristic approaches, including the simulated annealing hyper-heuristics, are investigated to further improve the solutions from these greedy approaches.

1.4 Contributions

The work in this thesis makes the following contributions:

- Several important issues in the shelf space allocation problem are identified and a practical model for the general shelf space allocation problem is proposed. An upper bound of the model is derived via a two-stage relaxation method.
- The thesis, for the first time, adapts existing hyper-heuristics to the shelf space allocation problem.
- The thesis, for the first time, introduces simulated annealing into a hyper-heuristic framework which could potentially improve the performance and robustness of current hyper-heuristic approaches.

- A simulated annealing hyper-heuristic is successfully applied to the one-dimensional bin packing problems with competitive results being produced when compared with other state-of-the-art methods.
- A deterministic inventory control and shelf space allocation model is formulated for the retailing of fresh food, which, for the first time, considers the freshness condition as a potential demand influencing factor. We consider this as a major contribution of this thesis.
- A GRG (generalised reduced gradient) algorithm is extended and adapted to the inventory control and shelf space allocation problem.
- Several heuristics and meta-heuristics are designed and developed to optimise the fresh produce shelf space allocation model, including a GRASP algorithm, a simulated annealing algorithm, a tabu search hyper-heuristic and a simulated annealing hyper-heuristic. Their performance is compared and discussed.
- The investigation of simulated annealing hyper-heuristics on three different space allocation problems presents a better understanding of both simulated annealing algorithms and hyper-heuristics. Due to its success for all three problems, the author strongly believes the algorithm is also a promising research direction for some other combinatorial optimisation problems.

1.5 List of Presentations

Bai, R. and Kendall, G., Optimisation of Supermarkets Shelf Space Allocation. The Third EPSRC PhD Student Workshop on Scheduling, 12 May 2003, University of Bradford, UK.

Bai, R. and Kendall, G., An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics, The Fifth Metaheuristics

International Conference (MIC 2003), 23-25 August 2003, Kyoto International Conference Hall, Kyoto, Japan.

Bai, R. and Kendall, G., Recent Advances in the Production of Automated Planograms. The CORS/INFORMS Joint International Meeting, 16-19 May 2004, Banff, Canada.

Bai, R. and Kendall, G., Designing Efficient Low-level Heuristics in the Hyper-heuristic Framework. OR47, 13-15 September 2004, Chester, UK.

1.6 List of Publications

Bai, R. and Kendall, G., 2005a. A Model for Fresh Produce Shelf Space Allocation and Inventory Management with Freshness Condition Dependent Demand, Accepted for publication in the INFORMS Journal on Computing.

Bai, R. and Kendall, G., 2005b. An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics, in: Ibaraki, T., Nonobe, K., and Yagiura, M. (Eds.), Metaheuristics: Progress as Real Problem Solvers - (Operations Research/Computer Science Interfaces, Vol. 32), Berlin, Heidelberg, New York, Springer, pp. 87-108.

An early version of this paper is also published in the Proceeding of the 5th Metaheuristics International Conference (MIC 2003), Kyoto, Japan, Aug. 25-28, 2003.

Bai, R. and Kendall, G., 2005c. A Multi-heuristic Simulated Annealing for the One-dimensional Bin Packing Problem, Submitted to EJOR.

Bai, R. and Kendall, G., 2005d. Heuristic and Meta-heuristics for the Optimisation of a Fresh Produce Inventory Control and Shelf Space Allocation Problem, Submitted to Journal of Operational Research Society.

CHAPTER 2. THE SHELF SPACE ALLOCATION PROBLEM AND RELATED WORK

2.1 Introduction

The shelf space allocation problem is a real-world problem faced by many retail companies. The problem arises when there is a large number of products to display, but with limited shelf space available at disposal. This chapter firstly introduces the shelf space allocation problem and analyses the necessities and benefits of providing an automated solution methodology. A detailed description of the problem is then presented along with some possible hard and soft constraints. The previous research on shelf space allocation is then reviewed. It is shown that shelf space allocation problems share some similarities with some well-known capacity allocation problems, such as the bin packing and knapsack problems which are also reviewed in the chapter.

2.2 The Shelf Space Allocation Problem

The shelf space allocation problem involves distributing the scarce shelf space among different products held within a retail store.

2.2.1 Problem description

Firstly, let us introduce some concepts related to shelf space allocation. The first term is a *stock-keeping unit* (SKU) which is used to uniquely identify a specific product or goods. SKU is the smallest management unit in a retail store. *Inventory* refers to the quantity of each SKU that is currently held by a retailer. Keeping a minimum inventory could reduce or avoid the occurrence of *out-of-stock*. A *category* is a collection of products that have the same or similar functions or attributes. A category usually contains several *brands* with each brand having several SKU, usually corresponding to different sizes, colours, flavours and/or other properties. *Facing* is a very important variable for shelf space allocation. The number of the facings of a SKU is the quantity of an item that can be directly seen on the shelves or fixtures by the customers. The items placed behind other items cannot be seen directly and hence are not deemed as a facing. Note that a retailer normally only displays part of the inventory of a given item on the shelves (leaving the rest in the backroom) due to the limited amount of shelf space. This means that the number of facings of a SKU, or the amount of visible stock on the shelves, is normally less than the inventory.

During the last fifty years, the variety of available products has increased dramatically, and continues to do so in order to meet the diverse demands of customers. This diversity can be due to the different functions, brands, styles, colours, materials and even sizes, as well as many other factors. Although the supermarkets have continuingly increased their store sizes, the proportion of this increase is far less than the increase in the variety of the products. This creates a real challenge for most of the supermarkets in pursuit of effective product layouts such that some objectives are achieved, for example, maximising profit or sales, minimising operating costs, maximising customers’ loyalty, etc.

Shelf space allocation problems can be very different. This is due to the differences in a company’s long-term strategy, management style, categories of the products, competitive environment, retailer-vendor relationship, store layout, store size, fixture structure, etc. It is unlikely that we can develop a mathematical model which could exactly represent every real-world shelf space allocation problem. Therefore, for the purpose of this research, this thesis will mainly focus on an abstracted problem which can capture the main characteristics of the shelf space allocation problems existing in most retail stores.

In practise, space allocation, in a retail store, is usually decomposed into two levels: space allocation among categories and the space allocation for each SKU. The reasons are that: 1. A supermarket usually has thousands of products. Different categories may have different display conditions and requirements. Solving the problem for all products is unrealistic not only because of the difficulty in formulating a suitable model for all products from different categories but also because of the extremely high computational requirements. 2. Grouping similar functional products into a category allows customers to compare them before making a choice. 3. Category management is a common method for most stores, especially big supermarkets which are usually hierarchised into department, category, brand and stock-keeping unit (SKU) (Levy and Weitz, 1992; Yang and Chen, 1999; Gruen and Shah, 2000).

Buttle (Buttle, 1984) described a general retail store space planning process and listed several important in-store manipulation tactics to stimulate demand: *traffic flow design*, *category and brand location*, *space allocation to each category and product*, *point-of-sale (POS) promotions* and *special display*. Given a store with a given size, a retailer firstly needs to design a customer's traffic flow that will be guided by the fixtures and shelves in such a way that every part of the store has maximal exposure while customers can also have direct access to the section(s) that they wish to visit. The dimension and layout of the shelves are mainly determined by the store's physical shape and the customer traffic flow pattern that a retailer chooses. Once the shelves (or fixtures) have been placed within a store, a retailer has to make space allocation among different categories and then among different SKU in each category.

The space allocation among different categories is more related to the company's long-term strategies, competitive situation and customer purchasing habits. This aspect is beyond the scope of this thesis. In the following chapters, without specification, the shelf *space allocation problem* will refer to the problem of allocating space for each SKU within a given category. Several issues need to be taken into consideration:

Objectives

The ultimate aim of shelf space allocation is to either reduce cost or maximise the overall profit. Minimising the cost is used in EOQ (economic order quantity) models where the demand of a product is fixed and the space allocation does not influence the demand. However, if a product's demand is dependent on the decision variables of the shelf space allocation, a cost-minimisation objective becomes inappropriate because the minimisation of the cost would result in a decline of sales and profit because the model may try to reduce the product facings in order to reduce cost. However, the reduction of displayed shelf space may also lead to a decrease in sales and thus profit. To take an extreme case as an example, when no shelf space is assigned, the cost is minimal. However, clearly, no product can be sold when it is not displayed. Therefore, the aggregate profit maximisation is chosen to be the objective of the shelf space allocation problem in this thesis.

Decision variables

Facings and location are the two most common shelf space allocation variables.

Facing is a very important variable for shelf space allocation. It has been established that the number of facings has an important influence on customer purchases. Research has found that more than 33% of purchases are unplanned (Buttle, 1984). Products with a better exposure have a greater chance of being purchased by customers. However, the allocated shelf space may have a different impact on sales from one product to another.

Space elasticity is usually used to measure the responsiveness of the sales with regards to the change of allocated space. Curhan (Curhan, 1972) defined space elasticity as "*the ratio of relative change in unit sales to relative change in shelf space*".

<u>*Location*</u> is another variable which can influence the demand of a product. It is generally believed that shelves at eye-level ("eye-level is buy-level"), shelves at the end of aisles and at the store entrance are better positions, while top and bottom shelves are less important. However, there are some arguments with regards to the horizontal distances. Some research shows that the shelves at both ends of the aisles are better than the middle positions, while others believe that customers prefer middle locations as opposed to the ends of the aisles (Dreze et al., 1994; Ibbotson, 2002). These findings are based on the fact that some customers prefer to take the first item once they enter an aisle whilst others take time to "acclimatise" themselves and so ignore the first few items.

There are other marketing variables that are used to stimulate sales, including advertising, promotion, discounting, etc. Investigation of these issues is beyond the aim of this research. Our focus is on the facings and location variables.

<u>Constraints</u>

There are several potential constraints for the shelf space allocation problem. Although different stores may have different display requirements and considerations and thus have different constraints, there are some which are common.

Physical constraints are applied to every shelf space allocation problem. The total volume of the items assigned to a shelf cannot exceed the total shelf space available. This constraint can be one dimensional (ignoring the height and depth constraints) or two dimensional (ignoring depth constraints). The depth constraints are usually ignored because the depth of the shelf is usually much larger than the width of the SKU. The retailers do put as many items as possible behind the front items in order to reduce the number of replenishment times, however, the existence of stock behind the front facings has no effect on the demand function. The height constraints can also be ignored for some goods, for example, when placing products on top of another is not allowed (e.g. wine and milk bottles, etc). Also in many stores, the height of shelves can be adjusted. This could solve the problem when the product height exceeds the height of the shelf the product is assigned to or when there is not enough space for picking the goods from shelves.

Physical constraints are generally considered as *hard constraints*. That is: violation of these constraints will result in an infeasible solution (for some products, which can be "squeezed a bit", this constraint is not in a strong sense "hard" anymore).

Integrality constraints. Due to the fact that the physical products cannot be sub-divided (at least for most products), the space allocated to an item should be an integral times of the size of that item, usually measured by facings. This is also a hard constraint and must be satisfied. It does not make any sense to allocate 1.5 facings space to an item.

The physical constraints and integrality constraints of the shelf space allocation problem are very similar to the constraints in bin packing and knapsack problems, which are well-known NP-Hard problems (Martello and Toth, 1990a). However, a shelf space allocation problem may be even more difficult because it usually has a non-linear objective function and some additional constraints, which will be discussed in the following paragraphs.

Display requirements. Many retailers set a lower bound on the number of facings allocated to a product to ensure that the necessary exposure is given to the customers (In Tesco, for example, the minimal display space for a product is two facings). An upper bound is also enforced so that the number of facings is contained within reasonable values. In some cases important suppliers also have the power to influence the shelf space allocation decision, requiring more space and better location for their brands.

Block constraint. A block constraint is required based on the assumption that a SKU has a higher chance of being purchased by bundling several facings of a SKU together rather than spreading them onto different shelves. However, it may also be the case that putting the same product in several places throughout the store could increase purchases.

Adjacency. Although it may be reasonable that putting similar products of different brands together may make it easier for customers to make comparisons, it is also sensible to display complimentary products together by assuming that buying one product may encourage the customer to make another purchase for a complementary product (for example, beer and crisps, tea and biscuits, greeting cards and flowers and toothpaste and toothbrush).

Weight constraint. A weight constraint is necessary when the products are relatively heavy and the total product weight should not exceed the weight limit a shelf can sustain. Another consideration is that large and heavy products should be displayed on a lower shelf to allow easier access to the products both for customers and staff.

2.2.2 An overview of shelf space allocation

In this section, we shall give a review on the research and practice of shelf space allocation. In the literature, shelf space allocation research has been carried out both on the experimental studies and optimisation studies. The experimental studies are concerned with the effects of shelf space related tactics and operations on the demand and sales of the products. However, the optimisation studies focus on the appropriate model development and optimisation techniques.

2.2.2.1 Experimental studies

Due to the scarcity of space within stores, several researchers have concentrated on studying the relationship between the space allocated to an item and the sales of that item. Most have reached a common conclusion that a weak link exists between them and the significance depends on the types of items (Kotzan and Evanson, 1969; Cox, 1970; Curhan, 1972; Dreze et al., 1994; Desmet and Renaudin, 1998; Yang and Chen, 1999).

In 1969, Kotzan and Evanson (Kotzan and Evanson, 1969) began to investigate the relationship between the shelf space allocated to an item and the sales of that item and found that a significant relationship existed within the three tested drug stores. Cox's research (Cox, 1970) experimented with the shelf facings for two brands of two categories, salt and coffee cream. He found that the influence of shelf facings on sales was very weak and dependent on the category of products. However, his experimental results may be affected by the limited experimental samples. Curhan (Curhan, 1972) defined space elasticity as "*the ratio of relative change in unit sales* *to relative change in shelf space*" and reported an average value of 0.212. However, this is just an average value. The value of the space elasticity can be very different, depending on the products, stores and in-store layout (Curhan, 1973).

Dreze et al. (Dreze et al., 1994) carried out a series of experiments to evaluate the effectiveness of shelf space management and cross-category merchandise reorganisation. The experiments were carried out within sixty stores of a leading supermarket chain in Chicago, USA, of which eight categories were chosen for the experiments. The shelf space manipulation included changing product facings, deletion of slow moving items, changes of shelf height, etc. Cross-category merchandise reorganisation included manipulations to enhance complementary shopping by placing naturally complementary products together. The results showed that, compared with the number of facings assigned to a brand, location had a larger impact as long as a minimum inventory (to avoid out-of-stocks) was guaranteed. Complementary merchandising also experienced a positive boost in sales (above 5%) on the tested products (toothbrush, toothpaste and laundry care).

On the contrary, more recent research (Desmet and Renaudin, 1998) showed that direct space elasticities were significantly non-zero and varied considerably across different categories. Costume jewellery, fruit and vegetables, underwear, and shoes were among the highest space elasticities while textiles, kitchen and do-it-yourself products had low values.

If the products are always available and the consumers would never switch to another brand, the change of space allocated to an item would have no effect on its sales (Borin et al., 1994). However, in fact, nearly half of the consumers would switch to other stores or change their previous choice to an alternative brand if their first choice is out-of-stock (Verbeke et al., 1998). On the other hand, the purchase of one merchandise could increase the possibility of buying another with complementary functions (for example, a customer who bought a toothbrush may also buy toothpaste). Cross elasticities were introduced to evaluate the interdependence between two different items in Corstjens and Doyle's model (Corstjens and Doyle, 1981a). The values of cross elasticities were assumed to be within the range of [-1, 1]. It was positive if two items were complementary and negative if they could be substituted for each other. This effort was echoed in Borin et al. (Borin et al., 1994) and Urban (Urban, 1998), both of which employed cross elasticities in their models. Although cross elasticities are helpful in revealing the relationships between different items, it is quite difficult to obtain a reliable estimation of so many values ($n \times n$ for *n* items) due to the complicated merchandise relationships. Therefore, recent researchers have disregarded it in their models (Desmet and Renaudin, 1998; Urban, 2002).

Display location is another factor that has been studied. Apart from positive experimental results from (Dreze et al., 1994), several other publications emphasised the importance of location as a factor in improving sales (Buttle, 1984; Hart and Davies, 1996). Campo et al. (Campo et al., 2000) investigated the impact of location factors on the attractiveness of product categories and stated that the sales of the whole store were dependent on the intrinsic attractiveness based on category, store and trading area characteristics as well as cross elasticities between the categories. However, the model did not consider the difference in visibility or prominence between various locations in a store.

2.2.2.2 Shelf space allocation models and optimisation methods

Several space allocation models have been proposed in the literature. Most of them have formulated the demand rate of an item as a function of the space allocated to the item, of which a classic model appears as a polynomial form proposed by Baker and Urban (Baker and Urban, 1988):

$$D(x) = \alpha x^{\beta} \qquad \alpha > 0, \quad 0 < \beta < 1 \tag{2-1}$$

where $D(x)$ is the demand rate of the product, x is the number of facings or the displayed inventory. α is a scale parameter and β is the space elasticity of the product. The advantageous characteristics of this model include the *diminishing returns* (the increase in the demand rate decreased as the space allocated to this shelf increased), *inventory-level elasticity* (the space elasticity parameter represents the sensitivity of the demand rate to the changes of the shelf space), *intrinsic linearity* (the model can be easily transformed to a linear function by a logarithmic transformation and the parameters can then be estimated by a simple linear regression) and its *richness*.

Corstjens and Doyle (Corstjens and Doyle, 1981a) firstly formulated their model as a non-linear multiplicative form and incorporated the cross elasticities, a set of problem parameters that reflect the interrelationships between different products under consideration. The inventory and handling cost effects were also considered. Based on this model, some non-space factors were also taken into account in (Zufryden, 1986), such as price, advertising, promotion, store characteristics, etc. A dynamic programming approach was proposed to solve this model. However, this approach may only be suitable for small sized problems. The approach becomes computationally expensive for large problem instances.

Some integrated models have also been proposed based on the correlation of retailing decision processes (Borin et al., 1994; Urban, 1998; Hwang et al., 2005). Borin et al. (Borin et al., 1994) developed an integrated model whose objective is to maximise the category return on inventory. This model was supposed to help a retailer to decide which products to stock (*product assortment*) and how much space should be allocated to them. The demand function was formulated into three components: unmodified demand, modified demand and acquired demand. Unmodified demand represented the customers' direct preference for an item and was calculated according to its market share. Modified demand took account of the interdependence and substitution of different merchandise. Acquired demand represented the indirect demand captured from those products which were excluded from the assortment. The authors also considered the model's sensitivity analysis with regards to the different degree of parameter errors which may be introduced during their estimations (Borin and Farris, 1995). A heuristic procedure, based on simulated annealing, was employed to optimise the model. The neighbourhood was defined by swapping one facing of two random items. The results showed that simulated annealing was more efficient and flexible compared with the shelf allocation rule based on the share of sales (a common space allocation rule).

The above-mentioned models used the number of facings of an item to predict the demand quantity of that item. However, the effect of partially-stocked items (some facings are missing) was not explicitly reflected. Urban (Urban, 1998) replaced the number of allocated facings with average on-shelf inventory. His model also integrated an existing inventory-control model, a product assortment model and a shelf-space allocation model. A greedy heuristic and a genetic algorithm (GA) were proposed to solve the problem. A GA chromosome represented a given product assortment vector (i.e. "0": excluded, "1": included). The violations of some constraints were allowed in the initial solutions and then repaired by a heuristic procedure. However, the GA operations (crossover and mutation) were only applied to product assortment variables, not to space allocation variables. For this reason, the solution obtained by this approach is normally locally optimal.

Recently, Hwang et al. (Hwang et al., 2005) proposed an interesting integrated shelf space allocation and inventory control model. One characteristic of this model is the inclusion of vertical shelf location effects in the demand function. A gradient heuristic search and a genetic algorithm were proposed to optimise the model. Unfortunately, an underlying mathematical derivation (called "property 1" in the publication) is only applicable to continuous variables. The derivation cannot be extended to discrete variables. However, as mentioned in section 2.2, for the shelf space allocation and inventory control, one of the hard constraints is the integrality of decision variables. Hence other derivations in the paper based on "property 1" suffer from this drawback and are not correct in this sense.

One drawback of the above models is that they have many parameters and it is difficult to put those models into practice because of the difficulty in obtaining a reliable estimation of them. In fact, Yang (Yang, 2001) argued that: *"for commercial models, a very important criterion for selecting a space allocation method is the simplicity and ease of operation of the method"*. He proposed a simpler linear model based on the work of Corstjens and Doyle (Corstjens and Doyle, 1981a), by assuming that a product total net profit was linearly proportional to the number of facings allocated to that product. This is, however, unrealistic for the real-world retail environment and also contradictory to the experimental results from the literature which generally suggested a relatively small space elasticity value (Dreze et al., 1994). A greedy algorithm, in conjunction with three simple heuristics, was proposed to optimise the model. However, only several numerical examples were used to justify the algorithm and they are far from the real-world shelf space allocation problems which are usually much larger and more complicated. In addition, the three heuristics rejected all "bad moves" (a decrease in the objective value for a maximisation problem). The algorithm, in fact, worked in a random greedy fashion and could easily become trapped in a local optimum. Recently, Lim et al. (Lim et al., 2004) experimented with network flow, tabu search and a modified squeaky-wheel optimisation algorithm to this linear shelf space allocation model and was able to produce better results. Among the algorithms they experimented with, the modified squeaky-wheel optimisation problem outperformed others across the problem instances.

Due to the diversity of products' properties and business styles, there can be many different shelf space allocation problems. Therefore, it is difficult to develop a generic model that can represent all real-world shelf space allocation problems. For research purposes, chapter 5 will consider a general problem that has previously been the subject of most of the academic research on planograms. In chapter 6 we will address a shelf space allocation problem specifically for fresh produce. Two practical models are proposed for these two types of shelf space allocation problems. In the next section, some other related shelf space allocation problems are briefly reviewed.

2.3 Other Space Allocation Problems

2.3.1 Bin packing problem

The one dimensional bin packing problem is defined as follows. Given a set of items $I = \{1, ..., n\}$ each having an associated size or weight w_i and a set of bins with identical capacities c. The problem is to pack all the items into as few bins as possible, without exceeding the capacity of the bins. The bin packing problem is a well-known NP-Hard combinatorial optimisation problem (Martello and Toth, 1990a) and there is no known polynomial time-bounded algorithm that can solve every problem instance to optimality. However, it is not difficult to get a lower bound of the problem. A straightforward lower bound can be obtained by $L_1 = \left\lceil \sum_{i=1}^{n} w_i / c \right\rceil$ where $\lceil x \rceil$ is the smallest integer not less than *x*. Some stronger lower bounds were studied in (Martello and Toth, 1990b; Scholl et al., 1997). This problem can also be extended to two-dimensional and three-dimensional bin packing, where both the bin and the items have sizes (other aspects of the problem) in two or three dimensions. One-dimensional bin packing problems have been addressed by many researchers and both exact methods and meta-heuristic methods have been developed. See chapter 4 for a detailed review.

2.3.2 Knapsack problem

The knapsack problem has been intensively studied (Martello and Toth, 1990a) and there are several variations of the problem, of which 0-1 is the most commonly studied.

0-1 knapsack problem

The 0-1 knapsack problem can be described as follows. Given a knapsack with capacity *c* and a set of *n* items, each item *i* is associated a profit p_i and a weight w_i. The problem is to select a subset of items such that the total profits *z* of the selected items are maximised. The mathematical model was formulated in (Martello and Toth, 1990a) as follows:

$$\max \quad z = \sum_{i=1}^{n} p_i x_i \tag{2-2}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \le c \tag{2-3}$$

$$x_i = 0 \text{ or } 1, \quad i = 1, \ldots, n \tag{2-4}$$

The 0-1 knapsack problem can be exactly solved by a branch and bound algorithm (Martello and Toth, 1975) and dynamic programming (Toth, 1980). However, with very large problem instances (n>2000), approximation approaches are proposed due to the significant computational requirements of these exact approaches. Sahni proposed (Sahni, 1976) the first pseudo-polynomial approximation method with the prefixed worst-case performance. A fully polynomial-time approximation scheme was given by Ibarra and Kim (Ibarra and Kim, 1975) based on a dynamic programming algorithm.

Bounded knapsack problem

In the 0-1 knapsack problem, the variable x_i takes either 0 or 1. The problem can be extended by allowing the variable x_i to have several values bounded by a given range. The problem is formulated as follows in (Martello and Toth, 1990a):

n = number of items;

p_i = profit of item i;

w_i =weight of item i;

b_i = upper bound on the availability of item i;

c = capacity of the knapsack;

x_i = the number of item i being selected in the knapsack.

$$\max \quad \sum_{i=1}^{n} p_i x_i \tag{2-5}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \le c \tag{2-6}$$

$$0 \le x_i \le b_i \text{ and } x_i \in \mathrm{Z}^+, \quad i = 1,\ldots,n \tag{2-7}$$

Similarly, the bounded knapsack problem can be solved by dynamic programming and branch-and-bound approaches. However, it has been shown that the bounded knapsack problem can be more efficiently solved by transforming it into a 0-1 knapsack problem and solving the transformed problem by the approaches for the 0-1 knapsack problem (Martello and Toth, 1990a).

When $b_i \rightarrow +\infty$, the bounded knapsack problem degenerates into an *unbounded knapsack problem*. Still, dynamic programming and a branch-and-bound algorithm can solve the problem efficiently by transforming it into a 0-1 knapsack problem. However, it was proven to be not as efficient as when solving it directly (Martello and Toth, 1990a).

0-1 multiple knapsack problem

Another generalised 0-1 multiple knapsack problem is the 0-1 multiple knapsack problem, where the problem has a set of knapsacks rather than one. The problem is formulated as follows in (Martello and Toth, 1990a):

m = the number of knapsacks;

n = the number of items;

p_i = profit of item i;

w_i =weight of item i;

c_j = capacity of knapsack j;

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is assigned to knapsack } j; \\ 0 & \text{otherwise.} \end{cases}$$

$$\max \quad \sum\nolimits_{j=1}^{m} \sum\nolimits_{i=1}^{n} p_i x_{ij} \tag{2-8}$$

$$\text{subject to} \quad \sum\nolimits_{i=1}^{n} w_i x_{ij} \le c_j, \quad j = 1,...,n \tag{2-9}$$

$$\sum\nolimits_{j=1}^{m} x_{ij} \le 1, \quad i = 1,...,n \tag{2-10}$$

$$x_{ij} = 0 \text{ or } 1, \quad i = 1,...,n, \ j = 1,...,m \tag{2-11}$$

A straightforward upper bound of the model can be obtained by solving a relaxed 0-1 knapsack problem with a single knapsack of capacity $\sum_{j=1}^{m} c_j$. Branch-and-bound approaches are usually used to exactly solve the problem while dynamic programming was proven to be impractical because the multiple knapsack problem is NP-Hard in a strong sense (Martello and Toth, 1990a).

2.3.3 Generalised assignment problem

The generalised assignment problem is similar to the multiple knapsack problem except that the profit and weight of each item vary with respect to the containers assigned to it. The model is formulated as:

m = the number of containers;

n = the number of items;

p_{ij} = profit of item i if assigned to container j;

w_{ij} =weight of item i if assigned to container j;

c_j = capacity of container j

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is assigned to container } j; \\ 0 & \text{otherwise.} \end{cases}$$

$$\max \quad \sum_{j=1}^{m} \sum_{i=1}^{n} p_{ij} x_{ij} \tag{2-12}$$

$$\text{subject to } \sum_{i=1}^{n} w_{ij} x_{ij} \le c_j, \quad j = 1, \dots, m \tag{2-13}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, \dots, n \tag{2-14}$$

$$x_{ij} = 0 \text{ or } 1, \quad i = 1, \dots, n,\ j = 1, \dots, m \tag{2-15}$$

A practical application of the model is assigning n tasks to m processors (or n jobs to m machines) given the profit p_{ij} and the level of resource required w_{ij} for the assignment of task i to processor j and total resource c_j available for each processor j.

Note that all the problems discussed above are NP-Hard (Martello and Toth, 1990a).

2.4 Summary

The fierce competition that exists in the retailing industry compels retailers to adopt sophisticated systems to automate and optimise their decision making processes. The shelf space allocation problem is one of the key factors that can affect a retail company's financial performance. However, current software does not provide an optimised shelf space allocation decision and they require significant human interaction. This research aims to investigate the methodologies and algorithms that can be used in the next generation of planogram software, which will allow a user to produce automated, optimised planograms.

This chapter has placed the work in context. Several important issues with regards to the shelf space allocation problem have been discussed. Due to the different product proprieties, the shelf space allocation problem can be very different.

This chapter has reviewed previous research for shelf space allocation both on the experimental studies and optimisation model studies. The experimental studies have consistently shown the positive effect of shelf space allocation on the demand of the product. This effect is largely attributed to the consumers' unplanned purchases due to the improved visibility and appearance of products. Improvement of a retailer's shelf space allocation could dramatically increase its financial performance. The optimisation model studies have focused on the modelling and optimisation of shelf space allocation problems. Most of the models employed a non-decreasing polynomial function to formulate the relationship between the shelf space allocated to a product and the demand for that product. However, with the increase in shelf space, the rate of increase in demand diminishes. To measure the effect that shelf space has on the product demand, a parameter, space elasticity, was introduced. Space elasticity usually takes a value in the range of [0, 1]. A larger value of space elasticity means a larger influence on the product demand from the shelf space. Some researchers have also used cross elasticities to describe the relationship between two different products. However, recently researchers have argued that the inclusion of cross elasticity is impractical for real-world applications due to the increased complexity of the problem and the difficulty in obtaining a reliable estimation of these parameters.

Due to the NP-Hard nature of the shelf space allocation problem, it is impractical to work out a polynomial time bounded solution procedure that can solve every problem instance to optimality. Dynamic programming was firstly proposed to optimise the shelf space allocation model. However, this method may require extremely high computational times for large problem instances. As alternatives, heuristic and metaheuristic methods have also been used to solve the problem, such as simulated annealing, genetic algorithms and tabu search. This thesis shall focus on the heuristic and metaheuristic approaches, especially newly emerging metaheuristic search technologies. These new approaches generally broaden the search by making use of several neighbourhood structures or several heuristics to explore the neighbourhoods and have been reported to be superior to the conventional local search approaches that only use one single neighbourhood structure. It is assumed that these techniques are also promising for shelf space allocation problems.

In this chapter, several other space allocation problems have also been briefly reviewed, including bin packing, knapsack and generalised assignment problems. These problems are closely related to shelf space allocation. It is hoped that the study of these problems may be helpful in guiding us to choose appropriate search techniques for the optimisation of shelf space allocation problems in general.

The next chapter presents an overview and discusses the latest meta-heuristic techniques which may be promising for the optimisation of the problem that we are concerned with.

CHAPTER 3. OPTIMISATION TECHNIQUES: AN OVERVIEW

3.1 Introduction

As mentioned in chapter 2, shelf space allocation problems are related to the bin packing and knapsack problems, which are NP-Hard. There is no known polynomial-time bounded algorithm that can solve every instance to optimality. This chapter introduces several important concepts with regard to problem complexity. Some well-known optimisation technologies which have been successful for NP-Hard combinatorial optimisation problems are then reviewed and promising techniques are highlighted.

3.2 NP-Completeness and NP-Hardness

Combinatorial problems refer to the class of problems with discrete variables (Reeves, 1995) that arise in many areas and consist of a large subset of problems, such as resource allocation, planning, scheduling, routing, decision making, etc. The computational complexity of combinatorial problems is generally high, especially for those problems with a large solution space.

3.2.1 Algorithm complexity

Algorithm complexity is measured in terms of *time complexity* and *space complexity*. The time complexity of an algorithm is a measure of the amount of time required to execute an algorithm for a given number of inputs (also conveniently expressed as problem “size”). It is measured by its rate of growth relative to standard functions. The normal standard functions include *constant*, *logarithmic*, *polynomial* and *exponential*. The space complexity of an algorithm is a measure of how much storage is required by the algorithm. Typically, computer scientists are interested in minimising the time complexity of algorithms because computer memory costs have decreased dramatically over the past 25 years. Therefore, this section only discusses time complexity.

3.2.2 P and NP

Complexity theory mainly focuses on decision problems whose solutions are either "yes" or "no". However, because many optimisation problems have their counterparts of decision problems, complexity theory is still useful for general optimisation problems (Garey and Johnson, 1979). In many cases, a problem can be solved by several algorithms and each algorithm may have different time complexities. However, problem complexity is measured by the time complexity of the most "efficient" algorithm for the problem (Garey and Johnson, 1979). A problem is said to be *tractable* if there is an algorithm that can solve the problem in polynomial time. If no algorithm can solve the problem in polynomial time, the problem is said to be *intractable*. In this case, either the problem is *undecidable* (the problem is not solvable by any algorithm) or solving it requires exponential computational time.

The problems are usually classified under two distinct headings: P and NP. P (standing for *polynomial*) represents the class of the problems that are solvable by a *deterministic* algorithm with polynomial time complexity. NP is the class of the problems that can be solved in polynomial time by a *nondeterministic* algorithm (NP stands for nondeterministic polynomial). A nondeterministic algorithm is composed of two stages. The first stage of the algorithm simply *guesses* a structure *S* of the problem instance *I*, which are input into the second stage to *check* whether the structure *S* is a solution of the instance *I* or not. Note that the second stage will use a deterministic algorithm bounded by polynomial computation time (Garey and

Johnson, 1979). In this sense, NP contains the class of the problems for which a solution can be verified efficiently (in polynomial time) but where it is not known how the given solution is obtained. It is not difficult to understand that $\mathrm{P} \subseteq \mathrm{NP}$. However, because there is no known polynomial time algorithm for many problems (the travelling salesman problem, for instance) in NP, most researchers have a strong belief that $\mathrm{P} \neq \mathrm{NP}$. However there is, as yet, no theoretical proof.

3.2.3 NP-Completeness and NP-Hard

If $\mathrm{P} \neq \mathrm{NP}$, there are some problems which do not belong to P and hence are intractable. These problems are considered to be hard because tackling them requires exponential computation time. Cook (Cook, 1971) firstly identified a class of hard problems in NP based on the concept of the *satisfiability* problem. The satisfiability problems are defined as the problems to which every other problem in NP can be *reduced* by a polynomial time bounded transformation. These satisfiability problems consist of what we now call *NP-Complete* problems. NP-Complete problems are considered to be the hardest problems in NP because if satisfiability problems can be solved efficiently by a polynomial algorithm, every problem in NP can then be solved in polynomial time by reducing it to a satisfiability problem. However, it is generally assumed that finding a polynomial time algorithm for the problems in NP-Complete is unlikely.

However, sometimes there are problems which cannot be proved to belong to NP (i.e. there is no obvious polynomial time procedure for verification of a solution) but one can show that they are at least as difficult as the NP-Complete ones, even though they were not proved to be intractable. These problems are commonly labelled *NP-Hard*, meaning “at least as hard as any problem in NP”. An example of such a problem is where the problem of verifying a solution itself is an NP-Complete one.

Note that the definition of NP-completeness only considers decision problems. However, its counterparts of the optimisation problem are at least as hard as the former because verifying an optimal solution is not obvious in the latter (Falkenauer, 1998).

3.3 Review of Optimisation Approaches

3.3.1 Introduction

When solving an optimisation problem, one should seek *exact methods* to find the optimal solution to the problem. However, as discussed in section 3.2, some problems (NP-Complete problems, for instance) are very hard, such that the algorithm to find an optimal solution has an exponential time complexity. It is highly computationally expensive when dealing with large size problems (sometimes even for medium size problems). In such circumstances one may refer to some approximation approaches which can solve the problems with satisfactory solution quality within reasonable computational time. *Heuristic* and *metaheuristic* approaches are usually proposed to achieve this objective.

3.3.2 Exact methods

An exact method seeks to solve the problem to optimality. Well-known exact methods include linear programming, dynamic programming, branch and bound, and Lagrangian relaxation method. Although these approaches could obtain optimal solutions, it can be computationally expensive and impractical for many real-world applications.

3.3.2.1 Linear programming

Linear programming (LP) is ranked as the most important scientific advance in operational research (Hillier and Lieberman, 2005). It was developed as a discipline in the 1940's, motivated initially by the need to solve complex planning problems in wartime operations. Its development accelerated rapidly in the post-war period as many industries found valuable uses for linear programming. The most common type of application involves the general problem of allocating limited resources among competing activities in the best possible (i.e., optimal) way. Advances in the research of linear programming were mainly attributed to George B. Dantzig (Dantzig, 1951; Dantzig, 1963), who devised the *simplex method* that can efficiently solve a linear programming problem to optimality. A detailed description of the method can be found in (Nemhauser and Wolsey, 1988). One limitation of linear programming is that all mathematical functions (including the objective function and the constraint function(s)) in the model are required to be linear. Another is that linear programming cannot handle discrete variables.

3.3.2.2 Dynamic programming

The term dynamic programming (DP) was introduced by Richard Bellman (Bellman, 1957) who pioneered the theory and application of dynamic programming. Dynamic programming was originally proposed to solve sequential decision making problems but was later extended to solve many other combinatorial problems that can be decomposed into a nested family of sub-problems. The problems can hence be tackled by a recursive procedure, in which each iteration (or recursive call) corresponds to a sub-problem. Compared with linear programming, dynamic programming is a more general approach to problem solving. Dynamic programming can handle discrete variables and nonlinear models. However, the application of dynamic programming requires that the problem objective function is only dependent on problem's current state and its current decisions. Generally, dynamic programming is only suitable for small and moderately sized problems. The computation time may increase dramatically with an increase in the problem size, due to the recursive structure of the algorithm. See (Hillier and Lieberman, 2005) for a detailed discussion on the theory and application of this approach.

3.3.2.3 Branch-and-bound

The Branch-and-bound search technique is a reasonably efficient approach for solving integer programming (IP) and mixed integer programming problems (MIP). The basic idea behind branch-and-bound is *divide and conquer*, which means solving difficult problems by recursively dividing them into smaller and smaller sub-problems until those sub-problems can be solved. There are several versions of branch-and-bound algorithms but all of them can be divided into three stages: *branch*, *bound* and *fathom*. The branching corresponds to partitioning the entire set of feasible solutions into smaller and smaller subsets by fixing an integer variable's value (or its range if the integer variable could take many values) at each iteration. The bounds (upper or lower) of these subsets are then calculated in the bound phase using a relaxation method, such as LP relaxation or Lagrangian relaxation. In the third stage, the algorithm then acquires the solution space by discarding the subsets which are unlikely to contain the optimal solution based on the information of their bounds. Note that branch-and-bound is different from the complete enumeration method. The algorithm only searches the part of the solution space which could contain the optimal solution. See (Nemhauser and Wolsey, 1988; Hillier and Lieberman, 2005) for a detailed description of the algorithm.

3.3.2.4 Lagrangian relaxation

The Lagrangian relaxation method is a very useful tool in obtaining lower (or upper) bounds for combinatorial optimisation problems (Reeves, 1995; Hillier and Lieberman, 2005). This is done by relaxing some difficult constraints and adding them into the objective function such that the relaxed problem can be exactly solved to optimality, which is considered as the lower (or upper) bound of the original problem. In Lagrangian relaxation, the key issue is to decide which constraint(s) to relax and how to calculate the optimal multiplier factor.

3.3.2.5 Generalised reduced gradient algorithm (GRG)

The concept of the generalised reduced gradient algorithm was firstly used by Abadie and Carpentier (Abadie and Carpentier, 1969) and the underlying ideas of the algorithm were also described in (Gabriele and Ragsdell, 1977; Lasdon et al., 1978). Here we only give a brief description.

GRG is one of the reduced-gradient methods that are able to solve differentiable non-linear programming problems (both in terms of objective function and constraints) of the form:

$$\begin{aligned} &\text{maximise} && y(X) && && \text{(3-1)} \\ &\text{subject to:} && f_j(X) \le 0 && j = 1, \ldots l \\ & && g_k(X) = 0 && k = l+1, \ldots l+m \\ & && lb_i \le x_i \le ub_i && i = 1, \ldots n \end{aligned}$$

where $X = \{x_1, \ldots, x_n\}$ is a vector containing n natural variables or independent variables and $n>m+l$. To solve the problem, the model is firstly transformed into a model with only equality constraints by adding l non-negative *slack* variables $x_{n+1}, \ldots, x_{n+l}$. We have

$$\begin{aligned} &\text{maximise} && y(X) && && (3\text{-}2)\\ &\text{subject to:} && g_k(X')=0 && k=1,\ldots m+l &&\\ & && lb_i \le x_i \le ub_i && i=1,\ldots n+l && \end{aligned}$$

where $X'=\{x_1,\ldots,x_n,x_{n+1},\ldots x_{n+l}\}$.

The idea of the GRG method is to convert the constrained problem into an unconstrained one. For nonlinear constraints, the first order Taylor expansion is firstly applied to convert them to linear constraints. Then, variables are divided into *basic* ones and *nonbasic* ones. GRG uses (*m*+*l*) equality constraints to solve (*m*+*l*) *nature* variables, called basic variables, in terms of the remaining (*n*-*m*) non-basic variables. This will reduce the number of independent variables to (*n*-*m*). A search direction is then decided by the generalised reduced gradient in terms of every variable. To find the local optimality of the objective along this search direction, any one-dimensional search method can be used, such as the Newton's method and the quadratic interpolation method (Gabriele and Ragsdell, 1977).

3.3.3 Heuristics and metaheuristics

In the dictionary (Oxford Dictionary of Computing, 1997), *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 most of the time but their results or performance cannot be guaranteed.*

Reeves (Reeves, 1995) defined 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".

A heuristic could be used to create a solution (also called a constructive heuristic) or to improve an existing solution by exploring the neighbouring solutions based on certain rules or strategies. In this context, *greedy algorithm* and *hill climbing* are examples of heuristics. A greedy algorithm is a constructive heuristic which seeks the biggest reward (or the least penalty for a minimisation problem) at any point when building a solution. However, a hill climbing method starts from an initial solution and keeps moving to better neighbouring solutions until a stopping criterion is met. One problem with these simple heuristic methods is that they are prone to getting stuck in a local optimum.

To prevent these simple heuristic methods from getting trapped at local optima, many advanced heuristic approaches, called *meta-heuristics*, have been developed (Osman and Kelly, 1996; Voss et al., 1999; Glover and Kochenberger, 2003). Voss et al. (Voss et al., 1999) defined meta-heuristics as

"*an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method.*"

In (Glover and Kochenberger, 2003), 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 means 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 in the spotlight 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. A clear-cut classification of meta-heuristics is difficult because some approaches are actually general frameworks and usually hybridised with other (meta-)heuristic methods. However, there are several key components characterising them. One commonly used classification distinguishes between *single-point* and *population-based* (Blum and Roli, 2003). The former refers to 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, iterative local search, guided local search, variable neighbourhood search and greedy randomised adaptive search procedure while genetic algorithms, evolutionary strategies, ant colony optimisation, and scatter search can be regarded as population-based methods. However, this classification does not embrace some hybrid methods. For example, although GRASP is regarded as a single-point approach, some GRASP approaches hybridise the technique *path-relinking* which requires maintaining a population of high quality solutions and therefore also belongs to the population-based methods.

Some researchers (Taillard et al., 2001) also draw distinctions between the methods that make use of *memory* and *memory-less* methods. Tabu search is a typical meta-heuristic approach utilising the search history (memory). Usually a short-term memory is maintained to prevent the cycling of the search while the long-term memory is used to balance the intensification and diversification strategies. Other memory based methods include iterative local search, guided local search, ant colony optimisation, etc. Simulated annealing and greedy adaptive randomised search procedures are typical memory-less meta-heuristic approaches.

Some other classifications include *single neighbourhood* vs *various neighbourhood, static* vs *dynamic objective function* and *nature-inspired* vs *non-nature* inspiration. One can refer to (Birrattari et al., 2001) for further discussions.

In the following subsections, we shall sequentially overview some popular heuristic and meta-heuristic methods that have been widely used in many applications.

3.3.3.1 Constructive (meta)heuristics

Constructive heuristics *"build solutions to a problem under consideration in an incremental way starting with an empty initial solution and iteratively adding appropriate solution components without backtracking until a complete solution is obtained"* (Dorigo and Stutzle, 2003). Constructive heuristics are usually used as an initial solution builder for many local search approaches. To generate a high quality initial solution, the key is to choose which components to add to the solution at each iteration (Burke and Kendall, 2005). If the selection is carried out in a random way, the solutions returned by the constructive heuristics correspond to random solutions. The quality of these random solutions is normally very poor. The most common way is to refer to a function or a heuristic rule (for example, first-fit descent for a bin packing problem) for the next solution component selection. Other more complex constructive methods are also used where the constructive heuristics make use of several functions or heuristics using certain learning mechanisms (Dorigo and Maniezzo, 1996; Petrovic and Qu, 2002; Burke et al., 2005).

3.3.3.2 Simple local search

The search space consists of all solutions that satisfy the given hard constraints (soft constraints might be violated, but at the cost of solution quality). The size of the search space may be dependent on the problem size as well as the solution representation. Local search methods often use the concept of a *neighbourhood* which defines the set of solutions that can be reached from the current solution by a single step operation (or move) (Osman and Laporte, 1996). Starting from an initial solution, which can be generated randomly or by a constructive heuristic, the simple local search iteratively samples a candidate solution in the neighbourhood of the current solution. The candidate solution is accepted as the current solution if, and only if, it is better than the current solution. For most constrained combinatorial problems with a *rugged* search space, the simple local search approach is prone to getting stuck at a local optimum. This simple local search is also called *hill-climbing* for a maximisation problem or *descent* method for a minimisation problem.

3.3.3.3 Hyper-heuristics

Meta-heuristics have been intensively investigated and applied to a wide variety of applications in the last twenty years, including scheduling, production planning, resource assignment, supply chain management, decision support systems and bio-informatics (Reeves, 1995; Osman and Kelly, 1996; Glover and Laguna, 1997; Glover and Kochenberger, 2003; Burke and Kendall, 2005). However, many of these state-of-art algorithms are too problem-specific. Once the problem is changed (even slightly), the performance of the already developed specific-tailored meta-heuristic may decrease dramatically for the new problem. Significant parameter tuning may also be necessary for the purpose of adapting the algorithms to the new problem or a new problem instance. The "No-Free-Lunch" theorem (Wolpert and MacReady, 1997) states that there is no one algorithm that is superior to any other algorithm across all classes of problems. If an algorithm outperforms other algorithms on a specific class of problems, there must be another class of problems for which this algorithm is worse than the others. This drawback of meta-heuristics has motivated researchers to design algorithms which can be applied in many different situations, although recognising that the "No-Free-Lunch" theorem means we can never produce a fully generic algorithm.

Hyper-heuristic (Burke et al., 2003a; Ross, 2005) is a recently used term to describe algorithms which aim to raise the generality of the algorithms. The idea behind one type of hyper-heuristic is that each problem-specific heuristic may have some *weakness* in certain scenarios in which other heuristics may perform better. Better algorithmic performance could be achieved by combining a set of heuristics, instead of using just a single heuristic alone. Hyper-heuristics combine a set of easily implemented, problem-specific heuristics in a strategic way such that the algorithm is able to tackle not only a specific problem or problem instance but a batch of problems. Hyper-heuristics are defined as a procedure of "*using (meta-)heuristics to choose (meta-)heuristics to solve the problem in hand*" (Burke et al., 2003a). Unlike most meta-heuristics, which search the solution space directly, hyper-heuristics work on the problem indirectly by strategically calling appropriate heuristics at different times in the search. The hyper-heuristic normally lacks problem-specific knowledge although non-problem knowledge could pass into and out of the hyper-heuristic black box. A general framework was illustrated in (Burke et al., 2003a) and is shown in figure 3-1.

The algorithm is divided into two layers, a hyper-heuristic black box and a problem-specific layer, with a problem domain barrier separating them. The problem-specific layer includes a set of low level heuristics, which are different rules or strategies to transform the state of the current solution. Note that although low level heuristics could be meta-heuristics, they are usually simple and easily implemented heuristics. The hyper-heuristic black box usually only has access to general non-problem specific knowledge, such as the difference in the objective function, historical performance of each heuristic, solution states, etc.

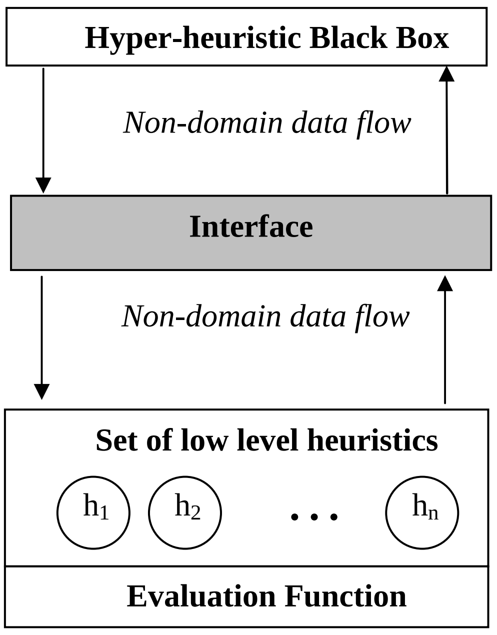


Figure 3-1: An example of a hyper-heuristic framework
Source: (Burke et al., 2003a)

This hyper-heuristic framework does not aim to beat other state-of-art problem-specific approaches, but to provide a generalised approach for many problems with solutions that are "*good enough, soon enough and cheap enough*". Meanwhile, hyper-heuristics do not aim to challenge the "No-Free-Lunch Theorem" but only tries to raise the generality of the algorithms as far as possible.

Soubeiga (Soubeiga, 2003) categorised hyper-heuristics into two types, learning based hyper-heuristic and non-learning based hyper-heuristics. Non-learning based hyper-heuristics included approaches which make use of several neighbourhood structures and heuristics but the choice of which neighbourhood or heuristic to call is in a predefined sequence. According to his classification, variable neighbourhood search (VNS) (Hansen and Maldenovic, 2001) was classified as this type of hyper-heuristic. Learning based hyper-heuristics refers to those approaches that dynamically change the preference of each neighbourhood or heuristic based on their historical performance guided by some learning mechanisms.

Hyper-heuristics can also be divided into constructive hyper-heuristics and local search hyper-heuristics. Constructive hyper-heuristics construct a solution from "scratch" by calling from a set of constructive heuristics (as opposed to the general greedy heuristic which uses only a single heuristic). However, the local search hyper-heuristics start from a complete initial solution and repeatedly select appropriate heuristics to lead the search in a promising direction. A constructive hyper-heuristic searches for a good sequence of heuristics (or a solution strategy) which can build a solution. A local search hyper-heuristic tries to select the "right" heuristic to guide the search in the promising direction.

It should be noted that hyper-heuristics are not new approaches. Their application can be traced back to the 1960's (Fisher and Thompson, 1961) although the term "hyper-heuristic" was not used at that time. Work was also carried out through the 80's and 90's (O'Grady and Harrison, 1985; Mockus, 1989; Kitano, 1990; Hart et al., 1998).

Soubeiga (Soubeiga, 2003) carried out a survey of the research and applications that have been carried out in the past using the ideas of hyper-heuristics. The following sections give an updated review of hyper-heuristics, which are separated into two different parts: constructive hyper-heuristics and local search hyper-heuristics.

Constructive hyper-heuristics

In constructive hyper-heuristics, the low-level heuristics are usually well-known constructive heuristics, for example, First-Fit Descent (FFD) and Best-Fit Descent (BFD) for bin packing problems (Martello and Toth, 1990a), Largest Degree Descent (LDD) and Saturation Degree Descent (SDD) (Carter, 1986; Carter and Laporte, 1996; Burke and Causemacker, 2003) for exam timetabling problems. Running these simple heuristics alone can create a solution efficiently. However, in many cases, they get trapped into local optima and produce poor quality solutions. Constructive hyper-heuristics could synchronise these simple heuristics and, at each decision point, choose the most appropriate heuristic to obtain good quality solutions.

Fisher and Thompson (Fisher and Thompson, 1961) are probably the first researchers to use the idea of a hyper-heuristic when studying a job-shop scheduling problem. In their experiments, two types of high-level strategies were used to combine two simple job-shop scheduling constructive heuristics (rules). The first strategy was an *unbiased random process*, which randomly selected an available rule to make a scheduling decision at each decision point. The second strategy used a probabilistic learning mechanism to guide the selection of heuristics. In this strategy, the probability with which a heuristic was selected was updated dynamically based on a reward-punishment procedure, similar to the idea of reinforcement learning (Kaelbling et al., 1996; Sutton and Barto, 1998). That is, the probability of selecting a heuristic increased if the heuristic improved the solution and decreased otherwise. The experimental results showed that the hyper-heuristic with a learning mechanism was shown to be superior to the unbiased random process and even the unbiased random rules combination produced much better results than any of them run separately.

Several genetic algorithm (GA) based constructive hyper-heuristics have been developed, although early research usually termed them as *indirect GAs*. In those approaches, a GA's chromosome represents a sequence of heuristics or rules by which a solution can be built. In this case, the genetic algorithm does not search in the solution space. It is used to evolve a strategy by which a good quality solution can be created. Such research includes (Kitano, 1990; Hart et al., 1998; Ross et al., 2002; Ross et al., 2003).

Kitano (Kitano, 1990) employed a GA-based hyper-heuristic to optimise neural network design. Instead of encoding the network configuration directly, his GA chromosome consisted of a set of rules that can be used to generate networks. This approach was shown to be superior to a conventional GA.

Hart et al. (Hart et al., 1998) solved a real-world chicken factory scheduling problem using a GA based hyper-heuristic. The problem involved scheduling the collection and delivery of chickens from farms to the processing factories. The problem was deconstructed into two stages and two separate GAs were used to tackle the problem in each stage. In the first stage, the orders were split into suitable tasks and these tasks were then assigned to different "catching squads". The second stage dealt with the schedule of the arrival of these squads. The GA chromosome in the first stage represented a sequence of orders, a set of heuristics to split each order into suitably sized tasks and another set of heuristics to assign these tasks to the different "catching squads". The GA was used to evolve a strategy to build a good solution instead of finding the solution directly. The experimental results showed this approach is fast, robust and easy to implement.

Recently, Ross et al. (Ross et al., 2003) also proposed another GA based hyper-heuristic. The problem addressed one-dimensional bin packing. Instead of working on feasible solutions, like most local search approaches, the proposed hyper-heuristic operated on a partial solution and gradually constructed the solution using different rules (heuristics) until a feasible solution was obtained. The heuristic selection was based on the state of the current partial solution. Each state was associated with a rule or heuristic whose relationship with solution states was evolved by a genetic algorithm. The chromosomes of their GA were defined as a set of *blocks* and each block contained a set of parameters which was used to define a solution state and its corresponding heuristics. The algorithm was firstly trained on some benchmark problems and after the training, the fittest chromosome was then applied to every benchmark problem, 80% of which were solved to optimality.

Other meta-heuristic approaches have also recently been employed as a high-level strategy in a constructive hyper-heuristic framework. Burke et al. (Burke et al., 2006) used a tabu search algorithm to hybridise well-known graph colouring heuristics in a hyper-heuristic framework to solve several exam and course timetabling problems. The tabu search was used to search for a good heuristic permutation which was then used to create a solution according to this heuristic permutation. The algorithm could produce competitive results (compared with other state-of-the-art algorithms) when applied to a set of benchmark problems.

Local search hyper-heuristics

In local search hyper-heuristics, low-level heuristics usually correspond to several neighbourhood functions or neighbourhood exploration rules that could be used to transfer the state of the current solution. Below we review a list of papers that use this idea.

Some hyper-heuristics use ideas from reinforcement learning to guide the choice of the heuristics during the search (Cowling et al., 2001; Nareyek, 2003). In (Cowling et al., 2001), a sales summit scheduling problem was solved by a "choice function" based hyper-heuristic, in which the choice function dynamically selected suitable heuristics at each decision point. The computational results showed that the choice function based hyper-heuristic was superior to applying the heuristics randomly. Nareyek (Nareyek, 2003) used a non-stationary reinforcement learning procedure to choose heuristics in solving two combinatorial optimisation problems. The author discussed the advantages of the hyper-heuristic approach, especially in solving complex real-world problems in which the computational cost is expensive.

A GA based local search hyper-heuristic algorithm (hyper-GA) was proposed by Cowling et al. (Cowling et al., 2002) to solve a trainer scheduling problem. Here, a GA chromosome represented an ordering of the low-level heuristics that were going to be applied to the current state. A good sequence was evolved during the search corresponding to the given problem instance. The computational results showed that the GA based hyper-heuristic outperformed both a conventional genetic algorithm and a memetic algorithm which directly encoded the problem as a chromosome. An enhanced version of the hyper-GA was presented in (Han et al., 2002) which used an adaptive length chromosome.

Smith (Smith, 2002) proposed a memetic algorithm (MA) using the concept of *co-evolution* (see section 3.3.3.11 for an review of MA). In his approach, the idea is to evolve a local search strategy. The chromosome encodes the information that represents which local search method to apply and in which way (e.g. single call or steepest descent). Therefore, in his algorithm, the solution and the local search methods co-evolve simultaneously. The algorithm was shown to be superior to both a general genetic algorithm and a conventional memetic algorithm.

Burke et al. (Burke et al., 2003c) applied a tabu search based hyper-heuristic to a nurse rostering problem and a university course timetabling problem. In their hyper-heuristic algorithm, the set of heuristics were ranked according to their performances in the search history. A tabu list was also incorporated to prevent the selection of some heuristics at certain points in the search.

Kendall and Mohd Hussin used a similar tabu search hyper-heuristic algorithm in tackling university timetabling problems (Kendall and Mohd Hussin, 2004a). However, this algorithm is slightly different. In (Burke et al., 2003c), each low-level heuristic is associated with a weight that is dynamically updated according to the given heuristic's previous performance. Each time, the best non-tabu heuristic is chosen and applied. However, in (Kendall and Mohd Hussin, 2004a), all heuristic calls are tried and the best heuristic is selected and applied. Each heuristic that has been applied becomes tabu and will not be called within a given number of iterations (called *tabu duration*). Their later work (Kendall and Mohd Hussin, 2004b) also incorporated some heuristic acceptance criteria to enhance the performance, including a great deluge algorithm. The experimental results on the benchmark problems show that this algorithm can achieve considerable improvement over a manual solution and is competitive when compared to other algorithms published in the literature.

Other high-level strategies have also been investigated within the framework of hyper-heuristics. In (Burke et al., 2005), a case-based reasoning paradigm was used to guide the selection of timetabling heuristics. The case-based reasoning system maintains a database of case information about which heuristic works well on previous timetabling problem instances. For a new timetabling problem instance, the system automatically recommends a heuristic to solve the problem based on the knowledge stored in the database.

Burke et al. (Burke et al., 2003b) investigated a hyper-heuristic approach in an ant algorithm framework in solving a presentation scheduling problem. Ant algorithms analogise a colony of real ants that seek the shortest path between the nest and the food source. An ant algorithm can be described as an optimisation technique that searches for the best path in a graph and is usually used in route-planning optimisation problems (see section 3.3.3.12 for further discussions of the ant algorithm). In Burke et al.'s ant algorithm based hyper-heuristics, each vertex in the graph represents a low-level heuristic and there are directed edges connecting two vertices. Each ant is associated with a solution for the problem. Initially, a population of ants are randomly placed at different vertices. These ants are moved from one vertex to another, corresponding to transferring the associated solution to another solution utilising the heuristic represented by the destination vertex. The probability with which an ant chooses the next vertex to move to is dependent on the pheromone trail on the edge connecting the two vertices. When an ant moves from vertex *i* to vertex *j*, the corresponding low-level heuristic represented by vertex *j* is applied to the solution associated with the ant, generating a new solution. The ant then deposits a given amount of pheromone trail on the edge between *i* and *j*. The quantity of the pheromone that the ant deposits is proportional to the improvement the heuristic achieved over the previous solution. Therefore, after a few iterations, more pheromone is deposited on the edges which could improve a solution more frequently. The ant algorithm hyper-heuristic is different from a general ant colony optimisation algorithm. The vertices in an ant algorithm hyper-heuristic represent different low-level heuristics which can transfer the solution to a candidate solution. However, the vertices in a conventional ant algorithm represent solution components. For example, when applying a general ant algorithm to TSP, a vertex normally represents a city (Dorigo, 1992; Dorigo and Maniezzo, 1996).

Some interesting work was also carried out in (Cowling and Chakhlevitch, 2003), in which the hyper-heuristic was designed to manage a large set of low-level heuristics constructed by combining different “event selection” rules and “resource selection” rules. Instead of selecting a low-level heuristic from the large set of available low-level heuristics, the algorithm selected a heuristic from a candidate list which contained only a small subset of promising low-level heuristics. The size of the candidate list determined the degree of greediness and randomness of the hyper-heuristics. The authors also included a tabu list, which made tabu some badly performing heuristics from being selected within a given period. The algorithm was shown to be able to efficiently handle a real-world trainer scheduling problem.

Yet another type of hyper-heuristic was proposed by Mockus (Mockus, 1989), using the concept of the *Bayesian heuristic approach* to randomise and optimise the probability distribution of each heuristic call. The Bayesian heuristic approach is based on the analysis of average-case performance of the heuristics. It attempts to determine a set of parameters or a probability distribution such that the deviation from the global optimum is minimised. The method has been applied to a variety of discrete optimisation problems. See (Mockus, 1994; Mockus et al., 1997; Mockus, 2000) for further details.

In the above local search hyper-heuristics, the candidate solutions returned by low-level heuristics are either all accepted or only accepted if they are better than the current solution. However, these criteria may be too simple and not appropriate, as accepting all heuristic moves may lead to a random search. Similarly, the search may get stuck at local optima if the algorithm only accepts better solutions (we shall discuss this further in chapter 4). Recently, research has been carried out to improve the heuristic acceptance criteria in a hyper-heuristic framework. Bai and Kendall (Bai and Kendall, 2003) firstly introduced a simulated annealing acceptance criterion into the hyper-heuristic framework. More investigations and discussions of the simulated annealing hyper-heuristic are given in (Bai and Kendall, 2005b) and will also be presented in chapters 4 and 5. Ayob and Kendall (Ayob and Kendall, 2003) investigated a hyper-heuristic approach that uses a Monte Carlo acceptance criterion. Both algorithms have been shown to be superior to the choice function based hyper-heuristics (Cowling et al., 2001), which employed simple acceptance criteria. In addition, similar threshold acceptance algorithms have also been introduced into the hyper-heuristic framework when solving a mobile network frequency allocation problem (Kendall and Mohamad, 2004a; Kendall and Mohamad, 2004b).

3.3.3.4 Simulated annealing

Simulated annealing is a local search method inspired by Metropolis et al.'s algorithm to simulate the physical cooling process (Metropolis et al., 1953). Since its introduction as an optimisation tool (Kirkpatrick et al., 1983), SA has been intensively studied both in theory and application. The theoretical analysis of SA have been concerned with its convergence criteria, based on the fact that simulated annealing can be treated as a series of homogeneous Markov chains or a single non-homogeneous Markov chain. Research has proven that SA is able to asymptotically converge to an optimal solution if certain conditions are satisfied (Aarts and van Laarhoven, 1985; Lundy and Mees, 1986). However, these theories are not very useful in practice because guaranteeing an optimal solution often requires more iterations than an exhaustive search. However, this does not deter SA from being used in many applications. In fact, SA has been widely used to solve a variety of difficult problems owing to its simplicity of implementation and robustness in many problems, including graph partitioning and colouring, route-planning, layout design, sequencing and scheduling, timetabling, signal processing, etc. (Carnevali et al., 1985; Sechen et al., 1988; Johnson et al., 1989; Ogbu and Smith, 1990; Abramson, 1991; Johnson et al., 1991; Thompson and Dowsland, 1998; Burke and Kendall, 1999; Tian et al., 1999; Liu, 1999; Chen and Luk, 1999; Bouleimen and Lecocq, 2003). Discussions on other applications of SA are also given in (Dowsland, 1995; Henderson et al., 2003).

The procedure for simulated annealing is fairly simple. For a maximisation problem with objective function *f* and neighbourhood structure *N*, SA starts from an initial solution and repeatedly generates and transfers to a neighbour of the current solution. During this process, SA has the possibility of visiting worse neighbours in order to escape from local optima. Specifically, a parameter, called temperature *t*, is used to control the possibility of moving to worse neighbour solutions. The algorithm, starting from a high temperature, repeatedly decreases the temperature in a strategic manner (usually referred to as a cooling schedule) until the temperature is low enough, or some other stopping criteria are satisfied. In each iteration, the algorithm accepts all *uphill* (a move which increases the objective value for a maximisation problem) moves and some of the *downhill* (a decrease in the objective value for a maximisation problem) moves according to the Metropolis probability, defined by $\exp(\delta / t)$ where δ is the difference in the objective function between the new candidate solution and the current solution. A general simulated annealing algorithm for maximisation problem can be described by the figure 3-2.

```
Initialisation: initial solution $s_0$, temperature $t_s$, cooling function $\varphi(t)$,
number of iterations at each temperature nrep and a neighbourhood
definition $N$;
Repeat
    Repeat
        Randomly select $s \in N(s_0)$;
        $\delta = f(s) - f(s_0)$;
        If $\delta > 0$
            $s_0 = s$;
        Else if $\exp(\delta / t) > random(0,1)$
            $s_0 = s$;
        Endif
        If $f(s_0) > f(s_{best})$
            $s_{best} = s_0$;
        Endif
    Until iteration_count = nrep
    Set $t = \varphi(t)$;
Until the stopping conditions are met
Output $s_{best}$ as the best solution found.
```

Figure 3-2: A general simulated annealing algorithm for a maximisation problem
Source: (Dowsland, 1995)

Two important factors have to be carefully considered before implementing this general simulated annealing algorithm. These are the definition of neighbourhood structure *N* and the cooling schedule which is determined by 1) a starting temperature t_s; 2) temperature reduction function $\alpha(t)$; 3) the number of iterations at each temperature *nrep* and 4) stopping condition(s).

Starting temperature

The initial temperature should be high enough to allow “free moves” at the initial state such that the final solution is not dependent on the initial state (Dowsland, 1995). However, if one wants SA to start from a good quality solution created by some sophisticated heuristics, the initial temperature should not be too high. This is due to the fact that, when the temperature is too high, the algorithm accepts almost all of downhill moves (without specification, it is assumed that we are trying to maximise the objective function). In this case, the search, in fact, starts from a random initial solution. The effort of obtaining a high quality initial solution is, therefore, irrelevant.

A lot of research has been carried out in order to identify an optimal initial temperature or a method by which an initial temperature can be determined. However, this is very difficult because even if there is an optimal initial temperature, its value may be different from problem (or even problem instance) to problem. In practice, some estimation methods have been suggested instead. Kirkpatrick et al. (Kirkpatrick et al., 1983) suggested an initial temperature $t_0 = \delta_{\max}$ where $\delta_{\max}$ is the maximal difference in the objective value between two neighbouring solutions. Another more intuitive method is setting an initial temperature value such that the ratio of accepted downhill moves to all neighbourhood moves is equal to a predefined value.

One way to get an estimation of this value was described in (Dowsland, 1995). Starting from a large initial temperature value, a number of neighbourhood moves are performed and the corresponding acceptance ratio is monitored. If the targeted acceptance ratio is not reached, the temperature is modified (decreasing or increasing depending on relationship between the current acceptance ratio and the given ratio) and the procedure is repeated until the predefined acceptance ratio is reached. The current temperature is then chosen as the initial temperature.

Johnson et al. (Johnson et al., 1989; Johnson et al., 1991) suggested using the average cost difference of a set of sample neighbouring solutions to approximate the initial temperature of a given acceptance ratio of downhill moves. Suppose $\overline{\delta}$

represents the average cost difference of a set of sampled neighbouring solutions and r_0 is the given acceptance ratio allowed at the beginning of the search, the initial temperature can be calculated by $t_0 = -\overline{\delta} / \ln(r_0)$.

Another iterative procedure was proposed by Ben-Ameur (Ben-Ameur, 2004) to obtain a more accurate estimation of the initial temperature. The author also discussed some properties of the acceptance ratio of bad moves. Some of the latest theoretical work can also be found in (Cohn and Fielding, 1999).

Cooling schedule

Considerable research has been carried out in the pursuit of a good cooling strategy. Two of the most popular methods are geometric cooling $\varphi(t) = \alpha t$ ($\alpha < 1$) and a non-linear cooling function $\varphi(t) = t/(1+\beta t)$ (where β is a very small positive value) proposed by Lundy and Mees (Lundy and Mees, 1986). In the geometric cooling function, the temperature reduction rate is a constant and usually takes value in the range of [0.8, 0.99]. However, in the Lundy and Mees' cooling function, the temperature drops very quickly when the temperature is high and relatively slower when the temperature is low. At each temperature only one iteration is executed. Both cooling schedules are monotonic decreasing functions. However, an optimal cooling schedule may be not monotonic and be dependent on different problems (Dowsland, 1995). Therefore, several other cooling strategies have also been proposed which take into account the history of the search and allow temperature increases during the search (also referred to as *reheating* in some publications).

When the temperature becomes very low, SA degenerates into a hill climbing algorithm and most of the time is being wasted in generating and rejecting inferior solutions. Connolly (Connolly, 1990) suggested that it is not necessary to reduce the temperature from a high value to zero. Instead, the temperature could be held constant throughout the search. He tested the idea on quadratic assignment problems and concluded that there exists a fixed temperature at which the performance is optimised. However, this optimal temperature might be different from problem to problem and is very difficult to obtain.

Some other researchers suggested that the temperature could be “reheated” if the search gets stuck at a local optimum (Kirkpatrick et al., 1983; Dowsland, 1993). In (Kirkpatrick et al., 1983), the problem under consideration was the travelling salesman problem and reheating was carried out in an interactive way, which let the user monitor the current solution and the moves. A “reheat” process was triggered whenever a user found that the algorithm got stuck at a local optimum. Dowsland (Dowsland, 1993) suggested a more frequent reheating process in solving rectangle packing problems, in which the temperature was reduced by the function $\varphi(t) = t/(1+\beta t)$ if an uphill move was found and whenever a move was rejected, the temperature was increased according to the function $\varphi'(t) = t/(1-\gamma t)$ where $\beta = k\gamma$. Hence, if the number of rejected moves is greater than *k* multiplied by the number of accepted moves, the temperature begins to “heat up”. The SA with this cooling schedule reported good experimental results.

Some other cooling functions have been introduced in (Dowsland, 1995; Aarts and Korst, 1998) which employed more complex temperature update functions.

Stopping condition(s)

The conventional simulated annealing algorithm stops when the temperature reaches zero or a value small enough such that the algorithm converges to a local optimum. Choosing an appropriate value of the stopping temperature can be based on experiments or one can monitor the acceptance ratio of downhill moves and the algorithm stops when the ratio decreases below a given very small value (0.01 for example). Other stopping conditions were also used, such as the allowed computation time, the number of consecutive non-improvement moves, etc.

Neighbourhood design

The neighbourhood structure is another vital aspect which influences the performance of SA, although its importance has not really been recognised until the recent advances and success of variable neighbourhood search (Mladenovic and Hansen, 1997; Hansen and Maldenovic, 2001) and very large neighbourhood search (Yeo, 1997; Gutin, 1999; Ahuja et al., 2000).

Some earlier researchers concentrated on the impact of the neighbourhood size and suggested that reducing its size during the final stages of the annealing could produce better results or speed up the algorithm. Greene and Supowit (Greene and Supowit, 1986) proposed a *rejectionless* SA. In their algorithm, each possible move was associated with a weight based on its effect on the cost function if it was applied. The probability of selecting a move was then based on the amount of the contribution of its weight to the total weight. The algorithm was tested on a logic partitioning problem and the results showed that the proposed algorithm could accelerate the search without undermining the solution quality. However, it does require extra memory. In (Sechen et al., 1988), simulated annealing was used to optimise a cell layout problem for VLSI design. In their algorithm, the neighbourhood size was reduced by prohibiting large distance moves when the temperature was getting low. Similar ideas were also tested in (Tovey, 1988) where the experimental results showed that probabilistically giving preference to those promising subset solutions performed better than completely restricting the search in a small subset space. He stated that the possible reason was that exclusion of some solutions might be detrimental to the neighbourhood reachability. Recently, Steinhofel (Steinhofel et al.,

2003) found that a non-uniform sampling SA performed better than a uniform sampling SA when tested on job shop scheduling problems.

It is suggested that neighbourhood structures for SA should be *symmetric* (i.e. it is possible to return to the state just visited) or at least *reachable* (i.e. every state should be reachable from every other state) (Dowsland, 1995). However, there are no general guidance rules because this is a problem-specific decision. Tian et al. (Tian et al., 1999) investigated the effect of neighbourhood structures of the SA algorithm in solving three permutation optimisation problems: travelling salesman problem (TSP), flow-shop scheduling problem (FSP) and quadratic assignment problem (QAP). Six types of neighbourhood structures were designed and proven to be asymptotically convergent. The performance of these neighbourhood structures were compared across different sizes of instances of the three problems. The results showed that the best neighbourhood structures for three problems were completely different. SA with the neighbourhood that performed best on TSP was significantly inferior to SAs with some of the other neighbourhood structures when applied to FSP and QAP. This shows that for different problems (although they might share some common properties), SA should choose different neighbourhood structures in order to obtain good results. However, defining the best neighbourhoods is not an easy task.

Csondes et al. (Csondes et al., 2002) used a so called *adaptive variable neighbourhood* structure in their SA algorithm in solving two real-world optimisation problems. The solution space was a variable vector that satisfied all constraints. When the temperature was high, more variables were allowed to be flipped. However, when the temperature was low, new solutions were only sampled by applying a small number of variable flips. The results were competitive with those obtained by existing commercial software.

3.3.3.5 Tabu search

Tabu search (TS) was originally proposed by Fred Glover in 1977 (Glover, 1977). However, it did not become a popular combinatorial optimisation method until his later work (Glover, 1989; Glover, 1990). Tabu search is a single point meta-heuristic approach that has found a variety of applications in practice. Tabu search differs from other local search approaches in that it makes use of historical information to prevent the search from cycling and becoming trapped in a local optimum. The *tabu list* is a *short-term memory* of recent neighbourhood moves that are prohibited during the search in order to prevent the search from going back to the recently visited points in the search space. The *length* of the tabu list decides how many moves are stored in the list and the *tabu tenure* defines how many iterations of each move in the tabu list are tabu (i.e. cannot be called). Although the tabu list is helpful in avoiding cycling during the search, in some cases, it may restrict the search too much such that some promising moves are prohibited. Therefore, most tabu search algorithms also incorporate a mechanism, a so called *aspiration criteria*, which is used to mitigate the strength of the tabu list.

Some *long-term memories* are also used that store a record of the entire search process for the purpose of the *intensification* and *diversification*. For example, in a *frequency memory*, one accounts the number of occurrences of a particular attribute that belongs to a solution or a move during the search. There could be many types of attributes. For example, an attribute can be a variable taking a specific value or an operator which sets a variable from a value to another. A simple intensification and diversification method can be carried out by introducing incentive or penalty values to modify the evaluation of moves in reference to the frequency memory (Glover F. and Laguna, 1995). Some other diversification methods in the tabu search were also discussed in (Soriano and Gendreau, 1996). For the detailed discussion and implementation of tabu search, one can refer to (Glover and Laguna, 1997). One can also refer to (Gendreau, 2002) for a survey of the recent advances in tabu search.

In practice, TS approaches have been widely used in many areas, including scheduling, transportation and routing, telecommunications, bioinformatics, network design and graph partitioning and colouring (Widmer and Hertz, 1989; Reeves, 1993; Skorin-Kapov and Vakharia, 1993; Taillard, 1994; Gendreau et al., 1994; Mazzola and Schantz, 1995; Rolland et al., 1996). A full list of applications is given in (Glover and Laguna, 1997).

3.3.3.6 Variable neighbourhood search (VNS)

Variable neighbourhood search (VNS) is another local search meta-heuristic that has recently been proposed for combinatorial optimisation problems (Mladenovic and Hansen, 1997). The approach differs from general meta-heuristics in that the algorithm systematically changes the neighbourhoods. A basic VNS algorithm consists of an initialisation stage and an iterative stage. The initialisation stage involves constructing an initial solution and defining a set of neighbourhood structures and their sequence. Note that the sequence of the neighbourhoods usually reflects an increasing order in the distance from the current solution to the given neighbourhood (Mladenovic and Hansen, 1997). The iterative stage consists of three subroutines, *shaking, local search* and *move decision*. The shaking works as a diversification element which samples a random solution from the current k^{th} neighbourhood of the current solution. The local search subroutine is then applied to this sampled solution to improve it to a local optimal solution. If it is better than the current solution, the search moves to this solution and the current neighbourhood is set to be the first neighbourhood in the list. Otherwise, the search does not move to this solution while the current neighbourhood is set to be the next neighbourhood in the list. The process above is only a basic VNS algorithm. There are several variants. The simplest one is *variable neighbourhood descent* (VND), where the local search method will explore the whole neighbourhood and return the best neighbour in it. Some other variants include *variable neighbourhood decomposition search* (VNDS), *skewed VNS* (SVNS) and *parallel VNS* (PVNS). VNS can also be hybridised with tabu search, simulated annealing and GRASP (see the next section 3.3.3.7). Recent advances and applications of this approach can be found in (Hansen and Maldenovic, 2001; Hansen and Maldenovic, 2003).

3.3.3.7 Greedy randomised adaptive search procedure (GRASP)

GRASP is multi-start meta-heuristic approach that explores the search space from different points (solutions) (Feo and Resende, 1989; Feo and Resende, 1995). Each thread of the search can be divided into two phases: the construction phase and the local search phase. The construction phase is responsible for creating a good quality solution from which the local search starts. Figure 3-3 illustrates a basic GRASP algorithm. It can be seen that the algorithm is an iterative procedure. Each iteration involves creating an initial solution (solution construction phase) and then performing a local search from it (local search phase). From an empty solution, the construction phase repeatedly inputs a candidate element into the partial solution until a complete solution is constructed. The selection of the candidate elements is based on a kind of “peckish” mechanism (a mechanism to combine randomness and greediness). Before constructing the solution, all non-initialised candidates are sorted according to a function or a criterion (usually the incremental costs or benefits if the given candidate element is input into the partial solution) and the first *k* elements are stored in a *restricted candidate list* (RCL) (this is the *greedy* part of the algorithm).

The algorithm is *randomised* in the sense that the candidate elements are randomly selected from the restricted candidate list rather than the best candidate element. The quality of the elements in the RCL can also be controlled by a threshold value. The algorithm is called *adaptive* because all candidate elements are evaluated and ranked each time a new element is incorporated into the solution. The solution obtained above is further improved in the local search phase, which can be a hill-climbing algorithm or any other local search approaches. During the process, the best solution is memorised and returned when the stopping criteria are met. A recent survey can be found in (Resende and Ribeiro, 2003).

```
Start with an empty solution;
Repeat
    Repeat
        Evaluate all non-initialised candidate elements by a function;
        Construct the restricted candidate list (RCL);
        Select a candidate from the RCL and apply it to the current
        solution;
    Until current solution is complete
    Apply local search to the current solution;
    Memorise the best solution;
Until stopping conditions are met.
```

Figure 3-3: Pseudo-code of the basic GRASP
Source: (Resende and Ribeiro, 2003)

3.3.3.8 Guided local search (GLS)

The main idea behind the guided local search meta-heuristic is to guide the local search algorithm to escape from local optima by introducing a penalty function into the objective function (Wang and Tsang, 1991; Wang and Tsang, 1994; Voudouris and Tsang, 1997). Once a local search algorithm gets stuck at a local optimum, certain features in the current solution are selected and punished by adding a penalty value to the objective value. Hence the GLS algorithm is able to guide the search efforts to more attractive areas. To implement GLS, one needs to define a set of features for a problem and a modified objective function that takes account of the features to punish. Each feature is associated with a penalty and their values can be changed dynamically during the search. However, selecting what kind of features to punish and defining the modification strategies of the penalty value may not be easy for various problems. One can refer to (Voudouris and Tsang, 2003) for a survey of recent advances and applications of GLS.

3.3.3.9 Iterated local search (ILS)

Iterated local search is another meta-heuristic proposed for combinatorial optimisation problems. The main components in an ILS algorithm include *LocalSearch*, *Perturbation* and *AcceptanceCriterion* (Lourenco et al., 2003). The algorithm starts from an initial solution, created randomly or by a greedy heuristic, and keeps improving the current solution by the *LocalSearch* until a local optimum is reached. A candidate solution is then sampled from the current solution using the Perturbation method. The LocalSearch is then applied to improve the candidate solution to a local optimum. The search then moves to this candidate solution if the AcceptanceCriterion is true. Otherwise, the move is rejected. Lourenco et al. (Lourenco et al., 2003) provided a basic ILS procedure, shown in figure 3-4.

```
$s_0$ = GenerateInitialSolution();
$s^*$=LocalSearch($s_0$);
Repeat
    s'=Perturbation($s^*$,history);
    $s^{*'}$=LocalSearch($s'$);
    $s^*$=AcceptanceCriterion($s^*$, $s^{*'}$, history);
Until stopping conditions are met.
```

Figure 3-4: Pseudo-code of a basic ILS
Source: (Lourenco et al., 2003)

It can be seen that the algorithm is very similar to a basic VNS. The main difference between them is that the perturbation in ILS uses historical information to bias the solution sampling. While in VNS, the solution is sampled by systematically changing the *perturbation strength*, a measure that describes the magnitude of difference between the current solution and the sampled solution (VNS *shakes* the current solution by using different neighbourhoods with increasing cardinality).

3.3.3.10 Genetic algorithms

Genetic algorithm, also abbreviated to GA, was firstly proposed by Fraser (Fraser, 1957) and Bremermann (Bremermann, 1962) independently. Holland's book *Adaptation in Natural and Artificial Systems* (Holland, 1975) is also often cited as one of the seminal works for GA. The idea of GA comes from the natural selection principle of *survival of the fittest*, which believes that only the fittest individuals will survive through many generations. A genetic algorithm holds a population of solutions that evolves from one generation to the next (Goldberg, 1989; Forrest, 1993; Michalewicz, 1996). For an optimisation problem, a solution (*individual*) is usually encoded in a specially designed string (called a *chromosome*). A given number of individuals, called a *population*, is maintained and evolves from one generation to another. A new population is generated by copying some fitter individuals from the current population and selecting some newly created individuals using genetic *operators,* such as *crossover* and *mutation*. The algorithm stops once the termination criteria are met. To implement a genetic algorithm, one needs to decide a solution encoding scheme, operators (crossover, mutation), and a selection method as well as various parameters values, such as population size, number of generations, crossover probability and mutation probability.

Although the process of the genetic algorithm is fairly simple, there are several important issues which need careful consideration. The first is the solution encoding system. Falkenauer (Falkenauer, 1998) found that the encoding, the process of mapping from the *phenotype* (the representation of a solution) to the *genotype* (the representation of the chromosome), could significantly affect the performance of the GAs and suggested that the encoding should be a *one-to-one* mapping procedure. That is, one solution of the problem maps to one chromosome. If one solution maps onto several chromosomes, the encoding is redundant and could impact the efficiency of the GA. However, if the encoding is in a *many-to-one* mapping fashion (several solutions of the problem correspond to one chromosome), such an encoding lacks the details of the problem although it may be beneficial in some cases because it reduces the size of search space.

Once the encoding system is decided, one needs to design the genetic operators. Several types of operators have been proposed in the past. However, crossover (also called recombination) and mutation are the two most popular operators. The role of crossover is to inherit some promising traits from two (possibly more) parents. Mutation is believed to be a beneficial supplement to the crossover by introducing some new traits which are not currently present in the parent solutions (Davis, 1987; Goldberg, 1989).

While it is agreed that fitter individuals should have a larger probability of being selected for the new generation, it is also very important to allow a few “less-fit” individuals to increase the *diversity* of the population. There are several ways to achieve this, among which tournament selection and roulette-wheel selection are frequently used. In tournament selection, only a small subset of individuals are chosen and compared, and the fittest ones are selected to be the parents. While in the roulette-wheel selection, the selection probability of an individual is proportional to its fitness value (Coley, 1999).

A good introduction to genetic algorithms can be found in (Sastry et al., 2005). Some other references with regard to genetic algorithms are available in (Goldberg, 1989; Davis, 1991; Beasley et al., 1993; Reeves, 1995; B鋍 k, 1996; Mitchell, 1996; B鋍 k et al., 1997; Michalewicz and Fogel, 2000).

3.3.3.11 Memetic algorithms

Genetic algorithms are thought to be suitable for problems with large search spaces. Because most genetic algorithms do not take advantage of the problem-specific knowledge, it is possible that some local optima are missed or are not being explored efficiently during the search (a genetic operator is said to be *blind* if it does not use problem-specific knowledge in determining where and how to apply an operation on a chromosome (Moscato and Cotta, 2003)). In this context, some local heuristic searchers can be incorporated into a genetic algorithm with the aim of enhancing the GA’s performance. Such an altered algorithm is named a memetic algorithm in (Moscato, 1989). In this sense, the memetic algorithm is also deemed a hybrid method. A good introduction to memetic algorithms can be found in (Moscato and Cotta, 2003). For recent advances in the theory and application of memetic algorithms, one can refer to (Hart et al., 2003).

3.3.3.12 Ant colony optimisation algorithm (ACO)

The ant colony optimisation algorithm originated from Dorigo’s Ant System (Dorigo, 1992; Dorigo and Maniezzo, 1996) which simulates an ant colony seeking a shortest path between a food source and a nest (Deneubourg et al., 1990). Although the behaviour of each ant is independent and asynchronous, communication among them is mediated by a *pheromone* trail. When deciding which path to use an ant chooses the shortest path to the food source by exploiting the level of the pheromone on each available path. At the initial state, no pheromone trail information is available and ants follow a random route. However, when an ant finishes a tour, it deposits a certain amount of pheromone on the path it takes according to the distance of that tour. That is, if a shorter path is found by an ant, more pheromone is deposited on every edge of the path. After a period of time, the shortest route will have high levels of pheromone so that the other ants are more likely follow this route.

To convert the above process to an ant colony optimisation technique, some modifications have to be made. For example, a proportion of pheromone *evaporates* after each iteration in order to prevent the system converging prematurely. To improve algorithmic performance, heuristic information and some local search procedures may be used in some ACO algorithm (Dorigo et al., 1996; Dorigo and Gambardella, 1997a; Dorigo and Gambardella, 1997b; Bullnheimer et al., 1999; Gambardella and Dorigo, 2000). For more publications regarding to ACO, one can see the web page maintained by Dorigo (http://iridia.ulb.ac.be/~mdorigo/ACO/).

3.3.3.13 Evolutionary strategies

Evolutionary strategies (EA) were firstly studied by Rechenberg (Rechenberg, 1965; Rechenberg, 1973) and later developed by Schwefel in his PhD thesis (Schwefel, 1975). Although closely related to genetic algorithms, EA's imitate genetic processes in the phenotype (i.e. solutions are encoded explicitly), in contrast with implicit solution representations in genetic algorithms. Initially, ES only used mutation and were mainly used to tackle problems with real valued variables. Later research work extended EA to use crossover (as a supportive operator) and solve discrete-variable problems. For more details of EA, one can referee to (Schwefel, 1975; B鋍k, 1996; B鋍k et al., 1997; Fogel, 1998; Yao, 1999; Fogel, 2000).

3.3.3.14 Scatter search and path-relinking

Scatter search is a population-based meta-heuristic, having been proposed for solving combinatorial and nonlinear optimisation problems (Glover, 1977). Scatter search takes advantage of a set of points “scattered” in the solution space and strategically combines some of them to generate new promising solutions (Glover et al., 2003).

```
Setp 1. Generate a large set of trial solutions P using diversification
        generation method with |P|=PSize. Improve these solutions by an
        Improvement Method.
Setp 2. Select a small set of distinct and diverse solutions, called reference
        set, RefSet, from P with |RefSet|=b <<PSize. Order the sets
        according to their objective function value non-increasingly.
Setp 3. Use Subset Generation Method to generate NewSubsets from RefSet
        which includes at least one new solution.
Setp 4. Select every subset from NewSubsets, apply Solution Combination
        Method to obtain one or more new solutions s.  If s is not in RefSet
        and is better than the worst solution s’ in RefSet, include s in RefSet
        and delete s’ from RefSet.
Setp 5. If at least a new solution is added into RefSet in the step 4, then go
        to the step 3; otherwise stop the procedure.
```

Figure 3-5: The pseudo-code of a basic scatter search algorithm
Source: (Glover et al., 2003)

Figure 3-5 illustrates the procedure of a basic scatter search algorithm. The algorithm starts from a group of diverse trial solutions constructed by a *diversification generation method*. An *improve method* is then applied to enhance these trial solutions, from which a small set of distinct and diverse solutions are selected, termed the *reference set*. The algorithm then enters an iterative procedure. At each iteration, a *subset generation method* is used to produce several subsets of the *reference set* and a *solution combination method* then transforms each subset into one or more combined solutions. The *reference set* is then updated by a *reference set* *update method* in order to maintain the quality of the solutions in this set both in terms of objective function value and its diversity. The algorithm stops when no new solution is added into the *reference set*. Note that the solution combination method employed in scatter search is similar to the crossover operator in the genetic algorithm although scatter search allows the combining of more than two solutions.

In many single-point based search algorithms, such as simulated annealing and tabu search, the previous local optimal solutions are discarded once a better solution is found. However, path-relinking enables such algorithms the capability to obtain better quality solutions by exploring trajectories between these local optimal solutions. Path-relinking was proposed to integrate intensification and diversification during the search (Glover and Laguna, 1997). The approach generates new solutions by exploring trajectories that connect high-quality solutions. Specifically, the algorithm firstly compares the *symmetric difference* between two elite solutions s_1 and s_2. Based on this comparison, a set of moves M are identified which could transfer the solution s_1 (*initial solution*) to s_2 (*guiding solution*). Path relinking then starts from the initial solution and repeatedly applies best moves from M until the guiding solution is attained. Many applications have shown that better quality solutions are usually found along this trajectory (Glover and Laguna, 1997).

3.4 Summary and Remarks

This chapter has overviewed current popular optimisation techniques. Both exact methods and (meta-)heuristic methods were reviewed and some meta-heuristic approaches were emphasised due to their advantages in tackling NP-Hard problems that are germane to this thesis.

The “no-free-lunch theorem” tells us that the performance of a meta-heuristic method may be dependent on different problems and no algorithm performs better than any other algorithm when considering all possible problems. Even so, it is still possible that we can find an algorithm or a set of specific algorithms that generally performs better than other algorithms on a set of problems under consideration. This is because the problems that are of interest may be only a small subset of all possible problems. In conventional methods, simulated annealing for example, a user may experiment with a set of parameters (for SA, the parameters can be the different neighbourhood structures, cooling schedules, etc.) on the tested problem instances and pick the best set of parameters. However, on many occasions, it is found that finding the best set of parameters is difficult because the best set of parameters may be different from instance to instance.

As a proposed framework to raise the generality of the conventional meta-heuristics, hyper-heuristic approaches make use of several neighbourhood exploration heuristics simultaneously. The generality is achieved by choosing the most appropriate heuristics according to the different problem instances or the different search states when solving a specific instance. By utilising several neighbourhood exploration heuristics, hyper-heuristics can broaden the search space and dynamically change the search direction according to the characteristics of the different problems. Due to these advantages, we shall adapt hyper-heuristics as a solution methodology for shelf space allocation problems.

As mentioned above, previous researchers have adapted several conventional meta-heuristic approaches for the shelf space allocation problem. However, none of them has published problem data sets on which their experiments were based. This makes it difficult to justify our proposed approach by comparing with those conventional methods. Furthermore, another difficulty arises from the fact that we have used a more practical model that is different from the ones presented in the literature. Alternatively, it is convenient to adapt the hyper-heuristic to the well-studied bin-packing problem, which is closely related to the shelf space allocation problem. The bin-packing problem has been studied intensively, with a large number of benchmark data sets in the literature. If the hyper-heuristic can obtain promising results on the bin packing problem and due to the similarity between the bin packing and shelf space allocation, we hypothesise that a hyper-heuristic will also perform well on shelf space allocation problems. In the next chapter, we shall introduce a simulated annealing hyper-heuristic and adapt it to the bin packing problem. Its performance is evaluated by comparing with the best results obtained by other meta-heuristics on the benchmark problems. If the algorithm can produce competitive results compared with results reported in the literature, we can then adapt this algorithm to the shelf space allocation problem. This takes very little work because of the advantages of hyper-heuristic (as shown in chapter 5 and 7, we only need to replace a new set of low-level heuristics and the objective function). However, it may not be so easy to adapt other methods to the shelf space allocation.

CHAPTER 4. A SIMULATED ANNEALING HYPER-HEURISTIC ALGORITHM FOR THE BIN PACKING PROBLEM

4.1 Introduction

As discussed in chapter 3, hyper-heuristics have been proposed as a generic optimisation framework for a range of problems. One of the advantages of hyper-heuristics is their ability to adapt to different problem instances by calling different low-level heuristics. This chapter analyses the drawbacks of current hyper-heuristics and proposes a new hyper-heuristic algorithm which incorporates a simulated annealing acceptance criterion. Instead of being applied directly to the shelf space allocation problem, the algorithm is initially tested on the well-studied bin packing problem. This is because there is no benchmark data available for the shelf space allocation problem in the literature. Therefore, it is difficult to evaluate the performance of this hyper-heuristic in comparison to other optimisation methods for shelf space allocation problems. It might be argued that this can be done by implementing all algorithms and comparing their results. However this would require a considerable development effort. The problem is compounded by the fact that each algorithm may have one or more parameters to be set and tuning those parameters could take considerable time. However, the bin packing problem, a well-known NP-Hard problem, which is closely related to shelf space allocation problems, has been intensively studied and the experimental results by different approaches on a large number of benchmark data sets are available in the literature. Therefore, it is applicable to test the performance of this hyper-heuristic algorithm on this well-known problem and if it is successful, the application of this algorithm to the shelf space allocation problem is more likely to be successful because they are closely related problems. Furthermore, this experimental strategy will also provide us with the opportunity to test the generality of this hyper-heuristic approach across different (although related) problems. The main contents of this chapter are drawn from (Bai and Kendall, 2005c).

4.2 Simulated Annealing Hyper-heuristics

4.2.1 Background

As an emerging search method, hyper-heuristics have received recent attention due to their adaptive nature. Hyper-heuristics have been used to either construct a solution (Ross et al., 2003) (constructive hyper-heuristic) or operate as a local search method (Cowling et al., 2001; Cowling et al., 2002; Burke et al., 2003b; Burke et al., 2003c) (local search hyper-heuristics). Ross et al. (Ross et al., 2003) used different packing heuristics to gradually construct a solution. The choice of the "right" heuristic was based on the knowledge of the state of the current partial solution. The solution states included the proportions of *huge*, *large*, *medium* and *small* items remaining to be packed. The results showed that it produced better solutions than when utilising just a single heuristic. However, it might not be easy to define the right set of solution states for many problems. Furthermore, only around 80% of the optimal solutions have been found, much less than other exact methods and local search algorithms (Falkenauer, 1996; Scholl et al., 1997; Valerio de Carvalho, 1999; Fleszar and Hindi, 2002).

The hyper-heuristics in this thesis focus on the local search hyper-heuristics which are used in (Cowling et al., 2001; Cowling et al., 2002; Burke et al., 2003b; Burke et al., 2003c), where hyper-heuristics either explicitly or implicitly focus on selecting the "right" low-level heuristics at every decision point. This selection is usually based on the historical performance of each heuristic (Cowling et al., 2001; Burke et al., 2003c). A typical example of these approaches is the *choice function* based hyper-heuristic in (Cowling et al., 2001). In section 4.3.3, we will also apply this algorithm to the bin packing problem for comparison purposes. Therefore, the next section gives a brief description of this approach.

4.2.2 Choice function based hyper-heuristic

The basic idea of a choice function based hyper-heuristic is that the selection of which low-level heuristic to call at each decision point is guided by the choice function, a learning mechanism that integrates both intensification and diversification strategies during the search. In the choice function based hyper-heuristic it is assumed that if a heuristic or a sequence of heuristics has previously performed well in the search, it may perform well in the future. The choice function dynamically ranks the different low-level heuristics according to their historical performance. In (Cowling et al., 2001), the choice function considers the recent performance of each low-level heuristic (f_1), recent improvement for consecutive pairs of low-level heuristics (f_2) and the amount of time elapsed since the given heuristic has been called (f_3). More specifically, f_1, f_2 and f_3 are defined as follows:

$$f_1(h_j) = \sum_n \alpha^{n-1}\left(\frac{I_n(h_j)}{T_n(h_j)}\right) \tag{4-1}$$

$$f_2(h_k, h_j) = \sum_n \beta^{n-1}\left(\frac{I_n(h_k, h_j)}{T_n(h_k, h_j)}\right) \tag{4-2}$$

$$f_3(h_j) = \tau(h_j) \tag{4-3}$$

where $I_n(h_j)$ (respectively $T_n(h_j)$) is the change in the objective function (respectively the amount of computational time spent) the *n*th time the heuristic h_j was called. Similarly, $I_n(h_k, h_j)$ (respectively $T_n(h_k, h_j)$) is the change in the objective function (respectively the amount of computational time spent) the *n*th time the heuristic h_j was called immediately after heuristic h_k. While $f_3(h_j)$ records the amount of time elapsed since heuristic h_j was called the last time. Overall, the choice function is defined as:

$$CF(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j) \tag{4-4}$$

It can be seen that both f_1 and f_2 describe the aggregate performance of the previous *n* calls of each heuristic or pair of heuristics. They are used as a method to intensify the search and f_3 is used as a diversification strategy. α, β and δ are scaling parameters to balance the different terms. Values of these parameters are changed adaptively according to the magnitude of recent improvement in the objective function and the corresponding CPU time consumed. The detailed procedure is provided in Eric Soubeiga's PhD thesis, together with several applications and experimental analysis (Soubeiga, 2003).

In the above hyper-heuristic approach, the main focus is a learning mechanism that can intelligently choose between heuristics. Once a heuristic is chosen or recommended by the choice function, it is used to produce a new candidate solution which is usually accepted by very simple rules. For example a new candidate solution may be accepted straight away or only better solutions are accepted, as was the case in (Cowling et al., 2001). However, in this thesis we are concerned with improving the acceptance criteria in the hyper-heuristic framework. Of course, the high-level heuristic selection strategy is important in adapting the search to different search states. However, if a low-level heuristic adopts any random elements, identical calls of this heuristic may produce different solutions. Even if the high-level strategy is “intelligent” enough to pick up such a heuristic due to its good previous performance, it is still possible that this heuristic may be detrimental to the search due to the random elements existing in it. Take the tabu search based hyper-heuristic proposed in (Burke et al., 2003c) as an example. When solving the timetabling problem, one low-level heuristic that contains randomness was defined as “*swap the timeslots of two random events*”. When applying this heuristic to the current solution, it may produce a better solution which is desirable for the search. However, it is also possible that this heuristic produces an inferior solution which may not be desirable. Currently, the hyper-heuristic algorithms reported in the literature either reject all inferior solutions or accept them completely. In the first case, the search corresponds to a hill climbing and is prone to getting stuck at local optima. In the later case, it may lead the search in unexpected directions.

It is for this reason that we introduce a new acceptance criterion into the hyper-heuristic framework. Hill climbing and random search actually correspond to two simple acceptance criteria. Hill climbing only accepts those moves which can improve the objective value, while random search accepts all moves. Simulated annealing has been shown to be a robust combination of these two acceptance criteria. SA accepts all objective-improving moves and some of the objective-detrimental moves in a systematically-controlled way. It is based on this consideration that we incorporate simulated annealing into the hyper-heuristic framework and use it as a more intelligent acceptance criterion. We have called this approach a simulated annealing hyper-heuristic.

The importance of the acceptance criterion in a hyper-heuristic framework was also echoed by recent research which investigated other acceptance criteria, including the criteria based on Monte Carlo algorithms (Ayob and Kendall, 2003) and great deluge algorithms (Kendall and Mohamad, 2004a; Kendall and Mohamad, 2004b). These works, together with the research carried out in this thesis, will be valuable for the development of a more powerful and efficient hyper-heuristic optimisation system.

4.2.3 Simulated annealing hyper-heuristics

Figure 4-1 shows a general framework for the simulated annealing hyper-heuristic proposed in this thesis. The system is very similar to other forms of hyper-heuristics except that a simulated annealing algorithm is used as an acceptance criterion. At each iteration, the algorithm selects a heuristic from the set of low-level heuristics available and applies it to the current solution. If the solution generated by this heuristic is better than the current solution, it is accepted. Otherwise, it is accepted according to a *Metropolis probability*. The temperature of the simulated annealing is then modified. When the stopping conditions are met, the system terminates and outputs the best solution found so far. Note that the proposed hyper-heuristic does not conflict with the existing local search hyper-heuristics. The selection of the heuristics could be in a random way or by utilising some intelligence that has been proposed in other hyper-heuristic frameworks. In this thesis, we require all of the solutions generated by the low level heuristics to be feasible, i.e. the low level heuristics searches in the feasible solution space. In section 4.3.3, we will apply this simulated annealing hyper-heuristic to the one-dimensional bin-packing problem. Strategies that are used to select which heuristic to call and parameters related to simulated annealing will be given in section 4.3.3.

Readers at this point might argue that this framework is no more than a simulated annealing algorithm with multi-neighbourhoods. Of course, it is similar but it is not exactly the same. Firstly, the simulated annealing hyper-heuristic biases the exploration of the neighbourhood by *heuristically* sampling the candidate solutions (using different low-level heuristics) rather than sampling them uniformly from the

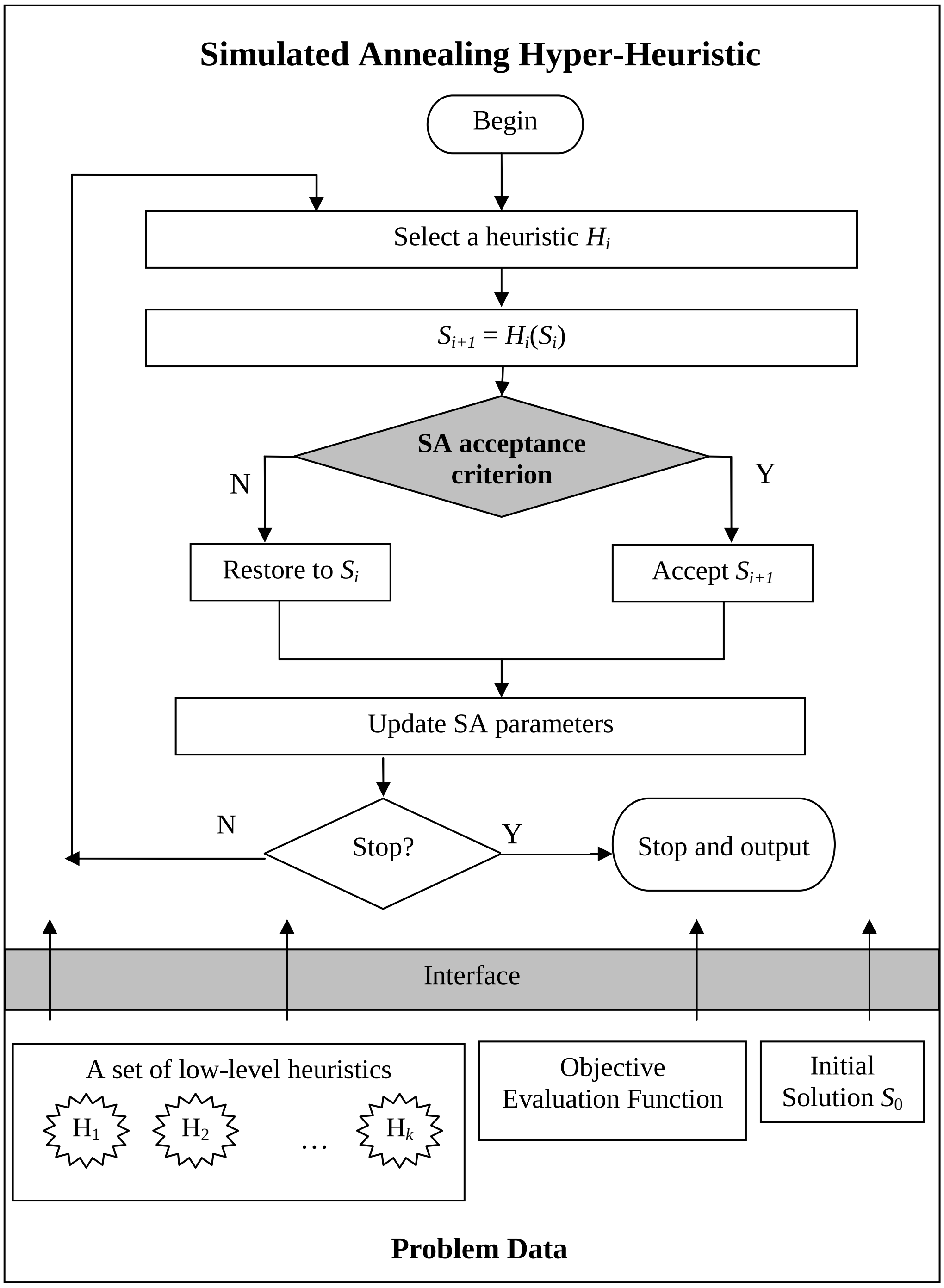


Figure 4-1: The framework of simulated annealing hyper-heuristics for a maximisation problem

given neighbourhood as does a traditional simulated annealing algorithm with multi-neighbourhoods. Secondly, a simulated annealing algorithm requires every state to be reachable (i.e. any solution can be reached by any other solution after a finite number of iterations of moves in the defined neighbourhood) (Dowsland, 1995). However, in the simulated annealing hyper-heuristic, the low-level heuristics do not necessarily satisfy this requirement as long as there is a combination of these heuristics that can make each solution reachable. This is very useful when we have several possible heuristics or operators that can transfer the state of the current solution but these operators alone are not able to generate a neighbourhood that satisfies the reachability condition. For example, when dealing with the bin packing problem, a neighbour solution can be created by "*interchanging two (random) items of two (random) bins*". However, using this heuristic alone cannot guarantee to reach every other solution (from the current solution) within a limited number of executions. Meanwhile, although the neighbourhood defined by another operator "*shifting a random item from one random bin to another random bin*" satisfies the reachability condition, the local search algorithms using this operator alone generally perform very badly. More discussion on this point is given in section 4.3.1.

Previous research has shown that even if the neighbourhoods satisfy the reachability condition, the simulated annealing algorithm for each of those neighbourhoods has a different performance on different problems (Tian et al., 1999). A neighbourhood that could obtain promising results on a problem might perform very badly on another problem with a similar solution space. Therefore, defining and selecting a good neighbourhood for simulated annealing is very challenging and often time-consuming. The users (or researchers) have to have knowledge and expertise of the problem domain and often need to conduct a series of experiments.

However, in a simulated annealing hyper-heuristic framework, the system only requires a set of low level heuristics or rules. These heuristics could be simple operations used when solving real-world problems. For the shelf space allocation problem addressed in this thesis, the low-level heuristics could add or delete an item from a shelf, or replace one item with another on a shelf or interchange two items from two shelves. These heuristics could be very simple and straightforward and could also have some intelligent elements such that the current solution is transferred to a promising direction rather than being purely random. Combining these simple heuristics may produce much better results than by running them alone. The individual heuristics do not necessarily satisfy the reachability requirements as long as a combination of these heuristics does. We will demonstrate this point in the next section and in chapters 5 and 7 via three different applications.

4.3 An Application of Simulated Annealing Hyper-Heuristic to the One-dimensional Packing Problem

The aim of this section is to test the performance of the simulated annealing hyper-heuristics described in figure 4-1. We shall test them on a well-known NP-Hard problem: one-dimensional bin packing. The reason for choosing this problem is two-fold: the bin packing problem has been intensively studied in the literature with a large number of benchmark data sets being available; secondly, the bin-packing problem is closely related to the shelf space allocation problem, which is the main focus of this thesis.

4.3.1 Introduction

The bin packing problem (see section 2.3.1) is a well-known NP-Hard combinatorial optimisation problem. One of the most successful algorithms for bin packing is MTP (Martello-Toth Procedure), a branch and bound based exact method originally proposed in (Martello and Toth, 1990b). Some other exact methods have also been proposed by (Scholl et al., 1997; Valerio de Carvalho, 1999; Belov and Scheithauer, 2004). Heuristic based approaches have also been used. Apart from some well-known constructive heuristics, which will be discussed in the next section, meta-heuristic approaches have also been applied to the bin packing problem. Falkenauer (Falkenauer, 1996) introduced a grouping genetic algorithm for the problem. The algorithm was hybridised with a local search procedure and tested on a set of benchmark problems. The algorithm was shown to be superior to MTP. Fleszar and Hindi (Fleszar and Hindi, 2002) combined a minimal bin slack heuristic with a variable neighbourhood search method. The algorithm obtained better results by achieving 1329 optimal solutions from 1370 benchmark problems.

The objective of the bin packing problem is to minimise the number of bins required. The objective function itself is very simple and easy to calculate. However, the landscape of the responding search space is extremely “unfriendly”. A very small number of optimal solutions are lost in a big “flat” search space where a large number of solutions correspond to the same objective value. The majority of neighbourhood moves generate solutions with the same objective value. This makes the bin packing problem very difficult because the objective function is not able to “guide” the search in promising directions and the search proceeds like a boat floating in a dark sea without knowing which way to go. Therefore, all the meta-heuristic approaches in (Falkenauer, 1996; Scholl et al., 1997; Fleszar and Hindi, 2002) used the transformed objective functions that could provide a more friendly search space. However, as pointed out by (Falkenauer, 1998), using a transformed objective function could result in a problem which is different from the original one.

For example, suppose a solution A is better than the solution B if measured by a transformed objective function. It is possible that A is worse than B if the solution quality is measured by the original objective function. Therefore, developing a new form of objective function requires a certain amount of expertise and experience and often involves some experimentation. This will inevitably undermine the generality and adaptability of the algorithm.

Although several meta-heuristics have been applied to the bin-packing problem, with promising results being produced, the application of simulated annealing to the one dimensional bin packing problems is rare. The only application we could find dates from 1994 (Rao and Iyengar, 1994) and was used to solve a variant version of bin packing whose search space is very different from the general bin packing problem. Usually, a conventional simulated annealing algorithm samples the candidate solution uniformly from a single neighbourhood structure. The difficulty of applying SA to bin packing probably comes from the fact that the objective function of the bin packing problem (the number of the bins occupied) is not sensitive to the general neighbourhood moves (Falkenauer, 1996). For instance, two general neighbourhood moves, interchanging two items between two bins and shifting an item from one bin to another, are usually not able to change the objective function. Employing some elaborate moves (e.g. moving several items simultaneously in a predefined way) could damage the reachability of the neighbourhood and prematurely lead the search into a local optimum. In fact, as will be seen later in this chapter, improving the bin packing solution really needs the cooperation of several types of neighbourhood moves rather than executing single type of moves. However, a general simulated annealing algorithm only allows a single neighbourhood structure with the requirement of reachability, which handicaps the application of SA in problems such as bin packing. Therefore, in this application we propose to utilise several heuristics under the framework of the simulated annealing hyper-heuristics (see figure 4-1), rather than just using a single neighbourhood. The heuristics used to transfer the state of the solution do not necessarily satisfy the reachability requirement as required by a neighbourhood definition. The heuristics we use could have some intelligent elements such that the current solution is transferred to the promising directions rather than being purely random.

4.3.2 Bin packing constructive heuristics

Being NP-Hard, one dimensional bin packing problems have been solved by several constructive heuristics.

4.3.2.1 Polynomial heuristics

There are several well-known constructive heuristics with polynomial time complexity. Assume the items are sorted by descending size:

- **Next-Fit-Decreasing (NFD)**: starting from the first item, this heuristic repeatedly packs an item in the current bin. If there is insufficient capacity, the item is packed into a new bin which is now considered to be the current bin. The procedure finishes once all items are packed.
- **First-Fit-Decreasing (FFD)**: in this heuristic, the items are repeatedly packed into the bin which has the smallest bin index but sufficient capacity. A new bin is introduced if there is no bin with sufficient capacity.
- **Best-Fit-Decreasing (BFD)**: this heuristic repeatedly packs an item into the bin which has the smallest, but sufficient, capacity. If no such bin is available, a new bin is opened.

- **Worst-Fit-Decreasing (WFD)**: this heuristic is similar to BFD, except that the item is packed into the bin with the largest residual capacity.

All of these heuristics have polynomial time complexity. However, FFD and BFD are superior to NFD and WFD in terms of the worst-case performance (Coffman et al., 1997). A better heuristic, Best-2-Fit (B2F), was also discussed in (Coffman et al., 1997), which executes FFD until a bin is filled. It then tries to exchange the smallest item in the current bin with two smaller items such that the residual capacity of the current bin is as small as possible. Scholl et al. (Scholl et al., 1997) further improved this heuristic by combining a *bin-oriented* FFD heuristic with B2F.

4.3.2.2 Minimal bin slack heuristic (MBS)

Gupta and Ho (Gupta and Ho, 1999) proposed another bin-oriented heuristic, which they called Minimum Bin Slack (MBS). This heuristic is different from item-oriented heuristics in that the packing process is carried out bin-by-bin rather than item-by-item, using a procedure called *MBSOnePacking*. At each iteration, instead of packing the items one by one based on certain rules, the *MBSOnePacking* procedure searches for a group of items (from all the unpacked items) that could fill a bin with minimal slack (i.e. the smallest residual capacity) and packs this group of items into a new bin. The MBS heuristic repeatedly calls *MBSOnePacking* procedure until all items are packed. A recursive version of *MBSOnePacking* was illustrated in (Fleszar and Hindi, 2002) and is shown in figure 4-2, where $\Pi = \{\pi_1, \pi_2, ..., \pi_{n'}\}$ is an item vector which contains all unpacked items and is sorted by size in descending order. *n'* is the number of the items in the vector Π. *c* is the bin capacity and τ_i is the size of the item π_i. P_b is the set of items in the best packing found so far and P_c is the set of items in the current packing. s_b (respectively s_c) is the slack (i.e. residual capacity) of P_b (respectively P_c). The procedure stops when either a combination with zero slack (i.e. no residual capacity left) is found or all item combinations have been explored.

```
Initialise: $c$, $n'$, $\Pi$, $q$=1, $P_b = P_c$ = Ø, $s_b = s_c = c$;
void MBSOnePacking ($q$)
{
    For (int $r = q$; $r<n'$; $r$++)
    {
        $i = \pi_r$;
        If $\tau_i \le s_c$
        {
            $P_c = P_c \cup \pi_r$;
            Update $s_c$;
            MBSOnePacking($q$+1);
            $P_c = P_c \setminus \pi_r$;
            Update $s_c$;
            If ($s_c = 0$) Exit;
        }
    }
    If ($s_b > s_c$) $P_b = P_c$, Update $s_b$;
}
```

Figure 4-2: The pseudo-code of MBSOnePacking procedure
(Source: Fleszar and Hindi, 2002)

Fleszar and Hindi (Fleszar and Hindi, 2002) presented a variant of this procedure (denoted by MBS'), which always packs the first item of the vector Π into the current packing. The modified algorithm gives similar solution quality but shorter computation time in most instances. Because MBS' is only slightly different from MBS, we will not distinguish them in this thesis. The MBS heuristic has shown to be superior to FFD, BFD, B2F and FFD-B2F in terms of the solution quality and is able to solve the problem to optimality where the optimal solution is two bins. It is especially efficient when the optimal solution requires the majority of bins to be fully filled. The worst time complexity of this algorithm is 2^n, where n is the number of items. However, the experimental results have shown it to be very efficient for most problem instances (Gupta and Ho, 1999; Fleszar and Hindi, 2002).

4.3.2.3 Relaxed minimum bin slack (R_MBS)

Despite the efficiency of the MBS algorithm, there are instances (when no combination of items can be found that exactly fills a bin) for which the algorithm could carry out an exhaustive exploration of different combinations (Gupta and Ho, 1999; Fleszar and Hindi, 2002). Therefore, even for some moderately sized problems, the computational time can still be very high. To solve this problem, the MBS stopping condition $s_c = 0$ in figure 4-2 can be relaxed by allowing a positive slack value ($s_c \leq slackValue$). Actually it is not necessary to enforce zero slack in the *MBSOnePacking* procedure because in many cases there is always residual capacity in the optimal solution. Fleszar and Hindi (Fleszar and Hindi, 2002) experimented with several slack values and took the one that gave the best results. In this research, we let $slackValue = \min(\tau_{\min} - 1, \left\lceil c - \sum_{i=1}^{n'} \tau_i / U_1 \right\rceil)$, where $\tau_{\min}$ is the size of the smallest item in the vector Π and U_1 is the optimal solution or best known lower bound if the optimal solution is unknown. The algorithm results in the same solution as from FFD when $slackValue = \tau_{\min} - 1$.

4.3.2.4 Time bounded relaxed MBS (TBR_MBS)

From some preliminary experiments, we found that when the items' sizes are drawn from a close range and the average number of items in a bin increases, then it becomes more and more difficult for MBS to find the optimal packing even if it is relaxed by allowing a positive slack value. In this case, R_MBS can still be computationally expensive. To solve this problem, we propose a time limit for each

MBSOnePacking procedure such that the algorithm does not exceed a given time limit. Note that the TBR_MBS procedure should not exit when more items can be added to the current packing even though the time bound is exceeded. Therefore, although enforcing a time limit to *MBSOnePacking* procedure can cause deterioration in its performance, it could still produce a packing at least as good as the *bin-oriented* FFD heuristic.

4.3.3 Applying hyper-heuristics to the one-dimensional bin packing problem

To obtain an initial evaluation of the effectiveness of a type of simulated annealing based hyper-heuristic, three types of hyper-heuristics (denoted by CFHH, SAHH and CFSAHH respectively) are distinguished. CFHH is the choice function based hyper-heuristic mainly studied in (Soubeiga, 2003). SAHH is the simulated annealing hyper-heuristic which randomly selects a low-level heuristic at each iteration but the selected heuristic is only accepted if it satisfies the SA acceptance criterion (refer to figure 4-1). However in CFSAHH, both the choice function based heuristic selection mechanism and SA acceptance criterion are employed. To implement these hyper-heuristics, several parameters need to be set. In this investigation, the choice function parameters (α, β and δ) were set by the same methods used in (Soubeiga, 2003). The rest of the parameters were set as follows:

4.3.3.1 Initial solution

The initial solution is constructed using the time bounded relaxed MBS heuristic described in section 4.3.2.4. The time limit is set to 0.2 seconds based on preliminary experiments. In order to test the generality and adaptability of the algorithm, the original objective function (i.e. the number of used bins) was used rather than using some transformed objective functions as in (Falkenauer, 1996; Scholl et al., 1997; Fleszar and Hindi, 2002).

4.3.3.2 SA parameters

After the preliminary experiments, the temperature was initially set to $t_s = 0.3 f(s_0)$ ($f(s_0)$ is the objective value of the initial solution s_0) and then repeatedly reduced according to Lundy and Mees's cooling schedule (Lundy and Mees, 1986) $t \rightarrow t/(1+\beta t)$ until the temperature drops to its stopping temperature $t_f = 0.1$. At each temperature, only one iteration is executed. The parameter β can be calculated by (refer to (Lundy and Mees, 1986)):

$$\beta = (t_s - t_f)/K \cdot t_s \cdot t_f \;=\; (t_s - t_f) \cdot T_{average} /(T_{allowed} \cdot t_s \cdot t_f) \qquad (4\text{-}1)$$

where $T_{allowed}$ is the total CPU time allowed by the user and $T_{average}$ is the average time spent for one iteration. Therefore, the total number of iterations for the annealing is $K = T_{allowed} / T_{average}$. The algorithm stops either when the temperature reaches the stopping temperature or the lower bound of the solution[1] is reached.

4.3.3.3 Low-level heuristics

As showed in figure 4.1, the implementation of the simulated annealing hyper-heuristic requires a set of problem-specific low-level heuristics. A total of five low-level heuristics are used, as follows:

H1 **Exchange largestBin_largestItem.** This heuristic selects the largest item from the bin with the largest residual capacity and exchanges this item with another smaller item (or several items whose capacity sum is smaller) from

[1] We use the lower bound published on the website http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm and in the paper (Falkenauer, 1996).

another randomly selected non-fully-filled bin. The idea behind this heuristic is to transfer smaller residual capacity from a random bin to a bin with the largest residual capacity such that this bin can be emptied by other heuristic(s).

H2 **Exchange randomBin_largestItem.** This heuristic is similar to H1 except that the exchange is carried out between two randomly selected non-fully-filled bins.

H3 **Shift**. This heuristic selects each item from the bin with the largest residual capacity and tries to shift them to the rest of the bins using the BFD heuristic (see section 4.3.2.1).

H4 **Split.** H1, H2 and H3 all operate on non-fully-filled bins. However, in some cases, a fully-filled bin may contain too many small items such that it is impossible to transfer to the optimal solution using H1, H2 and H3 because of the difficulty in packing large items. Hence this heuristic is designed to solve this problem. Once the number of the items in a bin is found to have exceeded the average number of items of other bins, this heuristic transfers half of the items, selected at random, into a new bin.

H5 **BestPacking.** This heuristic firstly selects the biggest item from a probabilistically selected bin. The TBR_MBS heuristic is then used to search a group of items (called one packing) that contains this item and considers all the other items (the sequence of these items in the vector Π is sorted by the residual capacity of the corresponding bins with tie broken arbitrarily). All the items that appeared in the packing found by TBR_MBS are then transferred into a new bin. The time limit is set to 0.2 second based on preliminary experiments. The probability of selecting a bin is calculated by $\psi = \dfrac{resCap_j}{\sum resCap_j}$, where *resCap*$_j$ is the residual capacity of the bin *j*. Hence the selection is in favour of the bins with the larger residual capacity. The bins with zero residual capacity will not be selected because they are already packed well.

Note that all of those heuristics will return feasible solutions (the incumbent solution is returned if the new solution is infeasible). All of those heuristics are straightforward and easy to implement. Repeatedly applying them alone would either lead to a local optimal solution (e.g. H1, H2 and H3) or make the solution gradually worse (e.g. H4). However, when allowed to combine the heuristics, the algorithm may be able to transform the state of the solution toward promising directions. For example, running heuristic H1, H2 could repeatedly transfer sporadic residual bin capacity to the bins with larger residual capacity. After applying H1 and H2 for a while, the current solution might have been transformed into such a state that the bin with the largest residual capacity only has one or two small items. In this state, it might be helpful to call heuristic H3 such that these small items are shifted into other bins and hence the total number of used bins is decreased. These three heuristics only operate on the bins that have residual capacity. Heuristic H4 is useful when some bins contain too many small items. H5 is based on the time bounded relaxed MBS. This heuristic is very useful when the optimal solution contains many bins which are almost full (Gupta and Ho, 1999). Note that heuristics H1 and H2 are normally not able to change the objective, while heuristic H3 is an objective improving heuristic and heuristic H4 is objective non-improving. Heuristic H5 could undermine or improve the objective. All these heuristics will be managed by the hyper-heuristics.

4.3.3.4 Benchmark problems

Two sources of benchmark problems are available for the one-dimensional bin packing problem. One of them is from OR-Library (http://www.brunel.ac.uk/depts/ma/research/jeb/orlib/binpackinfo.html), originally created and studied by Falkenauer (Falkenauer, 1996) and now maintained by John Beasley at Brunel University. This data set consists of two classes of problems: uniform and triplet. In the uniform class, the number of items is 120, 250, 500 and 1000 respectively and their sizes are uniformly distributed in the range of [20,100]. The bin capacity is 150. We shall denote these by FAL_U120, FAL_U250, FAL_U500 and FAL_U1000 respectively. There are 20 instances for each problem size and hence 80 problem instances in total. In the triplet class, the bin capacity is 1000 and the item sizes are deliberately generated such that, in the optimal solution, every bin contains exactly three items (one “big” and two “small” items) without any residual capacity. The number of the items is 60, 120, 249 and 501 (denoted by FAL_T60, FAL_T120, FAL_T249 and FAL_T501 respectively) and each of them contains 20 instances. This class of data set is claimed to be more difficult because of the fact that no residual capacity is allowed in any bin in the optimal solution.

The second source was generated and studied by Scholl et al. (Scholl et al., 1997) and is available at http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm. It contains three sets (denoted by SCH_Set1, SCH_Set2 and SCH_Set3 respectively). The parameters to create SCH_Set1 and SCH_Set2 include the number of the items (ranging from 50 to 500), bin capacity and the ranges that the items’ sizes are drawn from. SCH_Set1 consists of 720 problem instances and the expected average number of items per bin is no larger than three. However, in SCH_Set2, the average number of items per bin varies from three, five, seven to nine. The data set SCH_Set3 are considered harder problem instances because the item size is drawn from a very large range such that no two items have the same size. Only 10 instances are included in this set.

4.3.3.5 Computational results

The algorithms were coded in Microsoft Visual C++ version 6.0 and all experiments were run on a PC Pentium IV 1.8GHZ with 256MB RAM running Microsoft Windows 2000 professional Version 5. For a fair comparison, all three hyper-heuristics (CFHH, SAHH, CFSAHH) were run 10 times for every instance, using a different random seed each time. For each run, 200 seconds computation time was allowed. To compare the performance of the different algorithms, the following symbols are used:

- **#num**: the number of instances in the given data sets.
- **#opt**: the number of instances for which the given algorithm finds a solution with the lower bound objective value (i.e. the algorithm has solved those instances optimally). For the three hyper-heuristics, the average values over 10 runs were reported. For the other meta-heuristics approaches (GGA, BISON and VNS), single run results were used due to no average results being available.
- **av. abs**.: the average absolute deviation from the optimality or the best known lower bound if the optimal solution is not known.
- **max abs**.: maximal absolute deviation from the optimal solution or the best known lower bound if the optimal solution is not known.
- **av. cpu (s)**: average CPU time spent for the given data sets (in seconds).

Table 4-1 presents a comparison of MBS based heuristics and three hyper-heuristics. MBS has been shown to be superior to the well-known bin packing heuristics (FFD and BFD) in terms of solution quality (Gupta and Ho, 1999). However, as a heuristic with exponential time complexity, in some cases, MBS can be computationally expensive. Time bounded relaxed MBS was designed to solve this problem. Time bounded relaxed MBS is guaranteed to finish a packing within the given time limit, with the solution quality at least as good as FFD's. This is supported by the results from table 4-1 where time bounded relaxed MBS costs less CPU time than MBS when dealing with data sets SCH_Set2, SCH_Set3, for which it is very difficult to find a set of items that can exactly fill a bin. For the data sets SCH_Set2, time bounded relaxed MBS is about 7 times faster than MBS and for the data set SCH_Set3, time bounded relaxed MBS is more than 3 times faster than MBS. In terms of solution quality, time bounded relaxed MBS achieves more lower bounds than MBS for most of the data sets. It is only beaten by MBS in the data sets FAL_U1000, where, however, time bounded relaxed MBS has a smaller maximal absolute deviation than MBS. Across 1370 benchmark instances, time bounded relaxed MBS reached lower bounds on 28 more instances than MBS.

Among three hyper-heuristics, table 4-1 shows that all three hyper-heuristics have improved initial solutions. CFSAHH performed better than CFHH with around 20 more instances being solved to their lower bounds. Meanwhile, it also reduced the average and maximal absolute deviation. This shows that for the bin packing problem, the performance of the choice function based hyper-heuristic can be improved by introducing a SA acceptance criterion. However, both CFHH and CFSAHH were outperformed by SAHH which randomly selects a low-level heuristic at each iteration. Compared with CFSAHH, SAHH solved, on average, 19.9 more instances to the lower bounds with much smaller average and maximal absolute deviation. Overall, SAHH has solved more than 98% (around 1343 out of 1370) of the problem instances to their lower bounds. With those that were not solved to the lower bound, SAHH could find a solution that is only one bin away from it. In CFSAHH, the deterministic heuristic selection method (guided by the choice function) seems not to be suitable in this case. The choice function seems too sensitive to the CPU time consumed by each heuristic. If a heuristic happens to improve the solution in a very short time, the weight of this heuristic can be very large such that this heuristic would dominate the search for a very long period, without giving enough opportunities for the other low-level heuristics to improve their weights. In SAHH, each heuristic has an equal opportunity to be selected. However, the heuristic moves can only be accepted according to the SA acceptance criteria.

Table 4-2 shows a comparison of three hyper-heuristics with a hybrid grouping genetic algorithm (GGA) (Falkenauer, 1996), BISON TL=1000 (a hybrid algorithm which combines a tabu search strategy with a branch and bound algorithm (Scholl et al., 1997)) and a variable neighbourhood search algorithm (VNS) (Fleszar and Hindi, 2002). The computational results of these algorithms are taken from the relevant papers. We can see that for the first data sets, SAHH performed better than the hybrid GGA and slightly worse than VNS. The hybrid GGA failed to solve 6 instances to their lower bounds while SAHH only failed on 2.4 instances on average. VNS solved all instances except one in the data sets FAL5000. In the second data sets, SAHH performed better, in terms of solution quality, than both BISON and VNS. SAHH solved around 12 more instances than BISON and 15 more instances than VNS. In terms of computation time, SAHH is faster than the hybrid GGA and BISON while slower than VNS (for instance, grouping GA took an average of 118 minutes to solve problem instances in set FAL_U1000 on a R4000 Silicon Graphics

Table 4-1: Computational results of MBS based heuristics and hyper-heuristics

		MBS				**TBR_MBS**				**CFHH**				**SAHH**				**CFSAHH**			
Data Sets	**#num**	**#opt**	**av. abs.**	**max abs.**	**av. cpu(s)**	**#opt**	**av. abs.**	**max abs.**	**av. cpu(s)**	**#opt**	**av. abs.**	**max abs.**	**av. cpu(s)**	**#opt**	**av. abs.**	**max abs.**	**av. cpu(s)**	**#opt**	**av. abs.**	**max abs.**	**av. cpu(s)**
FAL_U120	20	11	0.45	1	0.01	17	0.15	1	0.01	19.6	0.02	1	5.59	**20.0**	0.00	0	0.40	19.9	0.01	1	1.26
FAL_U250	20	12	0.45	2	0.05	13	0.40	2	0.04	17.9	0.11	1	23.83	**18.6**	0.07	1	22.56	18.5	0.08	1	21.05
FAL_U500	20	11	0.60	2	0.15	11	0.55	2	0.13	18.7	0.07	1	18.56	**19.0**	0.05	1	20.09	18.5	0.08	1	23.78
FAL_U1000	20	7	0.80	3	0.51	5	0.85	2	0.43	19.8	0.01	1	11.55	**20.0**	0.00	0	7.78	18.2	0.09	1	12.66
FAL_T60	20	0	1.00	1	0.01	0	1.00	1	0.01	16.5	0.18	1	37.86	**20.0**	0.00	0	3.19	17.4	0.13	1	42.26
FAL_T120	20	0	1.00	1	0.05	0	1.00	1	0.05	19.6	0.02	1	12.74	**20.0**	0.00	0	9.01	18.3	0.09	1	22.44
FAL_T249	20	0	1.80	3	0.33	0	1.80	3	0.32	18.4	0.09	2	32.76	**20.0**	0.00	0	9.70	19.3	0.04	1	12.25
FAL_T501	20	0	3.80	7	1.78	0	3.80	7	1.76	19.0	0.10	4	47.74	**20.0**	0.00	0	26.14	18.9	0.06	2	39.91
SCH_Set1	720	633	0.14	3	0.03	646	0.12	3	0.03	688.6	0.04	3	10.06	**703.3**	0.02	1	8.04	700.2	0.03	3	8.48
SCH_Set2	480	381	0.34	9	3.49	391	0.32	9	0.52	462.4	0.04	5	8.23	**474.2**	0.01	1	5.10	468.1	0.03	3	8.00
SCH_Set3	10	0	3.30	4	15.46	0	2.30	3	4.20	2.2	0.89	3	167.62	**7.9**	0.21	1	96.58	5.8	0.45	2	153.40
All	1370	1055	1.24	9	1.99	1083	1.12	9	0.68	1302.7	0.14	5	34.23	**1343.0**	0.03	1	18.96	1323.1	0.10	3	31.41

Table 4-2: A comparison with other meta-heuristics

		Hybrid GGA		BISON TL=1000		VNS		CFHH		SAHH		CFSAHH	
Data Sets	**#num**	**#opt**	**max abs.**	**#opt**	**max abs.**	**#opt**	**max abs.**	**#opt**	**max abs.**	**#opt**	**max abs.**	**#opt**	**max abs.**
FAL_U120	20	18	1	--	--	**20**	0	19.6	1	**20.0**	0	19.9	1
FAL_U250	20	18	1	--	--	**19**	1	17.9	1	18.6	1	18.5	1
FAL_U500	20	20	0	--	--	**20**	0	18.7	1	19.0	1	18.5	1
FAL_U1000	20	20	0	--	--	**20**	0	19.8	1	**20.0**	0	18.2	1
FAL_T60	20	18	1	--	--	**20**	0	16.5	1	**20.0**	0	17.4	1
FAL_T120	20	20	0	--	--	**20**	0	19.6	1	**20.0**	0	18.3	1
FAL_T249	20	20	0	--	--	**20**	0	18.4	2	**20.0**	0	19.3	1
FAL_T501	20	20	0	--	--	**20**	0	19.0	4	**20.0**	0	18.9	2
SCH_Set1	720	--	--	679	2	694	2	688.6	3	**703.3**	1	700.2	3
SCH_Set2	480	--	--	473	1	474	1	462.4	5	**474.2**	1	468.1	3
SCH_Set3	10	--	--	2	1	2	1	2.2	3	**7.9**	1	5.8	2
All	1370	--	--	--	--	1329	2	1302.7	5	**1343.0**	1	1323.1	3

workstation under IRIX 5.1, compared with 7.78 seconds by SAHH. VNS is faster as it took an average of 0.07 seconds on a PC with Pentium II 400MHz CPU, running on Windows NT4.0). The slower speed of SAHH is partially due to the usage of the two-layer data structure (see figure 4-1) which requires additional time to make solution copies if low-level heuristics fail to generate a feasible solution. However, this hyper-heuristic is a more general problem solver. In the next chapter, the reader will see that the algorithm can also be easily applied to a shelf space allocation problem, with high quality solutions being produced. When adapting this algorithm to a different problem, a user only needs to input a set of problem-specific low-level heuristics and an objective function. The algorithm is able to solve the problem with very good quality solutions in reasonable computational time. However, it could take much more work when adapting a genetic algorithm or variable neighbourhood search algorithm to a different problem.

4.4 Summary and Remarks

This chapter was concerned with the acceptance criteria that exist in the current hyper-heuristic algorithms. In the current hyper-heuristic algorithms, two general acceptance criteria that are used to decide whether a given heuristic move is accepted are *improvement-only* and *all-moves*. The first criterion only accepts those heuristic moves that can improve the current solution while the second criterion accepts all moves regardless of whether they are better solutions or not. These acceptance criteria may either lead the search to getting trapped into local optima or result in a random search. In this thesis, a simulated annealing based acceptance criterion was proposed and incorporated into the hyper-heuristics. As an intelligent balance of these two acceptance criteria, the simulated annealing acceptance criterion accepts all objective-improving heuristic moves as well as some non-improving heuristic moves.

The proposed hyper-heuristic has firstly applied to the well-known bin packing problem. The reason is that it is not easy to compare the performance of the proposed hyper-heuristics with other meta-heuristic approaches on shelf space allocation problems due to the lack of benchmark data in the literature. However, as a problem closely related with the shelf space allocation problem, the bin packing problem has been the subject of considerable research with several benchmark problem results existing in the literature, which will allow us to make an easier comparison between the proposed simulated annealing hyper-heuristics and other meta-heuristic algorithms.

In this application, we have shown that, for the bin packing problem, although introducing the simulated annealing acceptance criterion can improve the performance of the choice function based hyper-heuristics, the best algorithm turns out to be a simulated annealing hyper-heuristic (SAHH) which randomly selects between low-level heuristics. The deterministic heuristic selection method in the choice function seems not to be well suited to the stochastic acceptance criterion of the SA.

Of the 1370 tested one-dimensional bin packing benchmark problem instances, SAHH solved 1343 of them to the lower bounds on average within reasonable computation time. For those that are not solved to the lower bounds, the gap is only one bin. The algorithm is generally superior to other meta-heuristic approaches (GGA and VNS) in terms of solution quality although SAHH is slower than VNS.

In the next chapter, a general shelf space allocation problem is considered and a practical model is proposed. The simulated annealing hyper-heuristic algorithm proposed in this chapter is then adapted for the optimisation of this problem.

CHAPTER 5. OPTIMISATION OF A GENERAL SHELF SPACE ALLOCATION PROBLEM

5.1 Introduction

This chapter addresses a general shelf space allocation problem that has been the subject of the previous research on shelf space allocation problems. A simplified, while practical, model is proposed as an alternative to a complex model, which is not practical for the real-world practice. It is shown that this model is an extension of the multi-knapsack problem which is NP-Hard and cannot be solved by any polynomial-time bounded algorithm (assuming $\mathrm{P} \neq \mathrm{NP}$). As such, the optimal solution of the problem is generally unknown. To effectively measure the quality of a solution and compare the performance of different algorithms, this chapter derives an upper bound for the problem by using a two-stage relaxation. Due to the lack of real-world and benchmark data for this problem, twelve problem instances are randomly generated, with different problem sizes and space availability ratios.

This chapter investigates an emerging search technique, hyper-heuristics, when applied to shelf space allocation problems. A set of problem-specific low-level heuristics are designed and input into the hyper-heuristic framework. Those low-level heuristics are very similar to the heuristics used in the bin packing and knapsack problems. As shown in chapter 4, a simulated annealing based hyper-heuristic has produced competitive results for the bin packing problem, a problem which is closely related to the shelf space allocation problem. In this chapter, the simulated annealing based hyper-heuristic is adapted to a shelf space allocation problem concerned in this chapter. Its performance is evaluated by comparing with a conventional simulated annealing algorithm and several other types of hyper-heuristics. Experimental results from these algorithms on the twelve problem instances are compared and reported. The research work described in this chapter is mainly drawn from (Bai and Kendall, 2005b).

5.2 Model Formulation

As discussed in chapter 2, because different stores have different requirements and merchandise styles, it is difficult to develop a generic model that can represent all real-world shelf space allocation problems. For research purposes, this chapter studies a general shelf space allocation problem which has also been studied by several other researchers.

Suppose there are m shelves, with each shelf j having a capacity of T_j, and n types of items (or SKUs) need to be displayed on the shelves. Each type of item could have more than one facing being displayed on the shelf. The problem concerned in this research is the assignment of appropriate shelf space to every SKU of a given category in order to maximise the overall profit, whilst not violating the given constraints.

A few shelf space allocation optimisation models have been proposed with the aim of maximising products' aggregate profits. These models have been improved by integrating several factors, such as the inter-relationships among the products, product assortment, handling costs, stock holding costs and some other variables, such as price, advertising, promotions, etc (Corstjens and Doyle, 1981b; Zufryden, 1986; Borin et al., 1994; Urban, 1998). We firstly introduce a complex model.

5.2.1 A complex model

A central issue in the shelf space allocation problem is defining a demand function that reveals the relationship between the displayed shelf space and the amount of demand it can capture. A diminishing return polynomial function has been widely used by several researchers in the literature (Zufryden, 1986; Baker and Urban, 1988; Dreze et al., 1994; Urban, 2002) to describe the relationship between the displayed facings of an item and the demand of that item. Figure 5-1 presents an illustration of this function. It can be seen that the demand of an item is continuously increasing with the increase of facings allocated to the item. However the rate of the increase does slow down.

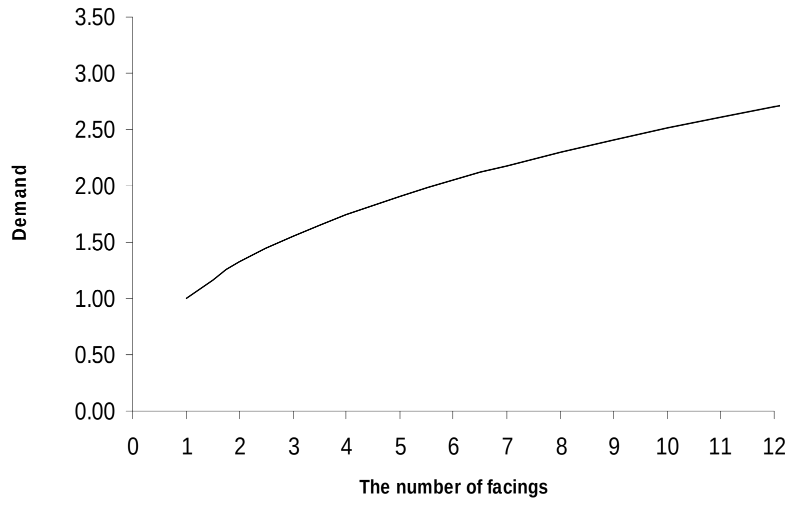


Figure 5-1: An illustration of demand rate with respect to the allocated shelf space

Comprehensive models were proposed in (Yang and Chen, 1999; Urban, 2002) which formulate the demand function in the following form:

$$D_i = \sum_{k=1}^{m} (\alpha_i x_{ik}^{\beta_{ik}} \prod_{\substack{u=1 \\ u \neq i}}^{n} x_i^{\gamma_{iu}} \prod_{t=1}^{R} y_{ti}^{\delta_{ti}}) \tag{5-1}$$

where

- n is the number of different types of items;
- m is the number of shelves;
- D_i is the demand function of item i;

- x_{ik} is the number of facings of item *i* allocated to shelf *k*.
- x_i is the total number of facings allocated to an item, i.e. $x_i = \sum_{k=1}^{m}(x_{ik})$;
- α_i is a scale parameter;
- β_{ik} is the space elasticity of item *i* on shelf *k*;
- γ_{iu} is the cross elasticity between item *i* and item *u*;
- y_{ti} is the t^{th} $(t = 1,...,R)$ market variable that can influence the demand. The possible market variables can be the price, advertising campaign, promotional manipulations, etc.;
- δ_{ti} is the elasticity value of *t*th market variable with respect to the demand value of item *i*.

The objective is to maximise the total profit which equals to the total gross profit from sales of all items deducting the aggregate costs to realise the sales.

$$\max \quad P = \sum_{i=1}^{n} g_i D_i - \sum_{i=1}^{n} \theta_i (D_i)^{\eta_i} \tag{5-2}$$

$$\text{subject to} \quad \sum_{i=1}^{n} l_i x_{ij} \le T_j \quad j = 1,...,m \tag{5-3}$$

$$L_i \le \sum_{j=1}^{m} x_{ij} \le U_i \qquad i = 1,...,n \tag{5-4}$$

$$x_{ij} \in \{0,1,\ 2,\ 3\ ...\} \quad i = 1,...,n \quad j = 1,...,m \tag{5-5}$$

where g_i is the unit revenue of item *i* and θ_i is the cost coefficient of item *i* and η_i is the cost elasticity of item *i* with respect to the sales volume. T_j is the capacity of the shelf *j* and l_i is the size of item *i*. Constraint (5-3) ensures that the shelf capacity constraints are satisfied. Constraint (5-4) confines the upper bound U_i and lower bound L_i of the number of facings of item *i* and constraint (5-5) are the integrality requirements of the shelf space allocation variables.

However, there are some drawbacks with this model. Firstly, the model includes many problem parameters, especially the $n \times n$ number of cross elasticities (γ_{iu}) for a problem with *n* items. It is a challenging task to get a reliable estimation of these values. Secondly, although the demand function (5-1) takes into consideration the shelf location impact, it will encourage a shelf space allocation decision which scatters facings of the same types of products among many shelves. By doing this, the demand of that item can be increased. This is due to the characteristics of polynomial function. For example, for a polynomial function $f(x) = x^{0.5}$, $f(2) \le f(1) + f(1)$. This means that with the demand function of (5-1), the demand can be increased by simply moving part of facings from the current shelf to another shelf even if this shelf has the same location quality to the previous one. This may not fit well with the real-world environment where displaying the same products together on a shelf may be able to attain greater demand than by displaying them separately. Finally, the objective function employed in the above model is complex and could be computationally expensive to calculate and optimise. Yang (Yang, 2001) proposed a simpler linear model by assuming that total net profit was linear with the number of facings of an item. However, this assumption is unrealistic for the real-world retail environment.

5.2.2 A simplified model

To simplify the problem, in this chapter, it is assumed that:

1) The shelf space constraints in depth and height are ignored. The problem is therefore a one-dimensional space allocation problem.

2) The retailer can prevent the occurrence of out-of-stock situations. All items are fully stocked.
3) The total profit of item *i* is proportional to its unit profit p_i (equivalent to η_i =1).
4) The cross elasticities between products are much smaller than the direct space elasticity and can be ignored.
5) The shelf-life of the products does not affect the demand of the products.

Note that in chapter 6, we will consider a shelf space allocation problem for fresh produce (that is, assumption 5, above, is removed), which generally have limited shelf-life and freshness is one of aspect that affects demand. Therefore, the last assumption does not hold for fresh produce.

Suppose a problem with *m* shelves and *n* items, each stock-keeping unit is defined by a five-tuple (l_i, p_i, β_i, L_i, U_i) where l_i (respectively, p_i, β_i, L_i, U_i) is the size (respectively profit, space elasticity, lower bounds, upper bounds) of item *i*. The capacity of shelf *j* is denoted by T_j. Based on the assumptions we discussed above, the problem can be expressed as:

$$\max \qquad P = \sum_{i=1}^{n} (p_i \alpha_i x_i^{\beta_i}) \tag{5-6}$$

$$\text{subject to} \quad \sum_{i=1}^{n} l_i x_{ij} \le T_j \quad j = 1,\ldots,m \tag{5-7}$$

$$L_i \le \sum_{j=1}^{m} x_{ij} \le U_i \qquad i = 1,\ldots,n; \tag{5-8}$$

$$x_{ij} \in \{0,1,\ 2,\ 3\ \ldots\} \quad i = 1,\ldots,n \quad j = 1,\ldots,m \tag{5-9}$$

The decision variables are x_{ij}, representing the number of facings of item *i* on shelf *j* and $x_i = \sum_{j=1}^{m} x_{ij}$ is the total number of facings of item *i*. α_i is a scale parameter and α_i >0. β_i is the space elasticity and $0 \le \beta_i \le 1$. The objective is to maximise the overall profit without violating the given constraints. The model is a non-linear, multi-constraint optimisation problem. If $\beta_i \to 1$, the model degenerates into a bounded multi-knapsack problem.

5.2.3 An upper bound of the model

As shelf space allocation problems cannot be solved to optimality in polynomial time (Borin et al., 1994), we usually do not know the optimal solution and hence cannot evaluate the quality of a given solution by comparing it with the optimal solution. Yang (Yang, 2001) compared his results with the optimal solution obtained by carrying out a complete enumeration. However, this method is only suitable for very small problem instances. For a shelf space allocation problem with *n* items (each item has an upper bound of facings *U*) and *m* shelves, it may require up to $U^{m \times n}$ iterations to find the optimal solution by using an exhaustive search. Even for a small problem instance: *n*=6, *m*=3, *U*=6, this could be computationally expensive. Another common method is to relax the problem to a simpler one whose optimal objective value is taken as an upper bound of the original problem. In this research a two-stage relaxation method was used, which can be described as follows:

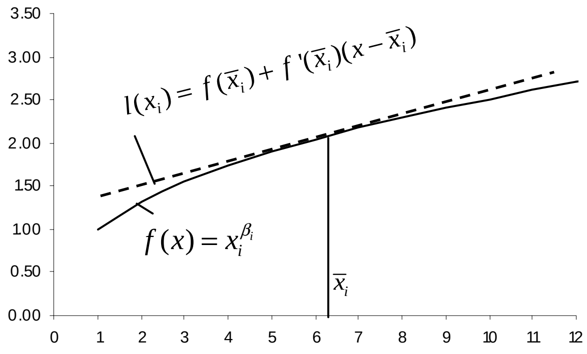


Figure 5-2: Approximate function $x_i^{\beta_i}$ with a linear function

Stage 1: The original non-linear model (5-6) is relaxed to a linear model. This is accomplished by applying a first order Taylor expansion to $x_i^{\beta_i}$ at the point $\overline{x}_i$

($L_i \le \overline{x}_i \le U_i, \overline{x} \in Z^+$) (as illustrated in figure 5-2). The model then becomes an integer programming (IP) problem:

$$\text{maximise} \quad P_{IP} = \sum_{i=1}^{n} p_i \alpha_i [\beta_i \overline{x}_i^{(\beta_i - 1)} (x_i - \overline{x}_i) + \overline{x}_i^{\beta_i}] \tag{5-10}$$

or

$$\text{maximise} \quad P_{IP} = \sum_{i=1}^{n} p_i \cdot \alpha_i \cdot l(x_i) \tag{5-11}$$

subject to the constraints (5-7), (5-8) and (5-9), where $l(x_i) = \beta_i \overline{x}_i^{(\beta_i - 1)} (x_i - \overline{x}_i) + \overline{x}_i^{\beta_i}$. Suppose $X^* = (x_1^*, x_2^*, ..., x_n^*)$ is the optimal solution for the original model (5-6) and P^* is its corresponding optimal objective value. P_{IP}^* is the optimal objective value for the IP model (5-11). From figure 5-2, we have:

$$\begin{aligned} P^* &= \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i} = \sum_{i=1}^{n} p_i \cdot \alpha_i \cdot l(x_i^*) - [\sum_{i=1}^{n} p_i \cdot \alpha_i \cdot l(x_i^*) - \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i}] \\ &\le P_{IP}^* - [\sum_{i=1}^{n} p_i \alpha_i \cdot l(x_i^*) - \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i}] \ \le P_{IP}^* \end{aligned} \tag{5-12}$$

Hence, the gap between P_{IP}^* and P^* is no less than:

$$\begin{aligned} G_1 &= \sum_{i=1}^{n} p_i \alpha_i \cdot l(x_i^*) - \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i} \\ &= \sum_{i=1}^{n} p_i \alpha_i [\overline{x}_i^{\beta_i} + (x_i^* - \overline{x}_i) \beta_i \overline{x}_i^{(\beta_i - 1)} - (x_i^*)^{\beta_i}] \end{aligned} \tag{5-13}$$

From equation (5-13), it can be seen that the closer $\overline{x}_i$ is to x_i^*, the smaller the gap is. In order to keep G_1 as a small value, we let $\overline{x}_i = x_i^{'}$ where $X^{'} = (x_1^{'}, x_2^{'}, ..., x_n^{'})$ is the best solution found by the algorithms (see section 5.4).

Stage 2: Based on the approximation from stage 1, the integer constraint (5-9) in the IP model is ignored and the model now becomes a linear programming (LP) model. The software “lp_solve” (a free LP software package) was used to obtain the optimal objective (denoted by P_{LP}^*) of this LP problem. We took this value as the relaxed upper bound of the shelf space allocation model (denoted by P^{ub}), i.e. $P^{ub} = P_{LP}^{*}$.

5.3 Optimisation of the Model

Several types of hyper-heuristics were applied to optimise the problem. The author specifically investigated the simulated annealing hyper-heuristic following its success in solving the bin packing problem. Hyper-heuristics are high level strategies which make calls to appropriate low-level heuristics in order to tackle different problems or problem instances with good quality solutions. However, in the simulated annealing hyper-heuristic, the algorithm is concerned with the acceptance criteria of each low-level heuristic call. Instead of accepting all low-level heuristic calls as other hyper-heuristic algorithms do, the simulated annealing hyper-heuristic only accepts some of the “deteriorating” low-level heuristic calls (a heuristic that results in an inferior solution), controlled by the simulated annealing Metropolis probability. From figure 4-1, it can be seen that, to implement the simulated annealing hyper-heuristic, one needs to generate an initial solution and define a set of problem-specific low-level heuristics. The parameters related to simulated annealing (e.g. initial and stopping temperature, temperature reduce scale, etc.) are supposed to be able to automatically tuned by the algorithm itself in order not to undermine the generality of the hyper-heuristics. The following subsections address these issues.

5.3.1 Low-level heuristics

Before describing the low-level heuristics which were used in the hyper-heuristics, we firstly define three order lists.

– P_{01}**: *item_contribution_list*:** item list ordered by $p_i \cdot \alpha_i / l_i$ decreasingly.

- P_{02} **: *item_length_list***: item list ordered by length l_i increasingly;
- S_0 **: *shelf_freelength_list*:** shelf list sorted by the current free shelf space decreasingly.

A total of twelve low-level heuristics were used in this application. They are categorised into four types: add product(s), delete product(s), swap and interchange. Note that each low-level heuristic only searches within the feasible region of the solution space. If a low-level heuristic cannot produce a new feasible solution, the current solution is returned.

- ***Add_random*:** this heuristic adds one facing of a random item to the first shelf of S_0. A maximum of five attempts are made if the heuristic fails to generate a feasible solution.
- ***Add_exact*:** this heuristic searches and adds one facing of the biggest possible item to all shelves (begins from the first shelf of S_0) until all shelves cannot be assigned any more items.
- ***Add_best_contribution*:** this heuristic repeatedly selects a shelf from S_0 (begins from the first shelf of S_0), repeatedly searches and adds as many facings of an item as possible from P_{01} (begins from the first item of P_{01}) until all shelves cannot be allocated any more items.
- ***Add_best_improvement*:** this heuristic selects the first shelf of S_0 and allocates one facing space to the item which gives the best improvement to the objective function.
- ***Delete_random*:** this heuristic deletes one facing of a random item from a random shelf. A maximum of five attempts are made if the heuristic fails to generate a feasible solution.

- ***Delete_least_contribution1*****:** this heuristic deletes one facing of the item with the least contribution value ($p_i \cdot \alpha_i / l_i$) from a random shelf.
- ***Delete_least_contribution2*****:** this heuristic deletes one facing of the item with the least contribution value from all shelves.
- ***Delete_least_improvement*****:** this heuristic deletes one facing of the item that causes the least decrease in the objective value from a random shelf.
- ***Swap_random*****:** this heuristic randomly deletes one facing of an item from a random shelf and adds as many possible facings of another randomly selected item. A maximum of five attempts are made if the heuristic fails to generate a feasible solution.
- ***Swap_best*****:** this heuristic repeatedly selects a shelf from S_0 , deletes one facing of the item with the lowest contribution value and adds one facing of another item with a higher/highest contribution value until the last shelf is swapped.
- ***Interchange_improvement*****:** this heuristic randomly selects two different items from two random shelves with non-zero residual capacity and interchanges one facing or multiple facings of the two items. The basic idea behind this heuristic is that the small free space can be transferred to the shelf with a larger free space so that another facing could be added to that shelf later.
- ***Interchange_random*****:** this heuristic selects two different items from two random shelves and exchanges one facing of the two items. A maximum of five attempts are made if the heuristic fails to generate a feasible solution.

5.3.2 Initial solution

Considering the similarity between the shelf space allocation problem and the knapsack problem, the initial solution was generated by a heuristic which was also used in the knapsack problem (Martello and Toth, 1990a). The pseudo-code of the heuristic is shown in figure 5-3.

```
Check if the total available space is large enough to satisfy the minimum
facing requirements for every item;
Allocate the space to every item in $P_{02}$ to meet the minimum facing
requirements;
Select the first shelf from $S_0$;
Do
    Select the first item from $P_{01}$;
    Do
        If free space is no less than the length of this item
            Allocate the maximum space to this item to meet its upper
            facing bound;
        Endif
        Select the next item from $P_{01}$;
    Loop until it reaches the last item in the list $P_{01}$;
    Select the next shelf from $S_0$;
Loop until it reaches the last shelf of $S_0$;
```

Figure 5-3: A greedy heuristic approach for the shelf space allocation

5.3.3 SA parameters

Typically, a simulated annealing algorithm has four parameters (except for the neighbourhood definition): initial temperature t_s, temperature reduction rate β, number of iterations at each temperature (*nrep*) and stopping condition(s). To test the generality of our simulated annealing hyper-heuristic algorithm, we used the same parameters that were employed in solving the bin packing problem (see section 4.3.3.2): the initial temperature was set at $t_s = 0.3f(s_0)$ where $f(s_0)$ is the objective value of initial solution s_0. The temperature is then reduced repeatedly according to the function $t \to t/(1+\beta t)$. At each temperature, only one repetitions was executed (i.e. *nrep*=1) and the algorithm stopped when the temperature dropped to 0.1. The temperature reduction rate β was calculated by $\beta = (t_s - t_f) \cdot T_{average} /(T_{allowed} \cdot t_s \cdot t_f)$ where $T_{allowed}$ is the total CPU time allowed by the user and $T_{allowed}$ is the average time spent for one iteration. In this application, the computation limit was set to 600 seconds (i.e. $T_{allowed}$ =600 seconds). Hence, the total number of repetition K can be calculated by $K = T_{allowed} / T_{average}$. Being consistent with chapter 4, we denote this version of the algorithm by SAHH.

Another version of the simulated annealing hyper-heuristic algorithm was implemented where the initial and stopping temperatures were calculated in a more intuitive way, similar to the method proposed in (Johnson et al., 1989; Johnson et al., 1991). At the beginning of the search, $K/100$ random solutions were sampled from the initial solution to approximately determine the maximal objective difference $\delta_{\max}$ and minimal objective difference $\delta_{\min}$ ($\delta_{\min} \neq 0$). The starting temperature was then set to a value such that about 85% of such inferior moves would be accepted. According to the Metropolis probability function, we have $t_s = -\delta_{\max} / \ln(0.85)$. Similarly, the stopping temperature was set to a value such that only about 1% of inferior moves would be accepted, i.e. $t_f = -\delta_{\min} / \ln(0.01)$. We denote this version of the algorithm by SAHH_adpt.

5.3.4 Other approaches

For the purpose of comparison, we also implemented the following heuristic and meta-heuristic approaches.

Choice function hyper-heuristics (CFHH)

In section 4.3, two choice function based hyper-heuristics (CFHH and CFSAHH) have been employed to optimise the one-dimensional bin packing problem. CFHH is the choice function based hyper-heuristic that was developed and studied in (Soubeiga, 2003). CFSAHH is an extended version of CFHH which incorporates a simulated annealing acceptance criterion. In this chapter, these two hyper-heuristics with the same parameter settings as chapter 4 were also applied to the shelf space allocation problem.

```
Initialise
    Assign appropriate initial weight $w(i)$ to each heuristic $i$;
    Set max_tabu_len, the maximal length of the tabu list, to an
    appropriate value;
    Generate an initial solution $s_0$;
Repeat
    Select the non-tabu low-level heuristic H* with the highest weight;
    Apply H* to the current solution $s$, resulting in a neighbour solution $s'$;
    If $f(s') - f(s) > 0$
        $w(H^*) = w(H^*)+1$;
    Else
        $w(H^*) = w(H^*)-1$;
        Push heuristic H* into the tabu list;
        If the maximal length of the tabu list is reached, release the first
        heuristic in the tabu list;
        If $f(s') - f(s) < 0$, release all heuristics in the tabu list except
        heuristic H*;
    Endif
    $s \leftarrow s'$;
Until stopping criteria are met.
```

Figure 5-4: The pseudo code of a tabu search based hyper-heuristic for a maximisation problem
Source: (Burke et al., 2003c)

Tabu search based hyper-heuristic (TSHH)

A recently developed tabu search based hyper-heuristic approach (Burke et al., 2003c) was also applied to the problem, abbreviated to TSHH. The main idea behind this algorithm is the incorporation of a tabu list in the heuristic selection mechanism that forbids the selection of some low-level heuristics at certain stages of the search.

For a maximisation problem with an objective function *f*(*x*). TSHH can be described by figure 5-4. Note that this tabu search hyper-heuristic differs from a general tabu search in that the tabu list stores low-level heuristics rather than solution attributes. Also note that once an inferior solution is generated by a low-level heuristic, all the heuristics in the tabu list are released because the current solution has been modified and the heuristics in the tabu list may now be useful.

In this application, the parameters were set as follows (these are the same as Burke et al., 2003c). Suppose a total of *k* low-level heuristics were used, the maximal length of tabu list was set to *k*/2. The upper bound and lower bound of weights for each low-level heuristic were set to *k* and 0 respectively. Once a heuristic's weight exceeded one of its boundaries, it was set to the corresponding boundary.

```
Initialise
    Assign appropriate initial weight $w(i)$ to each heuristic $i$;
    Set max_tabu_len, the maximal length of the tabu list, to an
    appropriate value;
    Set initial temperature $t_s$, stopping temperature $t_f$ and total iterations $K$.
    Generate an initial solution $s_0$, $t=t_s$;
Repeat
    Select the non-tabu low-level heuristic H* with the highest weight;
    Apply H* to the current solution $s$, resulting in a neighbour solution $s'$;
    If $f(s') - f(s) > 0$
        $w(H^*) = w(H^*)+1$;
        $s \leftarrow s'$;
    Else
        $w(H^*) = w(H^*)-1$;
        Push heuristic H* into the tabu list;
        If the maximal length of the tabu list is reached, release the first
        heuristic in the tabu list;
        If $f(s') - f(s) < 0$
            Release all heuristics in the tabu list except heuristic H*;
            If $\exp(\delta / t) > random(0,1)$  $s \leftarrow s'$;
        Else
            $s \leftarrow s'$;
        Endif
    Endif
    $t = t/(1+\beta t)$;
Until stopping criteria are met.
```

Figure 5-5: The pseudo code of a TSSAHH for a maximisation problem

Furthermore, a hybridised version of tabu search based hyper-heuristic was also implemented, denoted by TSSAHH, which adopts the simulated annealing acceptance criteria described in section 4.2.3. Figure 5-5 presents a detailed description of the algorithm. The parameters with regards to simulated annealing are the same as SAHH.

Random heuristic methods

Two simple random heuristic methods, RHOI (Random Heuristics Only Improving) and RHAM (Random Heuristics All Moves), were also applied to the problems. Both methods randomly call the low-level heuristics at each iteration. RHOI repeatedly selects a random low-level heuristic and applies it to the current solution until some stopping criteria are met (in this chapter, the stopping criterion is 600 seconds computational time), during which only those heuristics that can improve the objective function value are accepted. RHAM works in a similar way but all moves are accepted.

Simulated annealing algorithms

Two conventional simulated annealing algorithms, SA_swap and SA_interchange, were also applied to the problems. Both of the algorithms employ the same cooling schedule that was used in SAHH but utilise different neighbourhood structures. In SA_swap, the neighbourhood structure was defined by randomly swapping one facing of two different items (i.e. randomly select an item from a random shelf, delete one facing of this item from the shelf and add one facing of another randomly chosen item to the shelf). The neighbourhood in SA-interchange was generated by randomly selecting two different items from two random shelves, interchanging one facing of the two items, and then adding as many facings as possible of the item with the largest possible *item_contribution* value to the shelf that has the largest free space.

Note that if the above neighbourhood moves produced an infeasible solution, another attempt was made until a feasible move is generated.

5.4 Experimental Results

As there is no real-world data available (due to commercial confidentiality) and neither is there any benchmark data available from the literature, a number of simulated problems were generated as follows. The length of the products conforms to a uniform distribution between 25 and 60. The net profit p_i (i=1,…n) of the products were created by a normal distribution in the similar way to the one that was described in (Yang, 2001). The mean of p_i is uniformly drawn from the range [3, 3.5] and the ratio of the mean to standard deviation has a uniform distribution from 0.05 to 0.15. α_i, β_i, L_i, U_i and T_j have uniform distributions in the ranges of [1, 2], [0.1, 0.4], [2, 3], [7,10] and [300, 450] respectively. In the light of Yang's (Yang, 2001) experimental results which show that the problem size is a potential factor affecting algorithm performance, in this research, five problem instances with different problem sizes were generated to test this relationship. For simplicity, we shall call it data set S (denoted by S1, S2, S3, S4 and S5 respectively). We also took into account the influence of space availability in the performance of the algorithms. As the number of facings of each item has a lower bound and an upper bound, the available shelf space of a problem must be greater than a minimal space value to satisfy the lower bound of facings and should also not exceed a maximal space value to avoid the situation that the shelf space is sufficient enough such that all items can reach the upper bounds of facings and no optimisation is required. Two parameters, *r_min* and *r_max*, were introduced to describe the space availability. *r_min* represents the ratio of the minimal space to the available space and *r_max* is the ratio of the available space to the maximal space. Hence both *r_min* and *r_max* are in the range of [0, 1]. Seven problem instances (denoted by R1, R2, ..., R7 respectively) with different *r_min* and *r_max* values were generated to test the corresponding algorithms' performance.

Table 5-1 and 5-2 shows the problem sizes, space availability ratios and the upper bound values (P^{ub}) of these twelve instances.

All algorithms were coded in Microsoft Visual C++ version 6.0 and all experiments were run on a PC Pentium IV 1.8GHZ with 256MB RAM running Microsoft Windows 2000 professional Version 5. All algorithms started from the same initial solution produced by the greedy heuristic described in section 5.3.2 and we allowed 600 seconds computation time to give a fair comparison. All algorithms were run 30 times and their average objective value (P^h), minimal objective value (min) and maximal objective value (max) were observed. The corresponding standard deviations (stdev) were calculated and compared. The performance of the different algorithms was evaluated by the ratio of their average objective value (P^h) to the relaxed upper bound (P^{ub}).

Table 5-1: Five test problem instances with different sizes (data set S)

	S1	**S2**	**S3**	**S4**	**S5**
r_min* / *r_max	0.95 / 0.24	0.95 / 0.33	0.95 / 0.25	0.95 / 0.24	0.95 / 0.24
(*m*, *n*)	(5,20)	(12, 54)	(22, 60)	(30,80)	(40, 100)
P^{ub}	186.53	422.25	610.67	884.62	1077.376

Table 5-2: Seven test problem instances with different space availability ratios (data set R)

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
r_min*/*r_max	0.95 / 0.33	0.85 / 0.35	0.7 / 0.46	0.6 / 0.53	0.5 / 0.66	0.4 / 0.79	0.34 / 0.95
(*m*, *n*)	(12, 54)	(11, 48)	(15, 48)	(16, 48)	(17, 48)	(22, 48)	(29, 50)
P^{ub}	422.25	401.33	411.05	435.55	471.30	526.04	482.64

The first round of experiments was carried out on the data set R to test the performance of different algorithms under different shelf space availabilities. Figure 5-6, table 5-3a and 5-3b present the corresponding computational results. It can be seen that both SAHH, SAHH_adpt, CFHH and CFSAHH have greatly improved over the initial greedy heuristic. SA_swap also produced good quality solutions while SA_interchange performed much worse. This shows that the performance of the simple simulated annealing algorithm can be largely dependent on the neighbourhood structure. It can also be seen that SAHH and SAHH_adpt outperformed all other algorithms in all cases with surprisingly high quality solutions. Both SAHH and SAHH_adpt achieved over 99% of the upper bound for six problem

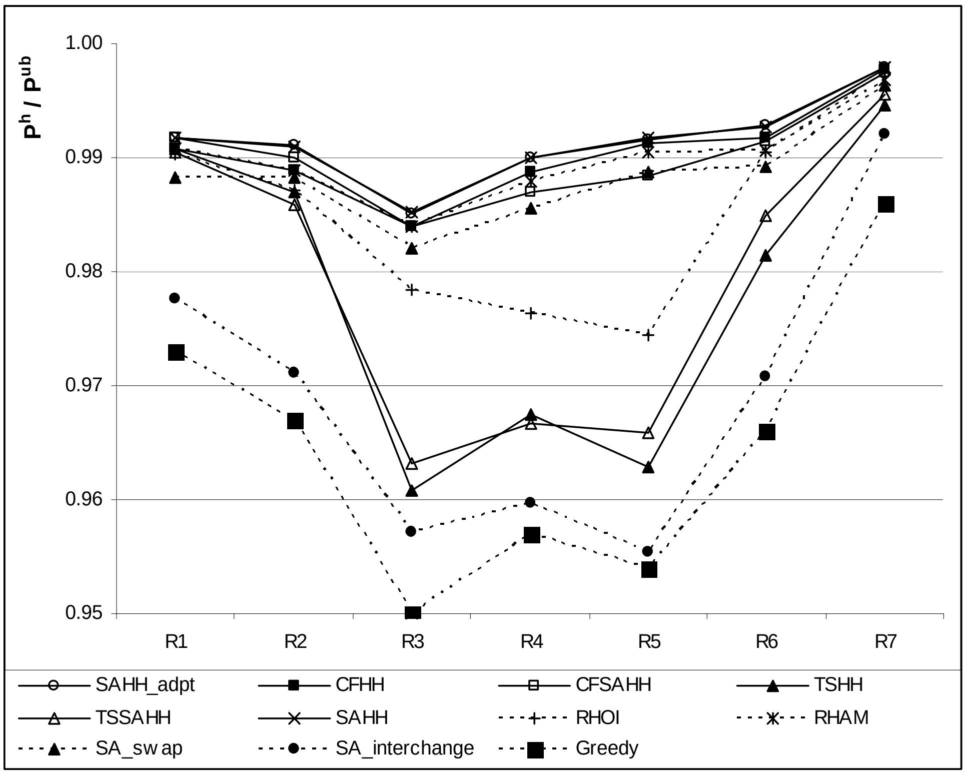


Figure 5-6: The average performance of different algorithms on the data set R

instances and 98.5% of the upper bound for one instance. This is a very good performance considering the fact that the upper bound was obtained by a two-stage relaxation and the algorithms used the same parameters to those that were used in

Table 5-3a: The performance of different algorithms for the data set R (see 5-3b for other results)

		R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
(r_min, r_max)		(0.95, 0.33)	(0.85, 0.35)	(0.7, 0.46)	(0.60, 0.53)	(0.50, 0.66)	(0.40, 0.79)	(0.34, 0.95)
(*m*, *n*)		(12, 54)	(11, 48)	(15, 48)	(16, 48)	(17, 48)	(22, 48)	(29, 50)
P^{ub}		422.25	401.33	411.05	435.55	471.30	526.04	482.64
Greedy	P^h	410.81	388.24	390.47	416.75	449.55	507.96	475.95
	P^h/P^{ub}	0.97%	0.967	0.95	0.957	0.954	0.966	0.986
SAHH	P^h	418.74	397.70	**404.96**	**431.20**	**467.43**	522.18	**481.67**
	min	418.42	397.39	404.61	430.94	467.23	521.72	481.29
	max	419.04	398.12	405.49	431.47	467.65	522.65	481.94
	P^h/P^{ub}	**0.992**	**0.991**	**0.985**	**0.990**	**0.992**	**0.993**	**0.998**
	stdev	0.16	0.17	0.20	0.13	0.10	0.22	0.14
SAHH_adpt	P^h	**418.77**	**397.79**	404.95	431.19	467.33	**522.26**	481.62
	min	418.42	397.45	404.54	430.53	466.93	521.73	481.37
	max	419.17	398.05	405.42	431.57	467.92	522.49	481.98
	P^h/P^{ub}	**0.992**	**0.991**	**0.985**	**0.990**	**0.992**	**0.993**	**0.998**
	stdev	0.23	0.14	0.22	0.22	0.26	0.18	0.14
CFHH	P^h	418.36	396.85	404.44	430.63	467.19	521.73	481.58
	min	412.10	388.24	402.29	428.14	464.89	520.21	480.90
	max	419.04	397.55	405.00	431.29	467.71	522.34	481.89
	P^h/P^{ub}	0.991	0.989	0.984	0.989	0.991	0.992	**0.998**
	stdev	1.21	1.61	0.64	0.75	0.46	0.44	0.18
CFSAHH	P^h	418.74	397.29	404.45	429.87	465.84	521.54	481.44
	min	418.07	395.73	403.06	426.33	460.81	519.87	480.38
	max	419.17	398.15	405.10	431.12	467.40	522.63	481.88
	P^h/P^{ub}	**0.992**	0.990	0.984	0.987	0.988	0.991	**0.998**
	stdev	0.27	0.65	0.54	0.97	1.85	0.65	0.33
TSHH	P^h	418.39	396.08	394.94	421.38	453.77	516.27	480.04
	min	417.41	394.45	391.25	419.93	451.11	511.94	478.43
	max	419.14	397.23	400.56	423.09	455.47	518.09	481.50
	P^h/P^{ub}	0.991	0.987	0.961	0.967	0.963	0.981	0.995
	stdev	0.38	0.62	2.35	0.77	1.07	1.46	0.76
TSSAHH	P^h	418.25	395.64	395.93	421.02	455.19	518.10	480.47
	min	417.07	393.93	393.29	418.73	452.63	514.29	479.61
	max	418.78	396.71	400.60	422.15	457.19	520.10	481.15
	P^h/P^{ub}	0.991	0.986	0.963	0.967	0.966	0.985	0.996
	stdev	0.35	0.63	1.77	0.74	1.06	1.21	0.39

solving the bin packing problem in chapter 4. The performance of most of the algorithms slightly decreased when *r_min* and *r_max* reached the middle of their ranges. This is probably because when *r_min* is large while *r_max* is small, the shelf space is very scarce and the optimal solution is near the lower bound and hence is relatively easier to obtain. Similarly, when *r_min* is small and *r_max* is large, space is abundant so that the optimal solution is almost the upper bound. However, when the available shelf space belongs to none of these two cases, the problem becomes harder to solve.

Table 5-3b: The performance of different algorithms on the data set R (continued)

		R1	R2	R3	R4	R5	R6	R7
(r_min, r_max)		(0.95, 0.33)	(0.85, 0.35)	(0.7, 0.46)	(0.60, 0.53)	(0.50, 0.66)	(0.40, 0.79)	(0.34, 0.95)
(*m*, *n*)		(12, 54)	(11, 48)	(15, 48)	(16, 48)	(17, 48)	(22, 48)	(29, 50)
P^{ub}		422.25	401.33	411.05	435.55	471.3	526.04	482.64
RHOI	P^h	418.16	396.14	402.21	425.23	459.23	521.03	481.38
	min	417.23	395.56	401.54	423.14	457.74	520.44	481.09
	max	418.92	396.68	403.12	426.47	460.42	521.68	481.65
	P^h/P^{ub}	0.990	0.987	0.978	0.976	0.974	0.990	0.997
	stdev	0.47	0.32	0.42	0.84	0.67	0.34	0.16
RHAM	P^h	418.39	396.90	404.46	430.29	466.84	521.15	481.10
	min	418.07	396.67	404.12	429.88	466.57	520.63	480.82
	max	418.78	397.23	404.84	430.91	467.14	521.49	481.40
	P^h/P^{ub}	0.991	0.989	0.984	0.988	0.991	0.991	0.997
	stdev	0.15	0.13	0.18	0.22	0.15	0.20	0.15
SA_swap	P^h	417.27	396.59	403.68	429.25	466.01	520.35	480.85
	min	416.23	395.98	403.07	428.00	464.53	519.61	480.23
	max	418.15	397.10	404.34	430.21	466.68	521.35	481.50
	P^h/P^{ub}	0.988	0.988	0.982	0.986	0.989	0.989	0.996
	stdev	0.42	0.27	0.31	0.51	0.44	0.43	0.26
SA_interchange	P^h	412.80	389.73	393.43	417.97	450.28	510.72	478.78
	min	412.67	389.00	392.66	417.83	449.84	510.55	478.19
	max	413.47	390.21	394.13	418.19	451.43	511.32	479.38
	P^h/P^{ub}	0.978	0.971	0.957	0.960	0.955	0.971	0.992
	stdev	0.30	0.37	0.45	0.08	0.49	0.26	0.33

The second round of experiments investigated the effect of the problem size on the performance of different algorithms. To avoid the influence of the space availability, as can be seen from table 5-1 all the problem instances were created

Table 5-4a: The performance of different algorithms on the data set S (see table 5-4b for other results)

		S1	S2	S3	S4	S5
(r_min, r_max)		(0.95, 0.24)	(0.95, 0.33)	(0.95, 0.25)	(0.95, 0.24)	(0.95, 0.24)
(*m, n*)		(5,20)	(12, 54)	(22, 60)	(30,80)	(40, 100)
P^{ub}		186.53	422.25	610.67	884.62	1077.376
Greedy	P^h	151.73	410.81	511.51	753.35	928.07
	P^h/P^{ub}	0.813	0.973	0.838	0.852	0.861
SAHH	P^h	**186.53**	418.74	599.75	**870.01**	1052.61
	min	186.53	418.42	590.27	862.77	1040.24
	max	186.53	419.04	603.97	872.49	1058.81
	P^h/P^{ub}	**1.000**	**0.992**	0.982	**0.983**	0.977
	stdev	0.00	0.16	3.08	2.51	4.37
SAHH_adpt	P^h	**186.53**	**418.77**	**600.18**	867.60	**1056.44**
	min	186.53	418.42	594.47	857.17	1045.79
	max	186.53	419.17	603.97	872.49	1062.98
	P^h/P^{ub}	**1.000**	**0.992**	**0.983**	0.981	**0.981**
	stdev	0.00	0.24	2.09	3.30	3.78
CFHH	P^h	182.18	418.36	572.15	836.26	1006.71
	min	160.43	412.10	527.72	774.71	928.07
	max	186.53	419.04	600.82	866.27	1049.03
	P^h/P^{ub}	0.977	0.991	0.937	0.945	0.934
	stdev	7.13	1.23	27.47	32.55	45.38
CFSAHH	P^h	183.34	418.74	586.45	858.23	1035.80
	min	169.13	418.07	528.04	796.07	956.60
	max	186.53	419.17	605.77	880.91	1058.81
	P^h/P^{ub}	0.983	**0.992**	0.960	0.970	0.961
	stdev	4.84	0.28	22.60	25.14	30.84
TSHH	P^h	**186.53**	418.39	595.60	833.73	987.97
	min	186.53	417.41	583.06	822.69	937.91
	max	186.53	419.14	603.97	848.35	1039.49
	P^h/P^{ub}	**1.000**	0.991	0.975	0.942	0.917
	stdev	0.00	0.39	3.96	6.08	23.36
TSSAHH	P^h	182.18	418.25	569.46	825.19	961.98
	min	177.83	417.07	543.10	806.75	937.91
	max	186.53	418.78	587.23	836.24	986.24
	P^h/P^{ub}	0.977	0.991	0.933	0.933	0.893
	stdev	4.42	0.36	10.14	7.13	12.39

such that their space availability ratios are almost the same. Figure 5-7 and tables 5-4a and 5-4b show the corresponding experimental results and comparison. It can be seen that, once again, SAHH and SAHH_adpt outperformed all other algorithms, including two simple simulated annealing algorithms. For the smallest problem instance S1, four algorithms, SAHH, SAHH_adpt, TSHH and RHAM, consistently produced the optimal solution for all 30 runs (when P^h/P^{ub}=1, it means that the algorithm has solved the problem to the upper bound. The solution found by the algorithm is the optimal solution). The results also show that most algorithms

Table 5-4b: The performance of different algorithms on the data set S (continued)

		S1	S2	S3	S4	S5
(r_min, r_max)		(0.95, 0.24)	(0.95, 0.33)	(0.95, 0.25)	(0.95, 0.24)	(0.95, 0.24)
(*m*, *n*)		(5,20)	(12, 54)	(22, 60)	(30,80)	(40, 100)
P^{ub}		186.53	422.25	610.67	884.62	1077.376
RHOI	P^h	185.57	418.16	598.69	866.62	1048.22
	min	172.18	417.23	594.47	857.17	1037.87
	max	186.53	418.92	603.97	872.49	1059.05
	P^h/P^{ub}	0.995	0.990	0.980	0.980	0.973
	stdev	3.64	0.48	3.00	4.56	5.33
RHAM	P^h	**186.53**	418.39	594.58	859.51	1039.01
	min	186.53	418.07	589.24	857.45	1029.47
	max	186.53	418.78	599.77	866.27	1048.22
	P^h/P^{ub}	**1.000**	0.991	0.974	0.972	0.964
	stdev	0.00	0.43	2.12	2.77	5.96
SA_swap	P^h	177.54	417.27	563.03	835.60	964.58
	min	169.13	416.23	543.10	818.57	944.51
	max	177.83	418.15	583.06	845.96	987.11
	P^h/P^{ub}	0.952	0.988	0.922	0.945	0.895
	stdev	1.59	0.43	9.24	6.36	10.80
SA_interchange	P^h	151.73	412.80	566.85	816.81	1026.59
	min	151.73	412.67	543.10	796.07	1018.37
	max	151.73	413.47	583.06	835.91	1035.44
	P^h/P^{ub}	0.813	0.978	0.928	0.923	0.953
	stdev	0.00	0.30	10.78	10.64	4.18

performed slightly worse when the problem size increased but both SAHH and SAHH_adpt still obtained more than 97% of the relaxed upper bound for a very large problem (m=40, n=100).

As SAHH and SAHH_adpt outperformed all other algorithms for both data sets, in the next three sections we carried out a specific comparison and analysis on the simulated annealing hyper-heuristics.

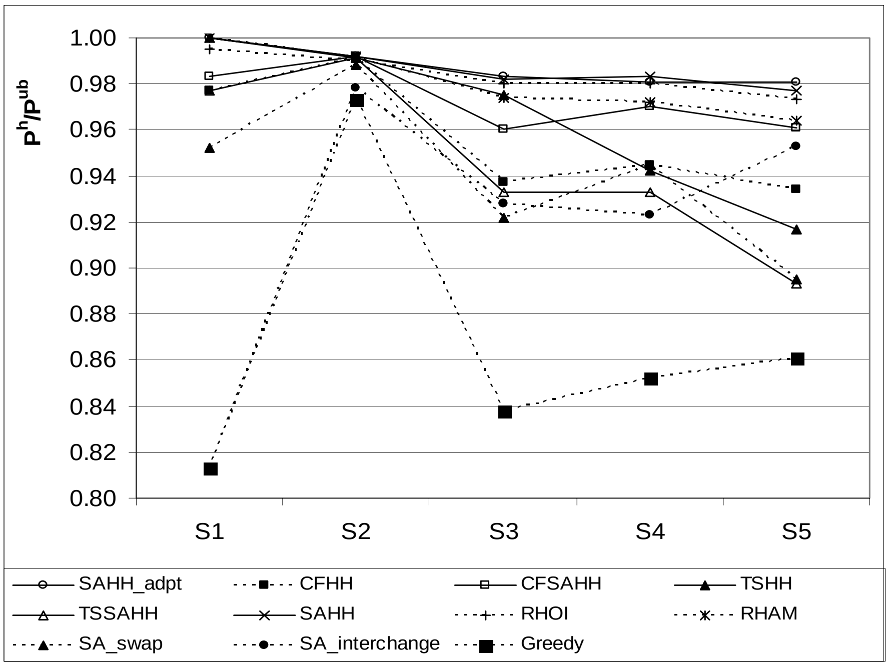


Figure 5-7: The average performance of different algorithms on the data set S

5.4.1 Comparison with conventional simulated annealing algorithms

As discussed in chapter 4, the simulated annealing hyper-heuristics simultaneously make use of a set of heuristics or neighbourhood functions. However, in a conventional simulated annealing algorithm, only a single neighbourhood structure is used. This section gives a more detailed comparison and analysis of their performance for the shelf space allocation problem. Figures 5-8 and 5-9 present a clearer comparison of the average performance and standard deviation between SAHH, SAHH_adpt, SA_swap and SA_interchange for the data set S and R respectively.

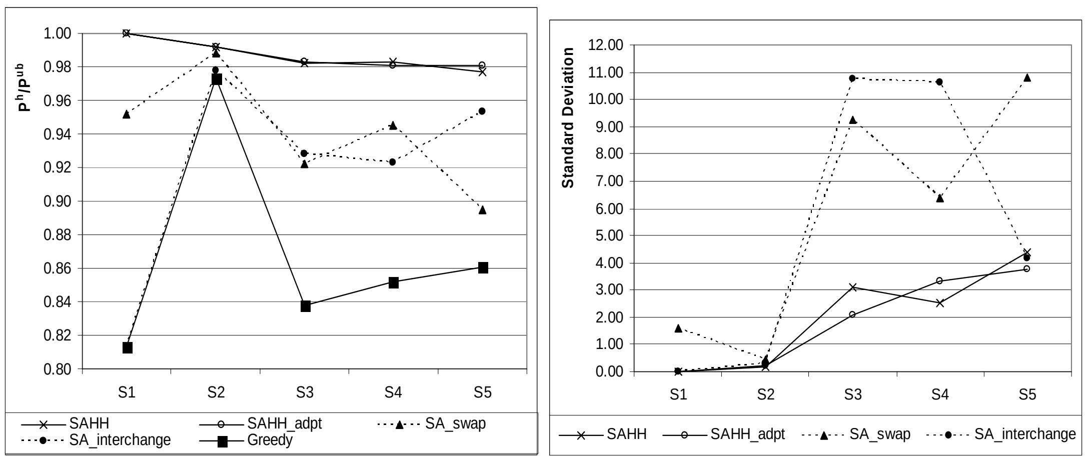


Figure 5-8: The average objective value and standard deviation of simulated annealing hyper-heuristics and the conventional simulated annealing for the data set S over 30 runs

From both figures, it can be seen that the two simulated annealing hyper-heuristics, SAHH and SAHH_adpt, clearly outperformed two general SA algorithms, SA_swap and SA_interchange, for all tested instances both in terms of average objective values and standard deviation, which reflects the consistency and robustness of the algorithms. It also appears that, for the shelf space allocation

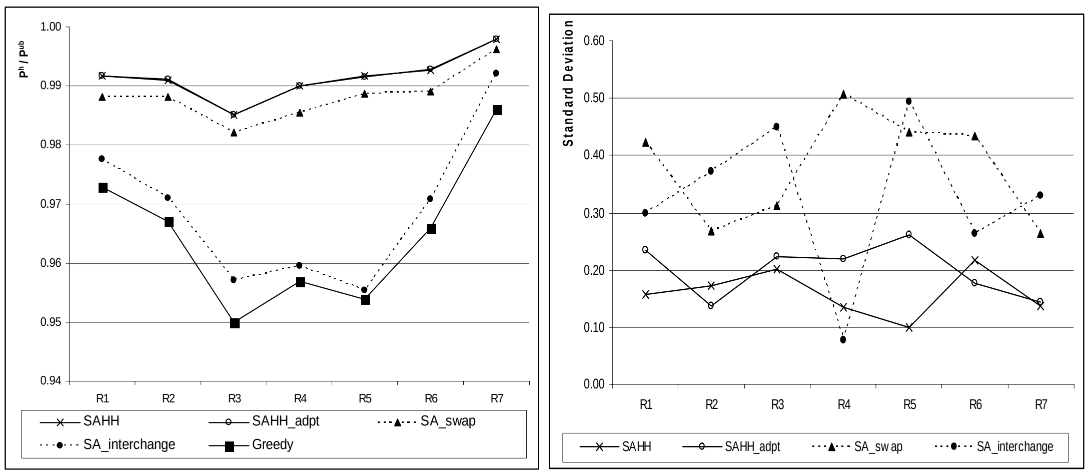


Figure 5-9: The average objective value and standard deviation of simulated annealing hyper-heuristics and the conventional simulated annealing for the data set R over 30 runs

problem, the simulated annealing hyper-heuristic is robust and not sensitive to the change of the starting and stopping temperatures. SAHH and SAHH_adpt have almost the same performance for all instances in the data set R and most of the instances in data set S even when their starting temperatures are very different. For example, when solving the problem instance R1, the starting temperature of SAHH was almost 4 times the value of SAHH_adpt, but both algorithms resulted in the same quality solutions. For problem instance S5, the starting temperature of SAHH was 7 times the value of SAHH_adpt, however, SAHH was only slightly beaten by SAHH_adpt by a margin of 0.4% (see table 5-4a).

In contrast, figures 5-8 and 5-9 show that both SA_swap and SA_interchange seem to be very sensitive to the change of the neighbourhood structures and the problem instances. For data set R, SA_swap performed much better than SA_interchange. This shows that for this set of problem instances, neighbourhood "swap" appears much better than the neighbourhood "interchange" even though the rest of the SA parameters are the same. Therefore choosing a "correct" neighbourhood structure is crucial for the success of SA. However, there is no guarantee that the neighbourhood "swap" is better than "interchange" for all problem instances. From figure 5-9, it can be seen that SA_swap performed better than SA_interchange for small problem instances and worse than SA_interchange for the large problem sizes. This shows that, for conventional SA, a good neighbourhood structure for a given problem instance does not guarantee good performance for another problem instance. However, by synergising several neighbourhood functions (or low-level heuristics), simulated annealing hyper-heuristics are able to achieve solutions with better quality and are also more general across different problem instances.

5.4.2 A comparison among different hyper-heuristics

Chapter 4 has discussed the motivation for the introduction of a simulated annealing acceptance criterion in the hyper-heuristic framework. It is expected that by the incorporation of an SA acceptance criterion, the performance of the current hyper-heuristic framework would be improved. To test this hypothesis, the average performance and the standard deviation of all hyper-heuristic algorithms have been plotted in figures 5-10, 5-11, 5-12 and 5-13 (SAHH was not included because its performance is very similar to SAHH_adpt).

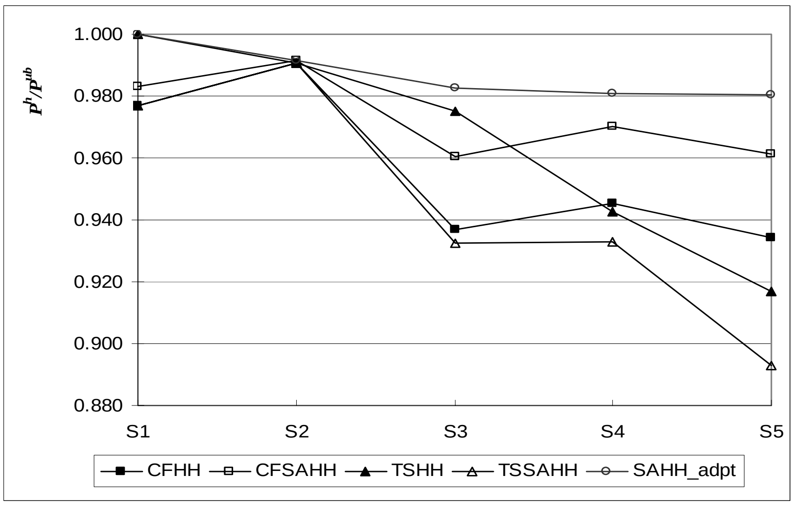


Figure 5-10: The average performance of different hyper-heuristics for the data set S over 30 runs

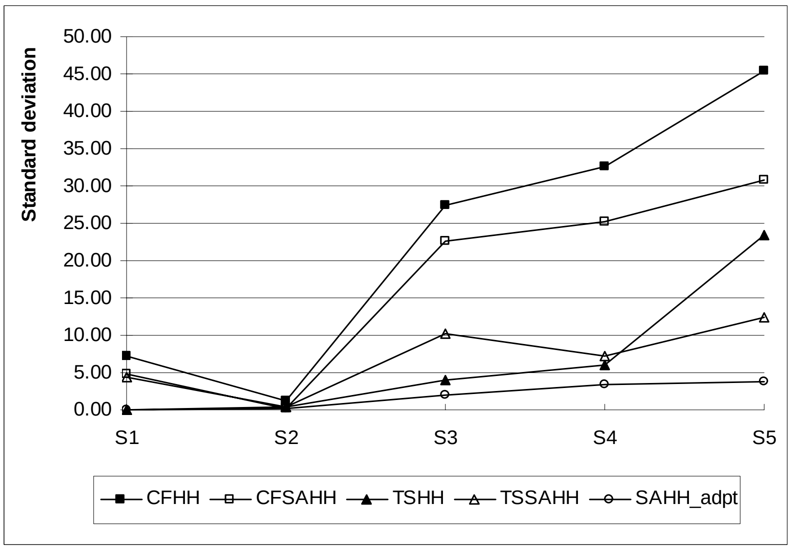


Figure 5-11: The standard deviation of different hyper-heuristics for the data set S over 30 runs

When solving data set S, it can be seen from figure 5-10 and 5-11 that CFSAHH, the choice function based hyper-heuristic with the assistance of an SA acceptance criterion, outperformed the pure choice function hyper-heuristic CFHH both in terms of the average objective value and the algorithm's robustness measured by the standard deviation. This is in line with our expectation. However, both CFHH and CFSAHH were still inferior to SAHH_adpt (or SAHH) which randomly selects different low-level heuristics rather than using the choice function heuristic selection mechanism. Being contrary to our expectation, SA assisted tabu search hyper-heuristic, TSSAHH, failed to show superiority to TSHH. This is probably because the deterministic heuristic selection strategies in the CFHH and TSHH may be unsuitable for simulated annealing which, in essence, is a stochastic method. The deterministic heuristic selection may undermine the neighbourhood reachability. Comparing the tabu search hyper-heuristics and the choice function based hyper-heuristics, neither algorithm demonstrated superior performance over the other for this data set. TSHH performs better than CFHH and CFSAHH on the instance S1 and S3, while it was beaten by both CFHH and CFSAHH on the instance S4 and S5. Both algorithms have similar results on the instance S2. However, results obtained by both TSHH and TSSAHH show better consistency than those by CFHH and CFSAHH whose corresponding standard deviation increased very quickly with the increase of the problem size.

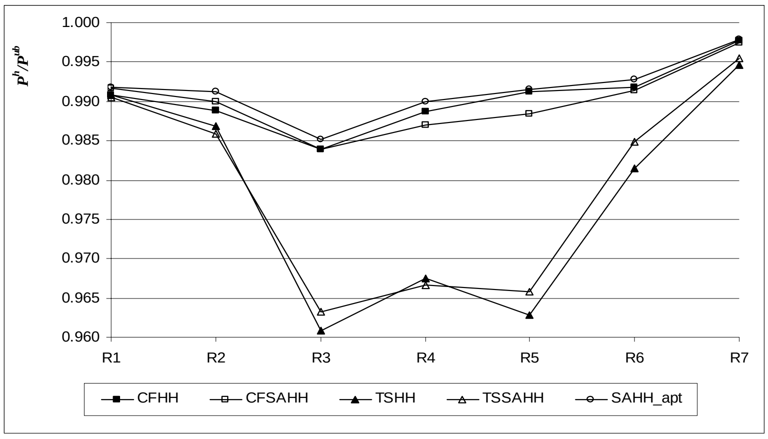


Figure 5-12: The average performance of different hyper-heuristics for the data set R over 30 runs

For the data set R, figure 5-12 and 5-13 shows that SAHH_adpt, again, outperforms all of the other four hyper-heuristics, although in one or two instances the difference between them are relatively small (see table 5-3a for the details). In terms of the average objective value, CFHH and CFSAHH obtained relatively good results that are close to the results by SAHH_adpt. However, the standard deviation of SAHH_adpt is much smaller than CFHH and CFSAHH. TSHH and TSSAHH performed badly for most instances except R1. Comparing CFHH and CFSAHH, no clear-cut difference can be observed. Similarly, there is no clear difference between TSHH and TSSAHH although TSSAHH seem to produce more consistent results with a smaller standard deviation on this data set.

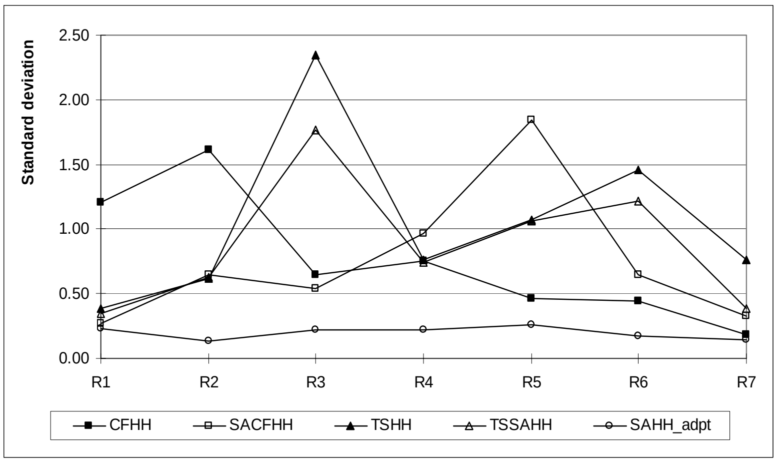


Figure 5-13: The standard deviation of different hyper-heuristics for the data set R over 30 runs

5.4.3 Robustness analysis

As all above meta-heuristic approaches include some random elements, different results may be obtained if running the same algorithm several times. It is not desirable that those results are significantly different from each other. An algorithm should be robust enough that the results obtained by different runs are clustered closely around the mean value. The standard deviation values in table 5-3a, 5-3b, 5-4a and 5-4b provides overall information of an algorithm's consistency and robustness. We also plot the distribution of the results obtained by different algorithms over 30 runs. For reasons of space, only results for the instance S5 are presented, as shown in figures 5-14 to 5-23 respectively. Similar distribution plots can be obtained for the majority of other instances. The horizontal axis represents the objective values an algorithm obtains. The vertical axis represents the number of occurrence that a given objective value appeared over 30 runs. It can be seen that both the results from SAHH and SAHH_adpt are clustered closely around their mean values. However, all of the other algorithms are either producing results scattered

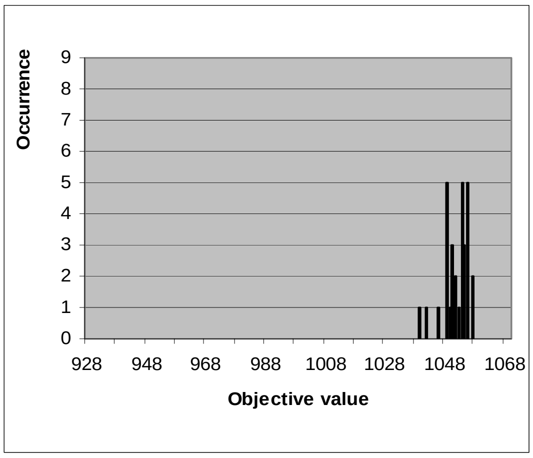


Figure 5-14: The objective value distribution of 30 SAHH runs on instance S5

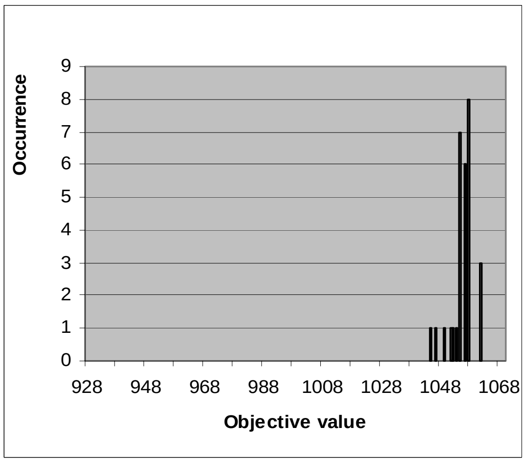


Figure 5-15: The objective value distribution of 30 SAHH_adpt runs for instance S5

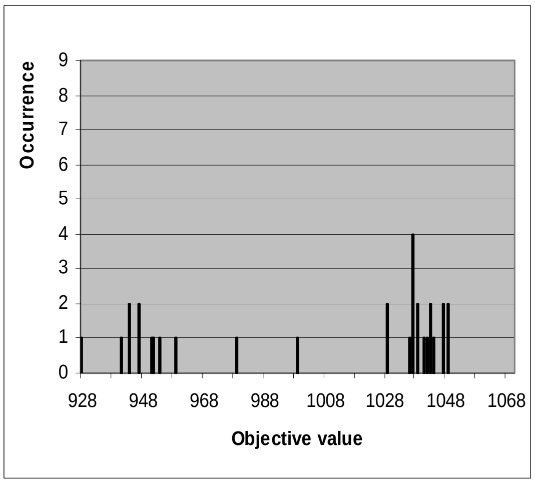


Figure 5-16: The objective value distribution of 30 CFHH runs for instance S5

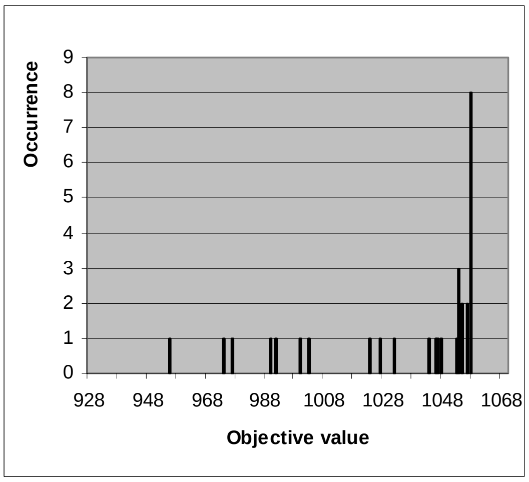
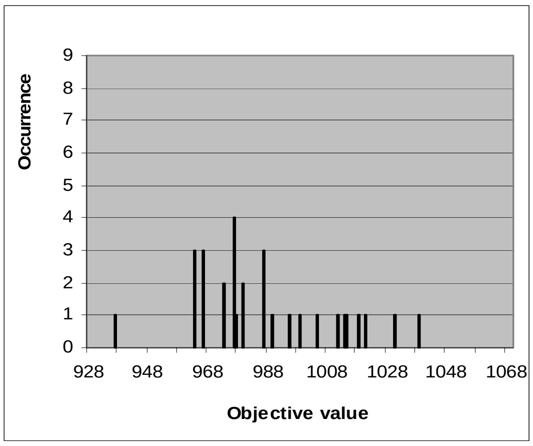


Figure 5-17: The objective value distribution of 30 CFSAHH runs for instance S5



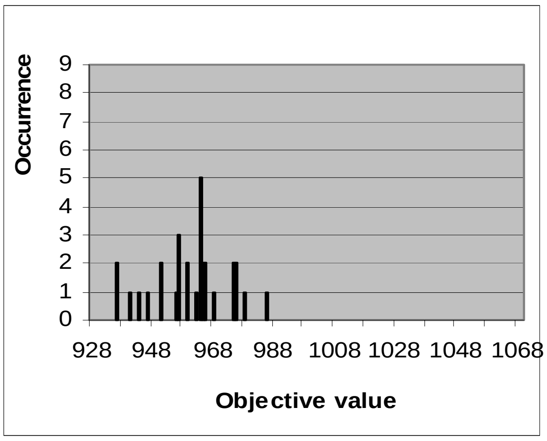
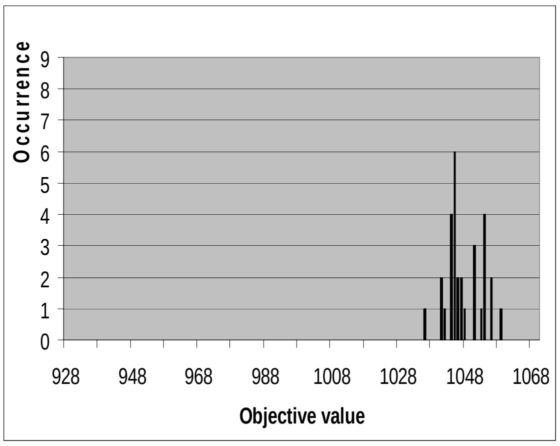


Figure 5-18: The objective value distribution of 30 TSHH runs for instance S5

Figure 5-19: The objective value distribution of 30 TSSAHH runs for instance S5



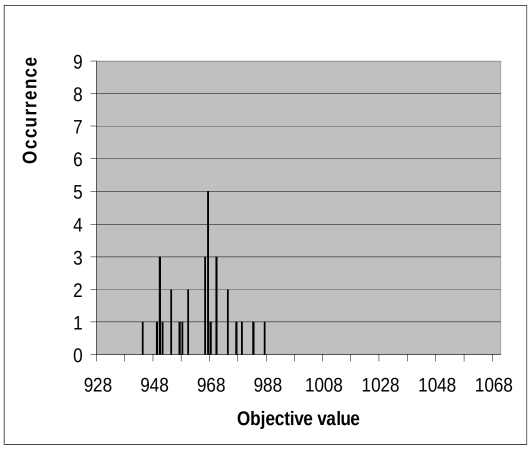
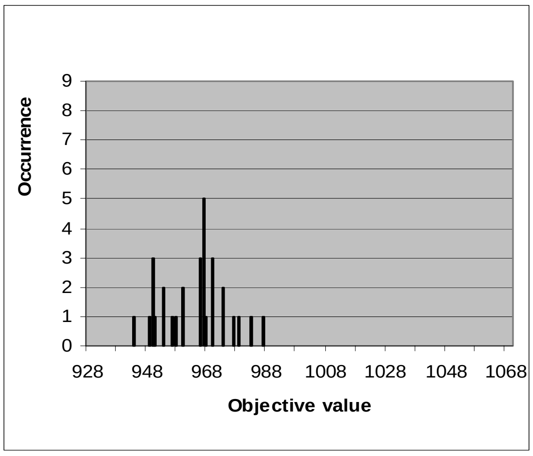


Figure 5-20: The objective value distribution of 30 RHOI runs for instance S5

Figure 5-21: The objective value distribution of 30 RHAM runs for instance S5



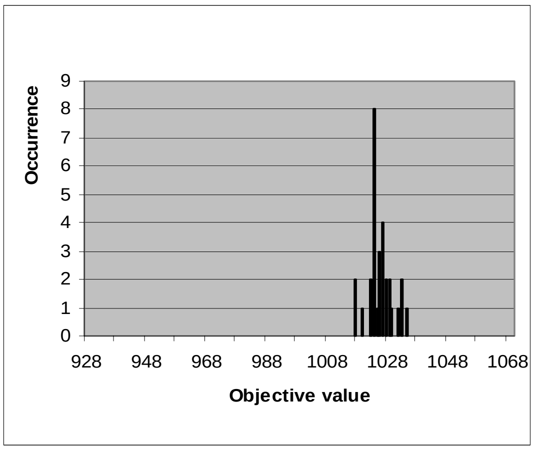


Figure 5-22: The objective value distribution of 30 SA_swap runs for instance S5

Figure 5-23: The objective value distribution of 30 SA_interchange runs for instance S5

widely along the horizontal axis, or are obtaining a much smaller average objective value.

5.5 Summary and Remarks

This chapter considered a general shelf space allocation problem that has been previously studied by the other researchers. A simplified, while practical, model was proposed as an alternative to a complex model which is not suitable for real-world applications. The model is a non-linear combinatorial problem and was shown to be related to the multi-knapsack problem. As an extension of the multi-knapsack problem, the shelf space allocation problem is difficult to solve. To give a better evaluation of the solution quality obtained by different algorithms, an upper bound of the objective function was derived by a two-stage relaxation method. In the first step, the non-linear model was transformed to a linear integer programming (IP) model by applying a first order Taylor expansion over the objective function. In the second step, the IP model was further relaxed to a linear programming (LP) model whose optimal solution was taken as the upper bound of the original non-linear shelf space allocation model.

Heuristic and meta-heuristic approaches, especially simulated annealing hyper-heuristics, have been investigated for the optimisation of a general shelf space allocation problem. The simulated annealing hyper-heuristic differs from other hyper-heuristics in that it is not only concerned with the intelligent selection of appropriate heuristics but also provides a robust acceptance criterion that systematically changes the acceptance ratio of inferior heuristic moves. Experiments and analysis have been carried out to compare SA based hyper-heuristics with two other hyper-heuristics that have been proposed by previous researchers. The SA based hyper-heuristic has also been compared with two general simulated annealing algorithms.

All of the algorithms have been tested on twelve simulated problem instances which can be divided into two sets, corresponding to two potential factors that may affect the performance of an algorithm. In the first set, space availability is considered and evaluated by two values, the ratio of minimal space requirements to the available space and the ratio of available space to the maximal space requirements. Seven problem instances with different ratios have been generated. The second data set consists of five problem instances of different problem sizes.

From the experimental results on these data sets, it has been observed that for the general simulated annealing algorithm, the neighbourhood structure plays a very important role in influencing the performance of an algorithm and an optimal neighbourhood structure may not exist across all problem instances. Among the problem instances that were tested in this chapter, it has been found that the performance of a given neighbourhood is dependent on the problem instance. A neighbourhood structure that works well on some problem instances may not perform well for other instances. However, the simulated annealing hyper-heuristic approach could simultaneously explore several neighbourhoods using different low-level heuristics. The experimental results show that this algorithm outperformed two random heuristic methods, two choice function based hyper-heuristics, two tabu search based hyper-heuristics and two versions of the conventional simulated annealing algorithms. The simulated annealing hyper-heuristic also does not seem to be parameter-sensitive, which has always been a problem for conventional simulated annealing algorithms. Overall, the simulated annealing hyper-heuristics has produced high quality solutions and even for the largest problem instance, they still achieved over 98% of the upper bound.

The low-level heuristics used in this chapter can be refined or more intelligent low-level heuristics may be designed and input in the simulated annealing hyper-heuristics. However, after discussions with our industrial collaborators, they are satisfied with their current shelf space allocation solutions and suggested us to work on another even more important shelf space allocation problem, fresh food. This is an area in which they have a particular interest due to increasing market competition. The problem is difficult because of the deterioration of the fresh food and the difficulty in managing inventory and shelf space allocation for such products. Indeed, the industries are not usually interested in finding the optimal solution which is usually unknown for many real-world problems. What they are seeking is a new approach that can be quickly and easily implemented to improve their current approaches. The next two chapters 6 and 7 will address this new problem. In Chapter 6 a model is formulated that can be used to manage inventory control and shelf space allocation specifically for fresh produce. A multi-start generalised reduced gradient algorithm (GRG) is developed for the problem. In chapter 7 several other popular heuristic and meta-heuristic approaches are investigated for the optimisation of the problem formulated in chapter 6.

CHAPTER 6. MANAGING FRESH PRODUCE INVENTORY AND SHELF SPACE ALLOCATION

6.1 Introduction

The profit on general foods, such as cans, frozen vegetables, fruit juices, etc., is gradually decreasing due to highly competitive retail conditions. The demand for these products is also slowing. On the other hand, the demand for some other merchandise, such as fresh produce, organic food and children clothes, has increased dramatically owing to improving living standards (Johnson, 2002). This requires retailers to concentrate more in these areas. This chapter considers an inventory control and shelf space allocation problem specifically for fresh produce, such as vegetables, fruits, fresh meats, etc. The main characteristics of fresh items are their very short shelf-life and decaying utilities (or freshness) over time.

Most of the literature treats fresh produce as deteriorating items with a random lifetime and non-decaying utilities (Nahmias, 1982; Goyal and Giri, 2001). In this research, we assume that the shelf-life of product has a continuous utility and physically deteriorates over time. Freshness is one of the main criteria in evaluating a product’s quality and could dramatically affect its demand if its condition is inferior. To obtain a good financial performance from fresh goods, it requires the adoption of strict temperature control and intelligent inventory and shelf management systems. Furthermore, although a large number of deteriorating inventory models have been proposed in previous research, most of them are based on the analysis of a single-item, without the constraints of shelf space which arise when considering a range of goods. There is no work in the literature which has integrated a deteriorating inventory model with a shelf space allocation model (which plays a very important role in retail decision making due to the scarcity of shelf resources). In this chapter, we formulate a fresh produce management model which can simultaneously decide the ordering policy and allocate shelf space among different items, together with consideration of utility deterioration. Some properties of the model are concluded which could significantly reduce the search space. A generalised reduced gradient algorithm (GRG) is proposed and extended in order to optimise this problem. Finally, the results on a numerical example are provided. Since the GRG algorithm may not be efficient for larger problem instances, in chapter 7, we shall investigate heuristic and meta-heuristic approaches for the optimisation of this problem. The main content of this chapter is drawn from (Bai and Kendall, 2005a).

6.2 Drawbacks of the Previous Models

Perishable inventory has been intensively studied and a large number of models have been proposed in the literature. See (Nahmias, 1982; Raafat, 1991; Goyal and Giri, 2001) for comprehensive reviews. However, most models assume that a fixed fraction of the inventory deteriorates completely over time but the utilities of the items do not decay before their expiration dates. Few models specifically consider the fresh produce with the characteristics we mentioned in section 6.1. In summary, these models have the following drawbacks:

1) Most models (Liu, 1990; Jain and Silver, 1994) assume that fresh produce, such as vegetables, fruit and fresh meat, have a random lifetime (normally assuming an exponentially distributed lifetime). However, an item’s utilities do not decay over time. Hence different ages of items capture the same demand however fresh they are as long as they are not completely spoilt. This is contradictory to the common sense view as freshness is one of the most important qualities for fresh produce.

2) Some models (Mandal and Phaujdar, 1989; Giri et al., 1996) formulate the demand as a deterministic function of instantaneous inventory with the assumption that all stock could be displayed on the shelves. However, this situation seldom occurs in most supermarkets because the shelf space for fresh food is normally limited. It is also expensive due to the low temperature requirements. Therefore, only a part of the inventory can be displayed on the shelf. Shelf space allocation among different items is especially important in this situation. The significance of shelf space allocation for non-perishable merchandise has already been well addressed in previous research (Kotzan and Evanson, 1969; Curhan, 1972; Borin et al., 1994; Urban, 1998; Yang and Chen, 1999; Bai and Kendall, 2005).
3) The approaches that were used to optimise the models (Ben-Daya and Raouf, 1993; Kar et al., 2001) disregarded the integer nature of the solution and assumed that the objective function is a quasi-concave function and is differentiable. The last assumption is usually too strict for problems which involve many constraints.

Different deteriorating inventory models have been classified into two types in the literature: fixed lifetime models and random lifetime models. Examples of fixed lifetime models include photographic film, medicine, computer chips, canned food, etc. A major characteristic of this type of model is that inventory control caters for different ages of items with either a First-In-First-Out (FIFO) or Last-In-First-Out (LIFO) issuing policy (Nandakumar and Morton, 1990; Liu and Lian, 1999). However, fresh produce was usually treated as a typical example of a random lifetime product due to uncertain spoilage (Liu, 1990; Jain and Silver, 1994). These models usually assumed a constant fraction of inventory decay or obsolescence over time (called *exponential decay* in some publications).

Since fresh produce only has a very limited shelf life, most of the literature employed a single-period inventory model although different forms of demand function are used. Both stochastic and deterministic demand inventory models have been proposed for the perishable products. Ben-Daya and Raouf (Ben-Daya and Raouf, 1993) proposed a multi-item, single-period perishable inventory model with a uniform distribution demand. The objective was to maximise the total profit of all the items during one period. The “optimal” solution was calculated by a Lagrangean optimisation with the assumption that the objective is differentiable. The integer nature of the variables was also disregarded. Furthermore, the method is not efficient when there are a large number of constraints.

Rajan et al. (Rajan et al., 1992) proposed a dynamic pricing and ordering decision making model for decaying produce, in which the demand was assumed to be deterministic and dependent on the selling price. The products were assumed to have an exponential deterioration. Abad (Abad, 1996) also formulated the demand function as a function of instantaneous price. A closed-form mathematical procedure was carried out to solve the problem and parameter sensitivities were analysed. However, the approach is heavily dependent on the mathematical description of the model so that even adding a single constraint could result in this approach becoming invalid.

Some other models formulated the demand as a deterministic function of instantaneous inventory. Mandal and Phaujdar (Mandal and Phaujdar, 1989) formulated a single-period inventory model for deteriorating items. The demand rate was linearly dependent on the instantaneous inventory level and the inventory deteriorated according to a given function. Backordering was allowed and holding and shortage costs were also considered in the model. The objective was to minimise the average cost. The model was optimised by applying the derivative to the objective function. The variables included the time slots for different inventory stages and maximal stock level and maximal stock deficit. Giri et al. (Giri et al., 1996) formulated the demand as a polynomial function of instantaneous inventory in their perishable inventory model which also assumed an exponential decay. The objective was to maximise the profit, with order quantity and reorder point (or cycle time) as decision variables. Some time-dependent demand functions were also proposed in deteriorating inventory models to describe changing demand over time. Xu and Wang (Xu and Wang, 1990) assumed a linear time-dependent demand function within a limited time horizon. Exponentially time-dependent demand was also proposed to simulate a rapidly increasing/declining market (Hollier and Mak, 1983; Zhou et al., 2003). Yet Urban and Baker (Urban and Baker, 1997) used a multiplicative demand function of price, time and inventory level in their single-period inventory model with the aim of finding optimal ordering and pricing policies for non-perishable products.

The first research to consider the effect of utility deterioration on demand is (Fujiwara and Perera, 1993) in their formulation of an Economic Order Quantity (EOQ) perishable inventory model. In this publication, an exponential penalty function $\alpha(e^{\beta t}-1)$ $(\alpha > 0, \beta > 0)$ was used to measure the cost of keeping an aging item in inventory. A closed form of economic order quantities was obtained by a quadratic approximation of exponential terms. The results show that this model is consistent with other EOQ models with exponential decay. Sarker et al. (Sarker et al., 1997) also attempted to incorporate the negative effect of aging inventory on demand.

In their production-inventory model, the demand function in the inventory build-up phase and depletion phase considered a constant term and a negative term which is proportional to instantaneous inventory (i.e. $f(t) = D - \beta I(t)$, where $f(t)$ is the demand function, $\beta > 0$, D is the constant demand and $I(t)$ is the instantaneous inventory level). However, illogically, the demand during the inventory depletion phase is actually an increasing function due to the continuous decrease of the inventory $I(t)$ over time. This contradicts the authors' initial intention to represent a declining demand with aging inventory.

Almost all of the models described above only considered a single item without any constraints being included. The optimal solution was normally obtained by some mathematical derivations. Recently, researchers have begun to incorporate the shelf space allocation technologies into their inventory systems. Kar (Kar et al., 2001) proposed a single-period inventory model for multi-deteriorating items with the constraints of shelf space and investment. The problem considers selling the deteriorating items from two stores. At the beginning of the period the ordered items are separated into fresh items and items that have begun to deteriorate. The fresh items are shipped to the main store, selling with a high price and the deteriorating items are delivered to the second store and sold at a lower price. During the period all decayed items in the main store are retained and delivered to the second store. The demand rate in the first store was formulated as a function of the item selling price and instantaneous inventory. However, the demand in the second store was only dependent on the selling price. A generalised reduced gradient (GRG) method was used to optimise the model. However, as stated in (Lasdon et al., 1978), GRG may not be efficient or robust for larger problem sizes and can only guarantee a local optimum. Besides, the assumption of non-integer variables and a differentiable objective function are the major drawbacks of this approach in solving many NP-Hard problems with integer variables. Hence, some meta-heuristic approaches (Glover and Kochenberger, 2003) have been introduced to optimise these models. Borin et al. (Borin et al., 1994) used a simulated annealing approach to solve a product assortment and shelf space allocation problem. Genetic algorithms were employed in Urban's publication (Urban, 1998) to solve an integrated product assortment, inventory and shelf space allocation model.

6.3 Model Formulation

Instead of assuming that fresh food has a random lifetime with an exponential decay, in this research it is assumed that the demand for the fresh produce is deterministic and is both dependent on the displayed inventory level and the freshness of the goods. The freshness condition decreases according to a known function over time. The main difference between these two assumptions is that the former assumes that all items that have not yet deteriorated capture the same demand however fresh they are. This may sound reasonable for long lifetime perishable items (like photographic film and medicine) but is unrealistic for fresh produce as freshness is one of the most important aspects in evaluating their quality. In this research all fresh items are assumed to have a fixed, but very short, lifetime and will not entirely lose utilities before their expiration date. However, freshness keeps deceasing over time, which has an effect on demand. It should be noted that the assumption of a fixed lifetime of fresh produce, with decreasing utilities is realistic considering the advances in food planting, packing and conservation technologies, especially the introduction of temperature control systems in most supermarkets.

The following notations are used in our model:

- $D_i(t)$ is the demand function of item *i* over time *t*.
- $f_i(t)$ is a decreasing function (within the range of [0,1]), representing the freshness condition of item *i* over time. A larger value indicates a higher level of freshness.
- $I_i(t)$ is the inventory level of the stock at time *t*.
- q_i is the procurement quantity of item *i*.
- s_i is the number of the facings assigned to item *i*.
- r_i is the surplus of item *i* at the end of the cycle.
- W is the total shelf space available.
- a_i is the space required for one facing of item *i*.
- p_i is the unit selling price of item *i*.
- p_{di} is the unit discount price of item *i*. This price should be low enough such that all of the remaining items at the end of period can be sold very quickly at this price.
- c_{ai} is the unit acquisition cost of item *i*.
- c_{hi} is the unit holding cost of item *i* (including the costs caused by inventory loses, damage, maintenance, interest, insurance, etc.).
- c_s is the shelf cost per unit space.
- C_{oi} is the constant order cost of item *i* (independent of the order quantity q_i).
- T_{ei} is the lifetime of item *i* after which the item is rotten (i.e. cannot be sold).
- L_i is the lower bound of the number of facings of item *i*.
- U_i is the lower bound of the number of facings of item *i*.
- T_i is the length of the cycle period of item *i*.

Many researchers (Kar et al., 2001; Urban, 2002) use the function depicted in figure 6-1 to describe the change of inventory level over time *t*. From time 0 to t_{1i}, s_i facings of item *i* are displayed on the shelf with some of stock stored in the backroom. As sales are made, the items in the backroom are moved to the shelf until stock in the backroom reaches zero (corresponding to the point when time reaches t_{1i}). Therefore, during this period, the shelf is fully stocked and the demand is only a function of product freshness. From time t_{1i} to t_{2i}, the shelf is only partly stocked and

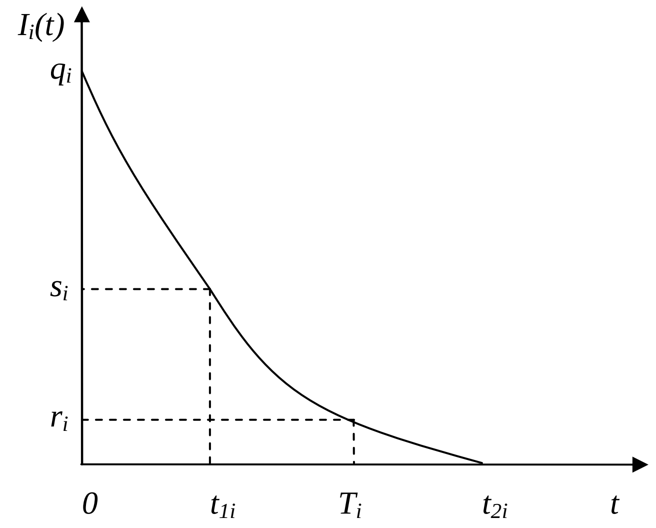


Figure 6-1: Graphical representation of inventory level changes over time

the demand is both dependent on the freshness and the instantaneous inventory level. Once the time reaches point T_i, a new order of quantity q_i is placed for item *i* (assuming no lead time) and the r_i surplus of item *i* are sold at a discount price p_{di}. In this research, we will adopt this representation together with a polynomial demand function that is widely used in many shelf space allocation models (Corstjens and Doyle, 1981b; Giri et al., 1996; Urban and Baker, 1997; Urban, 1998):

$$D_i^*(t) = \begin{cases} \alpha_i s_i^{\beta_i} & 0 \le t \le t_{1i} \\ \alpha_i [I_i(t)]^{\beta_i} & t_{1i} < t \le t_{2i} \end{cases} \qquad (6\text{-}1)$$

where α_i and β_i are scale parameter and the space elasticity of item *i* respectively and $\alpha_i > 0$, $0<\beta_i < 1$. In this research, we assume that the demand function conforms to a multiplicative form of the instantaneous inventory and the item's freshness condition, i.e. $D_i(t) = D_i^*(t)\cdot f_i(t)$ where $f_i(t)$ is a continuously decreasing function over time and $0 \le f_i(t) \le 1$. $f_i(t)$ could be a linear, quadratic or an exponential function of time. During the beginning of the period, the items are fresh and the value of freshness function is almost 1. The demand rate is only affected by the displayed inventory level. However, as time elapses, $f_i(t)$ gradually decreases and the demand is scaled down according to how long an item has been kept in inventory. To be consistent with the exponential decay assumption in the literature, here, we assume that the item's freshness condition decreases exponentially over time, i.e. $f_i(t) = e^{-\sigma_i t}$, where σ_i is a constant decay rate and $\sigma_i > 0$. Hence we have:

$$D_i(t) = D_i^*(t)\cdot f_i(t) = \begin{cases} \alpha_i s_i^{\beta_i} e^{-\sigma_i t} & 0 \le t \le t_{1i} \\ \alpha_i [I_i(t)]^{\beta_i} e^{-\sigma_i t} & t_{1i} < t \le t_{2i} \end{cases} \tag{6-2}$$

Based on the assumptions above, the inventory level of item *i* can be described by the following differential equation:

$$dI_i(t)/dt = -D_i(t) \tag{6-3}$$

During time [0, t_{1i}], we have

$$dI_i(t)/dt = -\alpha_i s_i^{\beta_i} e^{-\sigma_i t} \tag{6-4}$$

with the boundary conditions $I_i(0) = q_i$ and $I_i(t_{1i}) = s_i$. The solution of eq. (6-4) is:

$$I_i(t) = q_i + \frac{\alpha_i s_i^{\beta_i}}{\sigma_i}(e^{-\sigma_i t} - 1) \tag{6-5}$$

and

$$t_{1i} = -\frac{1}{\sigma_i}\ln\left(1 - \frac{(q_i - s_i)\sigma_i}{\alpha_i s_i^{\beta_i}}\right) \tag{6-6}$$

During time [t_{1i}, t_{2i}], we have the following differential equation:

$$dI_i(t)/dt = -\alpha_i [I_i(t)]^{\beta_i} \cdot e^{-\sigma_i t} \tag{6-7}$$

with the boundary conditions $I_i(t_{1i}) = s_i$ and $I_i(t_{2i}) = 0$. The solution of eq. (6-7) is:

$$I_i(t) = [\frac{\alpha_i(1-\beta_i)}{\sigma_i} e^{-\sigma_i t} + K_i]^{\frac{1}{(1-\beta_i)}} \tag{6-8}$$

and

$$t_{2i} = -\frac{1}{\sigma_i} \ln(1 - \frac{(q_i - \beta_i(q_i - s_i))\sigma_i}{\alpha_i(1-\beta_i)s_i^{\beta_i}}) \tag{6-9}$$

where $K_i = [q_i - \beta_i(q_i - s_i)]s_i^{-\beta_i} - \mu_i$ and $\mu_i = \dfrac{\alpha_i(1-\beta_i)}{\sigma_i}$.

In general, we have the following inventory function:

$$I_i(t) = \begin{cases} q_i + \dfrac{\alpha_i s_i^{\beta_i}}{\sigma_i}(e^{-\sigma_i t} - 1) & 0 \le t \le t_{1i} \\ [\mu_i e^{-\sigma_i t} + K_i]^{\frac{1}{(1-\beta_i)}} & t_{1i} < t \le t_{2i} \end{cases} \tag{6-10}$$

The length of cycle period T_i ($I_i(T_i) = r_i$) is:

$$T_i = -\frac{1}{\sigma_i} \ln\left[\frac{1}{\mu_i}(r_i^{(1-\beta_i)} - K_i)\right] \tag{6-11}$$

The holding cost during [0, t_{1i}] is:

$$HC_{1i} = c_{hi}\int_0^{t_{1i}} (q_i + \frac{\alpha_i s_i^{\beta_i}}{\sigma_i}(e^{-\sigma_i t} - 1)\ )dt = c_{hi}[(q_i - \frac{\alpha_i s_i^{\beta_i}}{\sigma_i})t_{1i} + (1 - e^{-\sigma_i t_{1i}})\frac{\alpha_i s_i^{\beta_i}}{\sigma_i^2}] \tag{6-12}$$

The holding cost during [t_{1i}, T_i] is:

$$HC_{2i} = c_{hi}\int_{t_{1i}}^{T_i} ([\mu_i e^{-\sigma_i t} + K_i]^{\frac{1}{(1-\beta_i)}})dt \tag{6-13}$$

The approximate expression of HC_{2i} can be calculated as follows. Denote $y(t)=[\mu_i e^{-\sigma_i t}+K_i]^{\frac{1}{(1-\beta_i)}}$. Divide range [$t_{1i}$, T_i] into k identical ranges by point x_0= t_{1i}, x_1, x_2,…, x_k= T_i. We have:

$$HC_{2i}=c_{hi}\int_{t_{1i}}^{T_i}([\mu_i e^{-\sigma_i t}+K_i]^{\frac{1}{(1-\beta_i)}})dt$$
$$\approx\frac{c_{hi}(T_i-t_{1i})}{k}[\frac{1}{2}(y(x_0)+y(x_k))+y(x_1)+...+y(x_{k-1})].$$

However, calculation results show that this part is very small and a simpler form is used in this thesis (using $(s_i+r_i)/2$ as an approximation of average inventory during [t_{1i}, T_i]):

$$HC_{2i}\approx c_{hi}(s_i+r_i)(T_i-t_{1i})/2 \tag{6-14}$$

Therefore, the average profit of item i per unit time being the total income less any costs involved divided by the time of the period, we have:

$$M_i(s_i,q_i,r_i)=\frac{1}{T_i}[p_i(q_i-r_i)+p_{di}r_i-c_{ai}q_i-C_{oi}-HC_{1i}-HC_{2i}]-c_s s_i a_i \tag{6-15}$$

The objective is to maximise the overall profit of all items during the unit time:

$$\max \qquad M=\sum_{i=1}^{n}M_i(s_i,q_i,r_i) \tag{6-16}$$

subject to:

$$\sum_{i=1}^{n}s_i a_i\le W \tag{6-17}$$

$$L_i\le s_i\le U_i \qquad i=1,2,...,n \tag{6-18}$$

$$r_i\le s_i\le q_i \qquad i=1,2,...,n \tag{6-19}$$

$$r_i<q_i \qquad i=1,2,...,n \tag{6-20}$$

$$0<T_i\le T_{ei} \qquad i=1,2,...,n \tag{6-21}$$

$$s_i, q_i \in \{1,2,3,...\} \quad i=1,2,...,n \tag{6-22}$$

$$r_i \in \{0,1,2,...\} \quad i=1,2,...,n \tag{6-23}$$

The decision variables are shelf space, order quantity and the amount of surplus at the end of the cycle. Constraint (6-17) ensures that the total shelf space allocated to each item is no more than the total available shelf space. Constraint (6-18) makes sure that the space allocated to each item must be within an upper and a lower bound. Constraint (6-19) ensures that the order quantity of each item must be no less than the shelf displayed quantity which itself should be greater than the number of surplus. Constraint (6-20) makes sure the order quantity is larger than surplus. Constraint (6-21) ensures that the length of the period T_i is non-zero and less than the product validity period. Constraint (6-22) and (6-23) ensures that the number of facings, order quantity and the number of surplus are integers. The model is a non-linear combinatorial optimisation problem and is difficult to optimise by utilising conventional mathematical approaches.

Suppose we have *n* products, the total number of variables is $3\times n$. From the model, we have the upper and lower bound of variables *r_i* ($0 < r_i \le s_i$) and s_i ($L_i < s_i \le U_i$) and lower bound of q_i ($q_i \ge s_i$). The upper bounds of q_i can be obtained from constraint (6-21). Since

$$T_i = -\frac{1}{\sigma_i}\ln\left[\frac{1}{\mu_i}(r_i^{(1-\beta_i)} - K_i)\right] \le T_{ei} \tag{6-24}$$

we have

$$q_i \le \frac{1}{(1-\beta_i)} r_i^{(1-\beta_i)} s_i^{\beta_i} + \frac{\alpha_i}{\sigma_i} s_i^{\beta_i} - \frac{\beta_i}{(1-\beta_i)} s_i - \frac{\alpha_i}{\sigma_i} e^{-\sigma_i T_{ei}} s_i^{\beta_i} \tag{6-25}$$

Let $\lfloor x \rfloor$ represents the largest integer no greater than value *x*, the upper bound of order quantity q_i^{ub} is

$$q_i^{ub} = \left\lfloor \frac{1}{(1-\beta_i)} r_i^{(1-\beta_i)} s_i^{\beta_i} + \frac{\alpha_i}{\sigma_i} s_i^{\beta_i} - \frac{\beta_i}{(1-\beta_i)} s_i - \frac{\alpha_i}{\sigma_i} e^{-\sigma_i T_{ei}} s_i^{\beta_i} \right\rfloor \qquad (6\text{-}26)$$

An interesting derivation of the model is that inventory depletes exponentially over time (see eq.(6-10)), which is consistent with the exponential decay models in the literature. In addition, when $\sigma_i \to 0$, $e^{-\sigma_i t} \to 1-\sigma_i t$, the inventory function becomes the same polynomial function derived in (Urban, 2002).

A further analysis of the model gives the following theorem:

Theorem: For given values of *s_i* ($L_i < s_i \le U_i, 1 \le i \le n$), the model (6-16) can be decomposed into *n* sub-problems with each sub-problem corresponding to optimising function (6-15) subject to the constraints (6-19, 6-20, 6-21, 6-22, 6-23).

Proof: For a fixed value of s_i ($L_i < s_i \le U_i, 1 \le i \le n$), the constraints (6-17, 6-18) can be ignored and the maximal profit of *i*th item (denoted by M_i^*) is independent of the decision variables of other items. Therefore, the optimal value of *M* (denoted by M^*) is equal to the sum of the optimal value of M_i, i.e. $M^* = \sum_{i=1}^{n} M_i^*$.

The theorem means that if the shelf space allocation decisions are made, the problem can be solved by independently searching for a pair of optimal order quantity (q_i) and surplus (r_i) for each item *i* ($1 \le i \le n$). As mentioned above, both variables have a lower bound and an upper bound. A simple way to achieve this is to carry out an exhaustive search, whose computational complexity is $O(q_i^{ub} s_i)$. The complexity for solving an *n*-item problem is $O(nq_i^{ub} s_i)$. Some mathematical approaches (binary search and the Newton method for example) may be more efficient if the function (6-15) can be proven to be a unimodal function. However, this is very difficult due to the complexity of the function.

By the theorem, we can reduce the size of the problem search space significantly. The original problem model (6-16) has a search space of a $n\times 3$ dimensional vector where n is the number of the items. However, with the theorem, the problem can be decomposed into two sub-problems: the first sub-problem aims to optimise n shelf space allocation variables (s_i). The second sub-problem is to search for the optimal values of ordering quantity (q_i) and surplus (r_i), for the given space allocation decisions made in the first sub-problem. Because the second sub-problem can be efficiently accomplished within a polynomial computational time, the search space is cut down to searching for a set of shelf space allocation decisions s_i in order to maximise the total profit. Once the shelf space allocation variables are decided, the corresponding optimal order quality and surplus can be decided efficiently. In this sense, the problem can be deemed as a nonlinear bounded knapsack problem.

However, even though the size of the problem search space can be decreased substantially and the model can be reduced to a nonlinear bounded knapsack problem, the problem is still NP-Hard (Bretthauer and Shetty, 2002). A generalised reduced gradient (GRG) algorithm was initially modified and developed to optimise the problem, which we describe in the next section.

6.4 A GRG Based Solution Procedure For the Problem

We initially used a generalised reduced gradient (GRG) algorithm to optimise the model. The GRG algorithm has been shown to be efficient in solving non-linear programming problems with smooth objective functions and its applications in optimising the inventory and shelf space allocation model include (Urban, 1998; Kar et al., 2001), with good results being reported.

```
Set MaxIter;
Set iter = 0;
Loop
    //Initialisation sub-procedure
    For each item i ($1 \le i \le n$) set $s_i = L_i$, $q_i = s_i$, $r_i = 0$;
    Loop
        Select a random item j;
        $s_j = s_j + 1$;
    Until no more facings can be added without violating the space
        constraint (6-17);
    For each item i
        Find the optimal values of $q_i$ and $r_i$;
    Output solution $S_0(q_i, s_i, r_i)$

    //GRG calling sub-procedure
    S'=Solver($S_0$);

    //Solution repair sub-procedure
    Round every $s_i$, $q_i$, $r_i$ ($1 \le i \le n$) in S' to their nearest integers
    While space constraint (6-17) is violated
        Rank the items by their unit space profit value $M_i/(a_i s_i)$;
        Delete one facing of the item with the smallest unit space profit
        value (if this operation causes a constraint violation, the next item
        in the ranking list is considered);
    Loop
    If free shelf space > the size of the smallest item
        Repeat
            Rank the items by their unit space profit value $M_i/(a_i s_i)$;
            Add one facing of the item with the largest unit space profit
            value (the next item in the ranking list is considered if the
            operation results in a constraint violation);
        Until no more facings can be added without violating the space
        constraint (6-17);
    Endif
    For each item i ($1 \le i \le n$)
        Find the optimal values of $q_i$ and $r_i$;
    Endfor
    Remember the best solution ($S_{best}$) found so far;
    iter++;
Until iter = MaxIter;
Output $S_{best}$;
```

Figure 6-2: Pseudo code of the multi-start GRG algorithm

The GRG algorithm is imbedded in many spreadsheet software packages. The one we used is called Solver, which is included in Microsoft Excel 2002. The GRG algorithm has two major drawbacks: 1. it can only solve continuous-variable models. Although the package included in Microsoft Excel 2002 can deal with integer variables, it takes too long for the search to converge (1800 seconds of computation time is needed even for a problem with 6 items, running on a PC with Pentium IV 1.8GHZ and 256MB RAM. For a problem with 18 products, the algorithm does not converge even after one hour). 2. GRG usually only gives a local optimum which is closest to the initial solution. Some preliminary experiments showed that, if the initial solution is not carefully chosen, GRG performs very badly. To solve these shortcomings, in this application, we used a multi-start GRG algorithm together with a solution repair heuristic to optimise the model. The multi-start search could prevent GRG from getting stuck at a local optimum and the repair heuristic is to recover the solution feasibility. Each run of the algorithm can be divided into three sub-procedures: initialisation, calling GRG and solution repair, described as in figure 6-2.

To prevent the GRG getting stuck at a local optimum, *MaxIter* runs of GRG were executed starting from different initial states (solutions) and the best solution was outputted as the final solution. In this application, we set *MaxIter* = 5 after some preliminary experiments considered both algorithmic performance and the required computational time. The initialisation sub-procedure was used to generate a set of diverse solutions that can be used by GRG. Note that because GRG is only efficient in handling continuous variables, a relaxed model (ignoring integer constraints (6-22) and (6-23)) was input into the Excel. Therefore, the solution output by GRG is not feasible. The solution repair sub-procedure was used to recover the feasibility of the solution and further improve it by using a simple local search method described in figure 6-2 (several other rounding heuristics were tried and the one presented in this chapter generally performs best across the five problem instances we tested). All results were averaged over ten runs on a PC with a Pentium IV 1.8GHZ CPU and 256MB RAM, running Microsoft Windows 2000 professional Version 5.

Table 6-1: Parameters of a numerical example

Item	a_i	p_i	c_{ai}	c_{hi}	p_{di}	C_o	α_i	β_i	σ_i
1	0.028	5.03	2.46	0.19	1.23	34.3	28.53	0.1532	0.06
2	0.061	9.37	5.67	0.20	2.84	48.9	23.62	0.2273	0.07
3	0.025	5.10	2.70	0.26	1.35	35.6	25.59	0.2089	0.06
4	0.060	11.48	6.11	0.16	3.06	47.9	22.40	0.2143	0.04
5	0.036	6.74	3.53	0.30	1.77	33.9	15.62	0.2955	0.03
6	0.033	5.97	3.41	0.27	1.71	39.1	10.50	0.3104	0.03
W=0.608(m^2), c_s=5.0(pounds/m^2/unit time), L_i =1, U_i =12, T_{ei} =7(days)									

Table 6-2: Solutions of the numerical example

Item	Solution by GRG				Optimal Solution			
	q_i	s_i	r_i	T_i	q_i	s_i	r_i	T_i
1	83	3	0	2.68	81	2	0	2.78
2	78	2	0	3.17	78	2	0	3.17
3	77	3	0	2.61	77	3	0	2.61
4	88	3	0	3.35	88	3	0	3.35
5	64	3	0	3.17	64	3	0	3.17
6	50	1	0	5.19	56	2	0	4.68
Objective	347.45				347.58			

6.5 A Numerical Example

To allow a better understanding of the model and the solution procedure described above, a numerical example with 6 items was generated (denoted by BORIN94/6). The problem scale parameters (α_i) and space elasticities (β_i) are taken from (Borin et al., 1994) and the other parameters are listed in table 6-1. The GRG algorithm described in section 6.4 was run 10 times with different initial random solutions. The algorithm consistently returned the same solution which is shown in table 6-2. For the purpose of an comparison, an exhaustive search was also carried out to get an optimal solution which is listed in table 6-2. It can be seen that for this numerical example, the solution obtained by GRG is very close to the optimal solution. The relative deviation from optimality is only 0.04% ($\frac{347.58-347.45}{347.58}\cdot 100\%$).

6.6 Summary and Remarks

In this chapter, a practical single-period inventory and shelf space allocation model has been proposed for fresh produce. Integrating an inventory model and shelf space allocation model is necessary because of the close relationship between inventory control and shelf space allocation. Many previous inventory models assume that the entire inventory can be displayed on the shelf. However, this is not practical because the shelf space for displaying fresh food is a very expensive resource and most retailers can only display part of the inventory on the shelf, with the rest being stored in the back room. Therefore, an inventory model should consider the availability of the shelf space. On the other hand, because the fresh produce only has limited shelf lifetime, it is necessary that all products should be sold out before their expiry dates. The shelf space allocation decisions should also consider the amount of the inventory and allocate more space to those products that have bigger inventories.

The second practicality of the model proposed in this chapter lies in the fact that our model introduces the freshness condition as a factor that could influence the demand of fresh produce. The freshness condition is continuously decreasing over time due to the utility decay associated with fresh produce. This is in contrast with existing fresh produce inventory models in the literature that usually assume that fresh produce has a random lifetime (normally assuming an exponentially distributed lifetime) and that item utilities do not decay over time.

In the proposed model, the demand for a fresh item is assumed to be deterministic and conforms to a multiplicative form of the displayed stock-level and items' freshness condition. The items' freshness condition is assumed to drop exponentially over time but could still capture some demand. Unlike other research, the proposed model considers the integer nature of the solution.

Some properties of the model have been analysed. It has been found that given a shelf space allocation decision s_i, the inventory control variables q_i and r_i can be optimised to optimality. Therefore, although the original problem model (6-16) has a search space of a $n\times 3$ dimensional vector where n is the number of the items, this search space can be reduced significantly by decomposing the problem using a two-step procedure: searching for a combination of shelf space allocation decisions and searching for the corresponding inventory variables for a given shelf space allocation decision obtained in the first step. The problem in the first step is similar to a non-linear bounded knapsack problem, which is still NP-Hard. For the problem in the second step, there exists at least one method that can solve it to optimality bounded by the time complexity of $O(q_i^{ub} s_i)$.

Because the problem in the first step is still NP-Hard, it is normally unrealistic to obtain the optimum of the model in reasonable computational time. A generalised reduced gradient (GRG) algorithm imbedded in Microsoft Excel 2002 Solver was used to optimise the model. To prevent GRG from getting stuck at local optima, the GRG algorithm was run several times from different initial points and the best solution was taken as the final solution. A post-procedure heuristic was also used to recover the feasibility of the solution. Finally a numerical example was given to allow readers a better understanding of the model and the solution procedure.

Although the multi-start GRG algorithm can produce good quality solutions, it may not be efficient when dealing with larger problem instances.

In the next chapter we will investigate several heuristic and meta-heuristic approaches for this problem. A set of larger sizes of problem data sets will be generated and the computational performance of the different algorithms will be evaluated and compared for these data sets.

CHAPTER 7. HEURISTICS AND META-HEURISTICS FOR THE FRESH PRODUCE INVENTORY CONTROL AND SHELF SPACE ALLOCATION PROBLEM

7.1 Introduction

In chapter 6, we formulated a practical shelf space allocation and inventory control model for the retail of fresh produce. The decision variables are the displayed facings s_i, order quantity q_i and the amount of surplus r_i for each item i. For an n-item problem, the total number of variables is $3\times n$. Further analysis of the model has shown that this search space can be reduced by decomposing the problem into a nonlinear bounded knapsack problem and a problem that can be solved by a polynomial time bounded algorithm (see section 6.3). A multi-start GRG algorithm was used to optimise the model. However, due to the NP-Hard nature of the nonlinear knapsack problem (Bretthauer and Shetty, 2002), GRG algorithm may not be efficient for large sizes of problem instances. Therefore, in this chapter, several heuristic and meta-heuristic approaches are investigated and compared for five problem instances. This chapter is mainly drawn from (Bai and Kendall, 2005d).

7.2 Test data sets

Although the numerical example in chapter 6 is helpful in understanding the model and testing the performance of the solution procedure, it is necessary to test the algorithm over larger problem instances. For this purpose, we created four larger benchmark problem instances using the parameters in table 7-1 (denoted by FRESH2, FRESH3, FRESH4 and FRESH5 respectively). The problem size ranges from 18 to

64 products. Those data sets can be downloaded from http://www.cs.nott.ac.uk/~gxk/research/.

Table 7-1: Parameters for generating problem instances

Parameters	Values	Parameters	Values
n	18/32/49/64	L_i	1
α_i	U(10, 30)	U_i	12
β_i	U(0.15, 0.3)	p_{di}	$0.5c_{ai}$
σ_i	U(0.03, 0.1)	c_s	5.0 pounds/m^2/day
a_i	U(0.01,0.09) m^2	C_o	U(30, 50) pounds
c_{ai}	N($100a_i$, 0.4) pounds	T_{ei}	7 days
p_i	N($1.8c_{ai}$, 0.4) pounds	W	2.5* *minSpace*
c_{hi}	U(0.1,0.3) pounds		

U(a, b): Uniform Distribution N(c, d): Normal Distribution
minSpace: the minimal space needed to satisfy products' minimal facings requirement

7.3 Optimisation of the Single-item Inventory

As illustrated in the previous chapter, given a set of shelf space allocation decisions s_i $(1 \le i \le n)$ that satisfy the constraint (6-17), the optimal values of q_i and r_i of item i $(1 \le i \le n)$ can be independently obtained by an exhaustive search with polynomial computational time. Further study of the model indicates that this exhaustive search procedure can be improved upon.

Let us firstly consider a single-item problem: for a given shelf space decision s, the problem is to search for a pair of order quantity and the amount of surplus (q and r) such that the unit space profit function (eq. 6-15) is maximised, subject to the constraints (6-19)-(6-23). Although no evidence has proven that function (6-15) is a unimodal function with respect to q, s and r, all of our experiments have shown this property. Figure 7-1 and figure 7-2 illustrates the relationships between the profit function (6-15) and the decision variables q, s and r. It can be seen that the profit function (6-15) has only one maximal value. The figures also show the sensitivity of profit function over the decision variables. From the figures, it can be seen that the profit function is more sensitive to the changes of facings s than order quantity q and surplus quantity *r*. This suggests that retailers should decide more carefully about displayed facings. A bad decision could result in a massive profit loss.

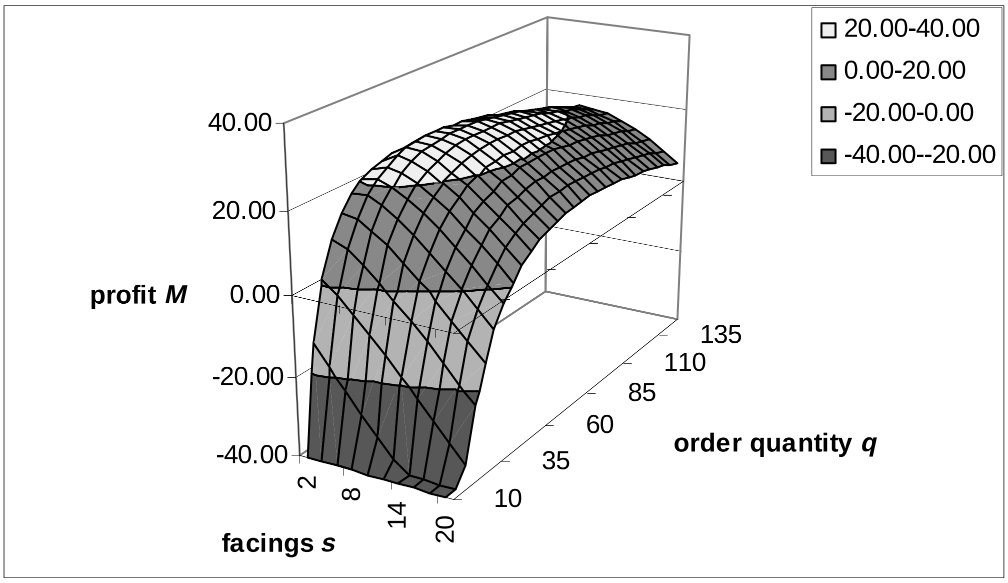


Figure 7-1: Graphic representation of an item's profit function with respect to facings *s* and order quantity *q* (surplus *r* = 0)

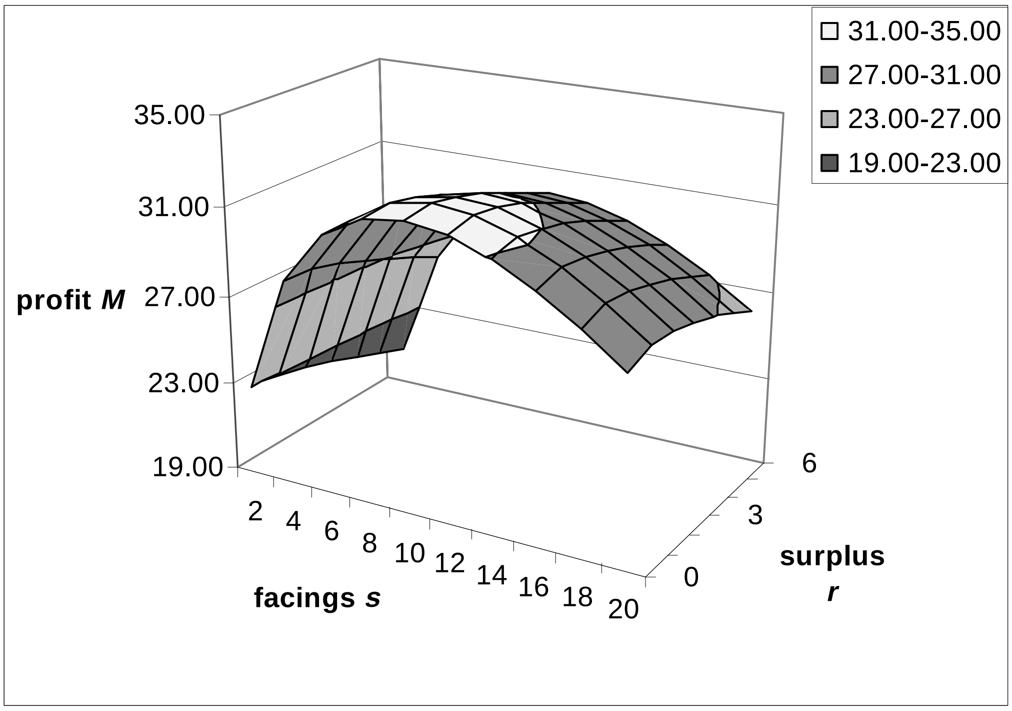


Figure 7-2: Graphic representation of an item's profit function with respect to facings *s* and surplus *r* (order quantity *q* = 90)

Figure 7-1 and figure 7-2 also show that the profit function (6-15) changes smoothly with the change of *q*, *s* and *r*. This encourages us to search for more efficient search methods. Generally, the Newton method can be used to achieve the optimal solution. However, because function (6-15) has a very complex form and no explicit expressions of derivatives can be obtained, it is very difficult to employ the Newton method in this case. Alternatively, a binary search was used in this research to get the optimal value of order quantity *q*. Meanwhile because *r* is relatively small (in most UK supermarkets, the number of facings of an item *s* is generally less than 12 and *r*<*s*), an enumeration method was used in the search for the optimal value of *r*.

```
Input s';
Set $s = s'$, $q_{opt} = 0$, $r_{opt} = 0$, $M_{opt} = -\infty$, $\varepsilon = 0.001$;
For each r=0 to s
    Set $q_l = s$;
    Set $q_r = q^{ub}$;
    Calculate $M_l(q_l, s, r)$ and $M_r(q_r, s, r)$;
    While ($q_r - q_l > 1$)
        $q = \lfloor (q_l + q_r)/2 \rfloor$;
        Calculate $M(q, s, r)$;
        If ($M^{-\varepsilon}(q - \varepsilon, s, r) < M$)
            $q_l = q$;  $M_l = M$;
        Else
            $q_r = q$;  $M_r = M$;
        Endif
    Loop
    If ($M_l < M_r$)
        $q'_{opt} = q_r$, $M'_{opt} = M_r$;
    Else
        $q'_{opt} = q_l$, $M'_{opt} = M_l$;
    Endif
    If($M_{opt} < M'_{opt}$)
        $q_{opt} = q'_{opt}$, $r_{opt} = r$, $M_{opt} = M'_{opt}$;
    Endif
Endfor
Output $q_{opt}$, $r_{opt}$;
```

Figure 7-3: The pseudo code of the procedure proc_qr(*s*')

Figure 7-5 presents the pseudo code for the binary search algorithm which, for simplicity, is denoted by proc_qr(*s'*). Suppose the shelf space allocated to an item is *s'*, for each possible value of *r*, a lower bound and an upper bound of *q* were calculated from inequalities (6-19) and (6-25) (denoted by q_l and q_r respectively). The algorithm then divides the range [q_l, q_r] into two equal parts (i.e. $q = (q_l + q_r)/2$) and checks in which half the optimal order quantity q'_{opt} lies. If q'_{opt} lies in the left half, it sets $q_r = q$, otherwise it sets $q_l = q$. This process is repeated until the length of the range [q_l, q_r] decreases to 1 and the optimal order quantity q'_{opt} is one of range boundaries (i.e. q_l or q_r). The total number of iterations of this procedure is no more than $s'\log_2^{q^{ub}}$ where q^{up} is the upper bound of order quantity. Because it is difficult to calculate the derivative of function (6-15), we used the method below to determine on which side the optimal order quantity q'_{opt} lies. Denote *M* as the profit when order quantity is *q* and $M^{-\varepsilon}$ the profit when we decrease *q* by a very small value ε (see figures 7-3 and 7-4). If $M^{-\varepsilon} > M_l$, q'_{opt} is at left side of *q*, otherwise, q'_{opt} is at right side of *q*.

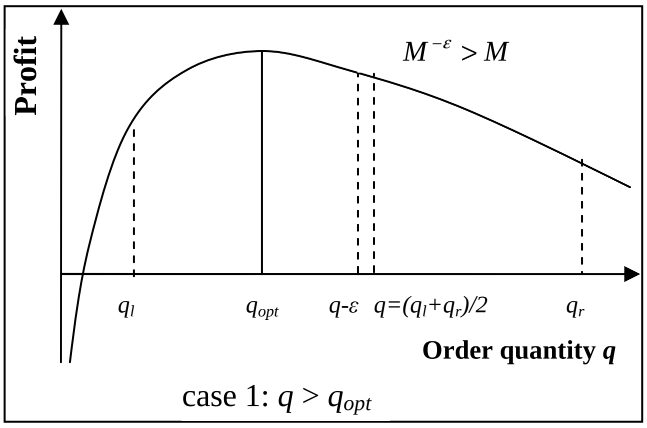


Figure 7-4: The relationship between order quantity and its unit time profit function ($q > q_{opt}$)

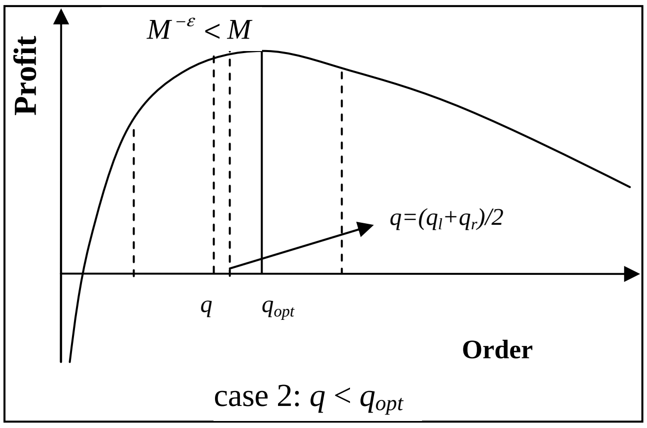


Figure 7-5: The relationship between order quantity and its unit time profit function ($q<q_{opt}$)

7.4 Greedy Heuristics for the Problem

In section 7.3, we have developed a sub-procedure proc_qr(*s'*) to obtain the optimal solution for a single-item inventory problem, with constant shelf space *s'* being allocated to the item. In this section, we shall consider the original problem (model (6-16)) where there are multiple items in the inventory with limited shelf space resources to display them. The items have to compete against each other for the shelf space such that the total profit is maximised. Once the amount of shelf space allocated to each item is determined, the procedure proc_qr(*s'*) can be applied to every item to find the corresponding optimal order quantity and the number of surplus. There could be many ways to allocate shelf space among items. A common sense rule to accomplish this would be to allocate shelf space in favour of more profitable items. The problem, in fact, degenerates into a problem similar to a bounded knapsack problem. However, it is also different. In the knapsack problem (see section 2.3.2), the profits of the items are constants and therefore each item's unit-space profit (i.e. profit/space) is constant as well. However, the space allocation problem in this research is much more difficult because the unit-space profit of every item is changing with the change of allocated shelf space. This chapter introduces four greedy heuristics for this problem.

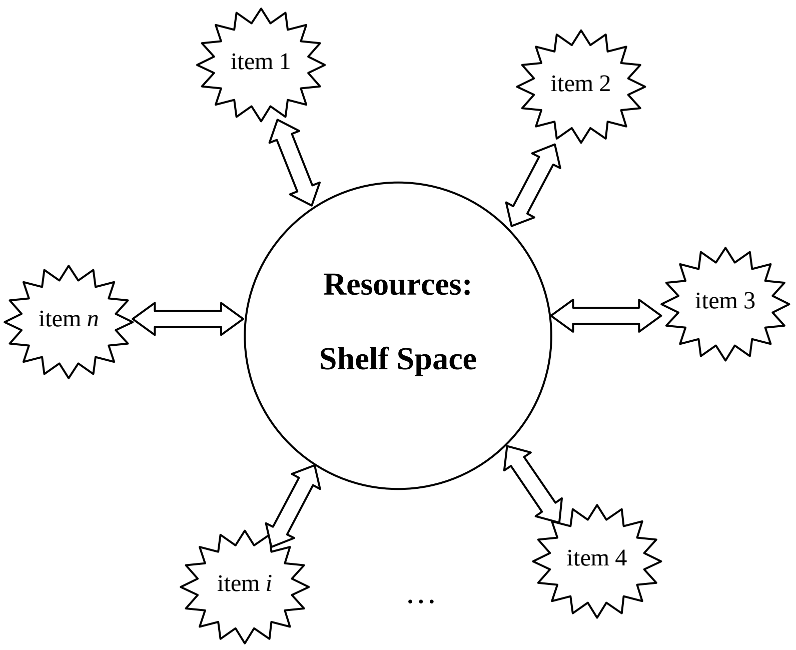


Figure 7-6: The graphic illustration of the greedy algorithms

Figure 7-6 shows the basic idea behind the algorithms. For a given amount of shelf space, each item is an intelligent entity optimising its own inventory variables (*q* and *r*) using the procedure proc_qr(*s'*). However, with the limited shelf space resources, these items have to compete and cooperate with each other such that the total profit of these items is maximised. Items that make less profit per unit shelf space must release part of their space to those which could make more profit if given more shelf space. Two functions were used to rank the profitability of different items with respect to the shelf space (denoted by C_1 and C_2 respectively). The first function is an item's unit space profitability, defined by $C_1 = M_i(s_i)/(a_i s_i)$. The second function is defined by $C_2 = (M_i(s_i) - M_i(s_i - \varepsilon))/(\varepsilon a_i)$ where ε is a small positive value (the derivative value is an ideal criterion but is difficult to calculate in this case). Because the profit function (6-15) is a non-linear function with respect to the facings *s*, both C_1 and C_2 are not constant and shall change with the changes of *s*.

Therefore, both profitability values C_1 and C_2 need to be recalculated at each solution construction step. There are two possible points where the greedy heuristics can start from. A greedy heuristic can start from a solution that has met the minimal space requirements and then repeatedly add a facing to the shelf according to the ranking functions C_1 or C_2 without violating the constraint (6-17). It can also start from a point where the facings of each item is equal to its upper bound and then repeatedly delete a facing according to the functions C_1 or C_2 until the space constraint (6-17) is satisfied. Therefore, there are a total of four combinations, denoted by GH_1, GH_2, GH_3 and GH_4 respectively, described as follows:

```
Step 1:
For each item $i$ ($1 \le i \le n$)
    $s_i = L_i$;
    Call proc_qr($s_i$) to get optimal $q_i$ and $r_i$;
    Calculate $C_1$ value for item $i$;
Endfor
Step 2:
If (FreeSpace > MinProdSpace)
    Select an item $j$ with largest possible profitability value of $C_1$
    and whose size is smaller than free space and the number of
    facing $s_j$ is less than its upper bound;
    If no such item is available, stop the procedure
    Else
        $s_j = s_j + 1$;
        Call proc_qr($s_j$) to get optimal $q_j$ and $r_j$;
        Update $C_1$ for item $j$;
        Go to step 2;
    Endif
Else
    Stop and output the solution.
Endif
```

Figure 7-7: Pseudo code of GH_1

GH_1 (Greedy_Fwd): This heuristic starts from a shelf space allocation decision that satisfies the minimal space requirements of each item and repeatedly adds to the shelf a facing of the item with the largest profitability value according to the criterion C_1. The heuristic stops as soon as no more facings can be added to the shelf. During this process, if adding a facing causes a constraint violation, the profitability value of this item is set to a very small value such that the item is of no further consideration. A full description of the algorithm is given in figure 7-7.

```
Step 1:
For each item $i$ ($1 \le i \le n$)
        $s_i = U_i$;
        Call proc_qr($s_i$) to get optimal $q_i$ and $r_i$;
        Calculate $C_1$ value for item $i$;
Endfor

Step 2:
While (SpaceUsed > SpaceAvailable)
        Select an item $j$ with the largest possible profitability value of
        $C_1$ and whose facing ($s_j$) has not reached its lower bound;
        $s_j = s_j - 1$;
        Call proc_qr($s_j$) to obtain the optimal $q_j$ and $r_j$;
        Update $C_1$ for item $j$;
Loop

Step 3:
If (FreeSpace > MinProdSpace)
        Select an item $k$ with the smallest possible profitability value of
        $C_1$ and whose area is smaller than free space and where the
        number of facing $s_k$ is less than its upper bound;
        If no such item is available, stop the procedure
        Else
                $s_k = s_k + 1$;
                Call proc_qr($s_k$) to obtain optimal $q_k$ and $r_k$
                Update $C_1$ for item $k$;
                Go to step 3;
        Endif
Else
        Stop and output solution.
Endif
```

Figure 7-8: Pseudo code of GH_2

GH_2 (Greedy_Bwd): This heuristic starts from an initial shelf space allocation that is equal to the corresponding upper bounds. Then the heuristic repeatedly deletes a facing of the item with the smallest profitability value of C_1 until the shelf space constraint is satisfied. Afterward, a sub-procedure is executed which tries to add (if possible) as many as possible facings to the shelf according the criterion of C_1 (see figure 7-8 for a detailed description).

GH_3 (Greedy_Derivative_Fwd): This heuristic is the same as GH_1 except that the greedy criterion is C_2 instead of C_1.

GH_4 (Greedy_Derivative_Bwd): This heuristic is the same as GH_2 except in using C_2 as the greedy criterion.

Table 7-2: The performance of the greedy heuristics in comparison with multi-start GRG

	BORIN94/6		**FRESH2**		**FRESH3**		**FRESH4**		**FRESH5**	
n	6		18		32		49		64	
	obj	**cpu(s)**	**obj**	**cpu(s)**	**obj**	**cpu(s)**	**obj**	**cpu(s)**	**obj**	**cpu(s)**
Multi-start GRG	**347.45**	3.2	1129.6	73.6	**2056.46**	74.3	**3163.98**	179.2	**4387.16**	209.7
GH_1	344.55	0.03	1126.8	0.03	2042.07	0.05	3144.02	0.05	4360.44	0.09
GH_2	344.55	0.05	1129.09	0.13	2041.59	0.28	3147.28	0.55	4358.96	0.91
GH_3	**347.45**	0.02	**1131.64**	0.02	2053.71	0.03	3159.17	0.06	4384.62	0.09
GH_4	346.90	0.06	1131.33	0.25	2054.14	0.50	3160.91	1.06	4382.66	1.45

obj: the objective value of the solution obtained by different algorithms (for multi-start GRG, this is the average value of 10 runs).
cpu(s): average CPU time consumed by different algorithms (in seconds).

Table 7-2 gives a comparison of the four greedy heuristics and the multi-start GRG algorithm (proposed in chapter 6) on the five test problem instances described in section 7.2. It can be seen that all greedy heuristics are very fast, compared with the multi-start GRG algorithm. GH_1 and GH_3 are also faster than GH_2 and GH_4. This is probably because the facings in the final solution are closer to their lower bound facings than to the upper bound facings for these instances. In terms of the solution quality, GH_3 and GH_4 performed better than GH_1 and GH_2 and are even competitive when compared with the multi-start GRG algorithm, which took much longer. Neither GH_3 nor GH_4 performed better than the other in terms of solution quality.

GH_3 is better on the instance BORIN94/6, FRESH2, FRESH5 while GH_4 is better than GH_3 on the other two instances. However, GH_3 consumed less time than GH_4.

7.5 Further Improvement over the Greedy Heuristics

Although the greedy heuristics in section 7.4 are very efficient in generating high quality solutions, for obvious reasons, they are prone to getting stuck at local optima. Three different meta-heuristic approaches have been adapted to the problem in an attempt to further improve the solutions obtained by these greedy heuristics.

7.5.1 A GRASP algorithm for the problem

A GRASP (greedy randomised adaptive search procedure) algorithm has been applied to the problem. GRASP is a multi-start meta-heuristic approach that explores the search space from different starting points. The idea of applying GRASP to this problem is that we have two profitability functions, C_1 and C_2, available for this problem. The greedy heuristics based on the function C_2 produces high quality solutions. This function can be utilised in the solution construction stage of a GRASP algorithm. Figure 7-9 presents the pseudo code of the GRASP algorithm used in this research. A total of *max_rep* runs are used and each run consists of a solution construction phase and a local search phase which improves the solution obtained in the construction phase. The solution construction phase is very similar to the greedy algorithm GH_3 except that a parameter α is introduced to control the degree of randomness and greediness. The case $\alpha=0$ corresponds to a random construction process, while $\alpha=1$ is equivalent to the greedy algorithm GH_3. The local search phase is a simple hill-climbing algorithm which repeatedly generates a candidate solution by swapping one facing of two random items and moves to it if a better solution is found. The local search phase stops when the number of total repetitions exceeds a given value *ls_max_rep*. After the preliminary experiments, we set α=0.85, *max_rep*=100 and *ls_max_rep*=n^2.

```
For nrep = 1 to max_rep
    /*  solution construction phase  */
    Start from an empty solution;
    For each item i ($1 \le i \le n$)
        $s_i = L_i$;
        Call proc_qr($s_i$) to obtain the optimal $q_i$ and $r_i$;
        Calculate $C_2$ value for item i;
    Endfor
    Initialise candidate list (CL);
    While (FreeSpace > MinProdSpace and $CL \ne \varnothing$)
        $C_2^{\min} \leftarrow \min\{C_2(i) \mid i \in CL\}$;
        $C_2^{\max} \leftarrow \max\{C_2(i) \mid i \in CL\}$;
        Construct Restricted Candidate List (RCL) by
        $RCL \leftarrow \{\text{item } i \mid i \in CL \ \& \ C_2(i) \ge C_2^{\min} + \alpha(C_2^{\max} - C_2^{\min})$;
        Select an item j from RCL at random;
        $s_j = s_j + 1$;
        Call proc_qr($s_j$) to get the optimal $q_j$ and $r_j$;
        Update the candidate list CL;
        Update $C_2$ value for item j;
    Loop

    /*  local search phase  */
    LocalSearch(ls_max_rep, solution);
    Update best solution found so far;
    nrep = nrep + 1;
Endfor
```

Figure 7-9: A GRASP algorithm for the problem

7.5.2 A simulated annealing algorithm for the problem

A simple simulated annealing algorithm is also used to optimise the problem. The neighbourhood structure is defined by randomly swapping a facing of two items, with the procedure proc_qr(*s*') being called immediately after swapping. The cooling schedule is similar to the one used in the algorithm SAHH_adpt in section 5.3.3. The initial temperature t_s is set a value such that only 85% of inferior moves are accepted and the algorithm stops when the acceptance rate of inferior moves falls to 1%. The temperature is gradually reduced according to Lundy and Mees's cooling function $t \to t/(1+\beta t)$ (Lundy and Mees, 1986) and at each temperature only one iteration is executed. For the purpose of a fair comparison with GRASP, the total number of iterations allowed by SA is set to $K = 100 \times n^2$ (same as the total iterations allowed by GRASP) and the temperature deduction parameter can be calculated by $\beta = (t_s - t_f)/K \times t_s \times t_f$. Once again, the algorithm starts from the solution produced by GH$_3$. Note that although the total number of iterations by GRASP and SA are the same, GRASP may take longer because of the extra time spent during the solution construction phase. This is especially true when the number of iterations of GRASP, *max_rep*, is very large.

7.5.3 Hyper-heuristic approaches for the problem

The hyper-heuristics discussed in section 5.3 were also implemented using the following low-level heuristics:

- **2-opt:** this heuristic swaps one facing of two different random items, i.e. selects two random items *i* and *j*, let $s_i = s_i + 1,\ s_j = s_j - 1$.
- **3-opt1:** this heuristic randomly selects three different items, *i*, *j*, *k*, set $s_i = s_i - 1,\ s_j = s_j - 1,\ s_k = s_k + 1$.
- **3-opt2:** this heuristic randomly selects three different items, *i*, *j*, *k* set $s_i = s_i + 1,\ s_j = s_j + 1,\ s_k = s_k - 1$.
- **4-opt:** this heuristic selects four different random items, deletes one facing of two random items and adds one facing of the other two items.

All the hyper-heuristics started from the same solution generated by GH3 and the approximate computational time was set to a same value that was spent by the multi-start GRG algorithm (see table 7-2). The parameters were the same as those values used in chapters 4 and 5.

7.6 Experimental results

The above algorithms were coded in Microsoft Visual C++ version 6.0 and all experiments were run on a PC Pentium IV 1.8GHZ with 256MB RAM running Microsoft Windows 2000 Professional Version 5. All meta-heuristics were run 30 times for each instance, using different random seeds. The computational results are averaged and presented in the tables 7-3 and 7-4.

It can be seen that the results obtained by GH_3 are very close to the results by different meta-heuristic algorithms. The biggest improvement for the instance BORIN94/6 is only 0.04% ($\frac{347.58\text{-}347.45}{347.45}\times 100\%$). Four algorithms have consistently solved this small instance to optimality over 30 runs (the optimal solution of the numerical example was obtained by a complete search). For the other four instances, the biggest improvements over the initial solutions are 0.17%, 0.16%, 0.16% and 0.06% respectively. Similar results were obtained even when the algorithms were given much more computational time or more repetitions. For the instance FRESH2, three algorithms (GRASP, SAHH, TSSAHH) consistently produced the same solution over 30 runs. We have a strong feeling that these results are already very close to the optima. However, this is only conjecture and cannot be proven due to the NP-Hard nature of the problem.

Among these algorithms, the GRASP algorithm performed well when compared with the multi-start GRG algorithm and the general SA. It was only marginally outperformed by multi-GRASP on instance FRESH3. However, on larger problem instances, both GRASP and SA consumed more computational time than the multi-

Table 7-3: A comparison of different algorithms on five fresh produce instances

	BORIN94/6			**FRESH2**			**FRESH3**			**FRESH4**			**FRESH5**		
n	6			18			32			49			64		
	av. obj.	**stdev**	**av. cpu**	**av. obj.**	**stdev**	**av. cpu**	**av. obj.**	**stdev**	**av. cpu**	**av. obj.**	**stdev**	**av. cpu**	**av. obj.**	**stdev**	**av. cpu**
Initial (GH3)	347.45	--	0.02	1131.64	--	0.02	2053.71	--	0.03	3159.17	--	0.06	4384.62	--	0.09
Multi-Start GRG	347.45	0.00	3.2	1129.6	0.00	73.6	2056.46	0.97	74.3	3163.98	0.51	179.2	4387.16	0.43	209.7
GRASP	**347.58**	0.00	2.79	**1133.51**	0.00	23.64	2056.43	0.17	78.90	3164.14	0.34	245.30	4387.23	0.31	438.99
SA	**347.58**	0.00	2.23	1133.22	0.36	19.11	2055.04	0.61	72.24	3163.81	0.41	226.70	4387.16	0.51	401.20
SAHH	**347.58**	0.00	3.70	**1133.51**	0.00	61.26	2056.93	0.27	60.62	3164.18	0.27	148.91	**4387.42**	0.37	185.09
CFHH	347.46	0.04	3.20	1131.65	0.04	73.61	2053.71	0.00	74.31	3159.17	0.00	179.21	4384.62	0.00	209.70
CFSAHH	347.55	0.06	3.78	1131.79	0.28	70.09	2054.24	0.81	110.36	3160.69	2.04	223.17	4386.27	1.32	185.28
TSHH	347.56	0.05	3.20	1131.64	0.00	73.61	2053.71	0.00	74.31	3159.23	0.33	179.21	4384.62	0.00	209.71
TSSAHH	**347.58**	0.00	3.72	**1133.51**	0.00	62.50	**2057.09**	0.17	56.31	**3164.21**	0.25	135.55	4387.41	0.32	185.29

av. obj.: average objective value of 30 runs
stdev: standard deviation of 30 runs
av. cpu: average CPU time spent

Table 7-4: Robustness of different algorithms

	BORIN94/6				**FRESH2**				**FRESH3**				**FRESH4**				**FRESH5**			
n	6				18				32				49				64			
	av. obj.	**best**	**worst**	**stdev**	**av. obj.**	**best**	**worst**	**stdev**	**av. obj.**	**best**	**worst**	**stdev**	**av. obj.**	**best**	**worst**	**stdev**	**av. obj.**	**best**	**worst**	**stdev**
Multi-Start GRG	347.45	347.45	347.45	0.00	1129.6	1129.60	1129.60	0.00	2056.46	2057.15	2055.17	0.97	3163.98	3164.59	3163.33	0.51	4387.16	4387.73	4386.66	0.43
GRASP	347.58	347.58	347.58	0.00	1133.51	1133.51	1133.51	0.00	2056.43	2056.74	2056.33	0.17	3164.14	3164.60	3163.52	0.34	4387.23	4387.92	4386.65	0.31
SA	347.58	347.58	347.58	0.00	1133.22	1133.51	1132.32	0.36	2055.04	2055.76	2053.71	0.61	3163.81	3164.6	3163.21	0.41	4387.16	4387.72	4385.85	0.51
SAHH	347.58	347.58	347.58	0.00	1133.51	1133.51	1133.51	0.00	2056.93	2057.16	2056.49	0.27	3164.18	3164.60	3163.27	0.27	4387.42	4387.92	4386.71	0.37
CFHH	347.46	347.58	347.45	0.04	1131.65	1131.85	1131.64	0.04	2053.71	2053.71	2053.71	0.00	3159.17	3159.17	3159.17	0.00	4384.62	4384.62	4384.62	0.00
CFSAHH	347.55	347.58	347.45	0.06	1131.79	1132.68	1131.64	0.28	2054.24	2055.83	2053.71	0.81	3160.69	3164.37	3159.17	2.04	4386.27	4387.79	4384.62	1.32
TSHH	347.56	347.58	347.45	0.05	1131.64	1131.64	1131.64	0.00	2053.71	2053.71	2053.71	0.00	3159.23	3161.00	3159.17	0.33	4384.62	4384.62	4384.62	0.00
TSSAHH	347.58	347.58	347.58	0.00	1133.51	1133.51	1133.51	0.00	2057.09	2057.16	2056.49	0.17	3164.21	3164.60	3163.59	0.25	4387.41	4387.92	4386.94	0.32

av. obj.: average objective value among 30 runs
best: best objective value among 30 runs
worst: worst objective value among 30 runs
stdev: standard deviation of 30 runs

start GRG. Comparing the different hyper-heuristics, both CFHH and TSHH were unable to improve the initial solution or only achieved a very small improvement. However, the performances of both algorithms were improved when a simulated annealing acceptance criterion was introduced (corresponding to CFSAHH and TSSAHH respectively).

In general, two types of hyper-heuristics performed best among all the algorithms. TSSAHH out performed all of the other algorithms for four instances and was only marginally beaten by SAHH on the remaining one instance. SAHH performed well and obtained best results on three instances (BORIN94/6, FRESH2 AND FRESH5). Even for the other two instances, it ranked as the second best algorithm and found solutions that are very close to the best solutions. This is a very good performance considering that the parameters of the algorithm used in this paper are exactly the same as those used in the previous applications from chapters 4 and 5.

7.7 Summary and Remarks

This chapter has investigated heuristic and meta-heuristic approaches for the optimisation of the fresh produce shelf space allocation model in chapter 6. A single-item inventory problem was firstly analysed and solved by a binary search procedure. Based on this, four greedy heuristic methods were then developed for the multi-item problems. The experimental results have shown that, compared with the multi-start GRG algorithm and meta-heuristics used in this chapter, these greedy heuristic methods are very efficient and capable of producing high quality solutions in much shorter time. Among the four greedy heuristics, the best algorithm is the one that repeatedly allocates shelf space to the item with the largest C_2 value. The solutions created by this heuristic were taken as initial solutions and further improved by several meta-heuristic approaches, including a GRASP algorithm, a general simulated annealing and three simulated annealing hyper-heuristics.

All meta-heuristics can only achieve (if at all) small improvements over the initial solutions. Increasing the computational time and the number of iterations does not produce significantly better results. It seems that the results obtained by our algorithm are already close to the optimal solutions. Among all the meta-heuristic approaches, two types of hyper-heuristics, SAHH and TSSAHH outperformed all other algorithms in terms of solution quality while using the same (or even less) computational time. This includes the multi-start GRG algorithm, the GRASP algorithm and a general simulated annealing algorithm. Considering the successes on three different, while related, problems (bin packing in chapter 4, general shelf space allocation problem in chapter 5 and fresh food inventory and shelf space allocation in this chapter), hyper-heuristics with the assistance of the simulated annealing appears to be a very promising and generic search technique for the other similar combinatorial optimisation problems.

CHAPTER 8. CONCLUSIONS AND FUTURE WORK

The problem of shelf space allocation has recently received increasing attention due to fierce competition in the retail industry. In order to improve efficiency and financial performance, retailers are willing to adopt more sophisticated systems in retail decision making processes. Shelf space allocation is an area that can increase a store's sales and increase customer satisfaction. In this thesis, we have developed and investigated practical models and efficient optimisation techniques for shelf space allocation problems. The main work in this thesis is in two main areas:

8.1 From the Shelf Space Allocation Perspective

This thesis has introduced and studied two mathematical models for two types of shelf space allocation problems.

A practical model for general products shelf space allocation

As stated in the aims and scopes, the overall aim of this research is to investigate novel approaches that can be used to generate automated, optimised planograms. Bearing this in mind, the thesis firstly discussed several issues and potential constraints that are involved in shelf space allocation decisions. Due to the diverse nature of the problem an abstracted problem has been devised and a simplified, while practical, model has been proposed with the advantages of practicability and ease of implementation. It has been shown that this model is an extension of the bounded multi-knapsack problem, a problem which is NP-Hard. A two-stage relaxation method was used to obtain an upper bound of the model, by which one can effectively compare and evaluate the quality of solutions obtained by different algorithms. Besides, the gap between the upper bound and the current solution can provide a user with a useful estimation of maximal possible improvement over the current solution.

A model for managing shelf space and inventory for fresh produce

The thesis has also studied a special shelf space allocation problem for fresh produce. This problem is particularly important for many retailers due to the increasing demand for fresh food. Because of very short shelf-lifetime, all fresh produce has to be sold before their expiry dates in order to avoid losses. This poses a real challenge for retailers. In this thesis a shelf space allocation model for fresh produce has been developed in integration with a single-period inventory model. The thesis contributes to the literature as this is the first fresh produce model that integrates shelf space allocation and inventory control. In formulating the model, the thesis, for the first time, introduces the term 'freshness condition' as a factor that influences demand for the product. It differs from the existing deteriorating models in the literature, which usually assume that products have a random lifetime but a non-decaying utility. Further study of this model has shown that the size of the search space can be reduced by decomposing the problem into a knapsack problem and a single inventory problem which can be efficiently optimised by a binary search procedure. We consider this as a major contribution of the thesis.

8.2 From Meta-heuristics Perspective

Both shelf space allocation problems studied in this thesis are closely related to the knapsack problem and the bin packing problem, which are NP-Hard. There is no known polynomial-time bound algorithm that can guarantee to solve them to optimality. In this thesis we have focused on heuristic and meta-heuristic approaches to search for the near-optimal (if not optimal) solutions for the problems. There are potentially many meta-heuristics (and their variants) available for solving NP-Hard combinatorial optimisation problems. Among them, hyper-heuristic is a generic approach that has recently attracted increasing research attention and has been successfully applied to several difficult scheduling problems. Instead of applying hyper-heuristics directly to the shelf space allocation problem, we initially tested hyper-heuristics on the well-known bin packing problem in comparison with other state-of-the-art algorithms. If this algorithm performs well on the bin packing problem (which it did), it is very likely that the hyper-heuristic is also suitable for the shelf space allocation problem because of the close relationship between the two problems.

Proposing a simulated annealing hyper-heuristic

The existing hyper-heuristics either explicitly or implicitly focus on “choosing”, at each decision point, appropriate low-level heuristics according to their previous performance. However, these deterministic heuristic selection strategies may not always be suitable because of the stochastic nature of some low-level heuristics. In this thesis, a simulated annealing algorithm was incorporated into the current hyper-heuristics in order to alter (and improve) the acceptance criteria of heuristic moves.

The resulting algorithm, called the simulated annealing based hyper-heuristic, was tested on the one-dimensional bin packing problem and applied to two shelf space allocation problems. Below are some observations and conclusions from the three applications.

Applying hyper-heuristics to 1D bin packing problem

The thesis, for the first time, applied hyper-heuristics to the 1D bin packing problem to test its performance. The reasons we chose the 1D bin packing problem are as follows: Unlike some classical problems (such as bin packing, TSP, timetabling, stock cutting, etc.) which have large benchmark data sets available in the literature, shelf space allocation problems do not have benchmark data available that allow us to compare the proposed algorithm with other approaches. Furthermore, the thesis has used new models because we believe the current shelf space allocation models in the literature are not practical for the production of automated planograms. It would have taken a considerable amount of work if we compared our proposed algorithms with every other search technique. However, shelf space allocation is closely related to bin packing, which does have a large set of benchmark data sets available. It is assumed that if the simulated annealing hyper-heuristics could produce high quality solutions for the bin packing problem, there is a reasonable possibility that it will perform well for shelf space allocation problems as well.

The experimental results on the bin packing problem have shown that the introduction of simulated annealing into the choice function based hyper-heuristics did improve its performance. However, it was inferior to the simulated annealing hyper-heuristic with random heuristic selection strategy, which also beat the grouping genetic algorithm (GGA) and a branch-and-bound based method (BISON), both in terms of solution quality and computational time. The simulated annealing hyper-heuristic produced better quality solutions than a variable neighbourhood search (VNS) algorithm in terms of solution quality, while it needed more computational time. Overall, the simulated annealing hyper-heuristic solved around 1340 instances out of 1370 to optimality on average. For those instances that were not solved to optimality, they were only 1 bin away from the optimum. The success of the simulated annealing hyper-heuristic on the bin packing problem encouraged us to apply it to two shelf space allocation problems. Some other meta-heuristic approaches have also been implemented for the purposes of comparison. Below are some observations and conclusions.

Optimisation of general shelf space allocation problem

Several hyper-heuristics have been implemented and applied to the general shelf space allocation problem. The thesis especially investigated simulated annealing hyper-heuristics and compared them with two conventional simulated annealing algorithms and other types of hyper-heuristics without the assistance of a simulated annealing acceptance criterion. For the twelve problem instances, it was observed that the performance of a conventional simulated annealing is heavily dependent on the choice of neighbourhood structure and the optimal neighbourhood structure may change from one problem instance to another. However, simulated annealing hyper-heuristics seem not to be parameter-sensitive and performed much better than the conventional SA algorithms in terms of both average solution quality and results consistency. Again, it has been shown that the performance of the existing two types of hyper-heuristics, choice function based hyper-heuristics and tabu search hyper-heuristics, can be improved by introducing a SA acceptance criterion. However, the best algorithm turned out to be the simulated annealing hyper-heuristics with a uniform heuristic selection probability. This type of simulated annealing hyper-heuristics produced solutions that were over 98% of the upper bounds for every tested instance.

Optimisation of fresh produce shelf space allocation and inventory

The thesis has also investigated the shelf space allocation problem specifically for fresh produce. Several heuristic and meta-heuristic approaches have been applied to five problem instances of different sizes, and their performance was compared. This comparison included four constructive heuristics, an improved generalised reduced gradient (GRG) algorithm, SA, GRASP and five versions of hyper-heuristics. The experimental results have shown that the proposed constructive heuristics are very efficient and produce high quality solutions. All meta-heuristics can only give a small improvement even with a very large computational time. This is probably because that the solutions obtained by these algorithms may be very close to the upper bounds. Among all meta-heuristics, simulated annealing hyper-heuristic and the tabu assisted simulated annealing hyper-heuristic algorithms are consistently superior to the other algorithms.

Overall, we have studied different optimisation approaches over three different, while related, space allocation problems. The research has mainly focused on the hyper-heuristic techniques that have recently been attracting research interest in scheduling and general optimisation. Across all three problems, it has been observed that the simulated annealing hyper-heuristics outperformed both the general meta-heuristic approaches (simulated annealing, grouping genetic algorithm, variable neighbourhood search, multi-start GRG and GRASP) and other existing hyper-heuristics for these problems. Introducing a SA acceptance criterion into the current hyper-heuristic framework has been beneficial. However, the simulated annealing hyper-heuristics with random heuristic selection performed better than the simulated annealing hyper-heuristics assisted by a choice function or by a tabu list. This is probably because the stochastic nature of SA is not complimentary with deterministic strategies used both in choice function and tabu list based hyper-heuristics.

8.3 Further work

With regard to the future work on the problem modelling, it will be interesting to integrate the research results (models and algorithms) into current planogram software. One can also extend the current shelf space allocation models to two (or even three) dimensions, in which case extra constraints might have to be considered. However, one would not have to spend a lot of time adapting hyper-heuristics to the new models. As the new problem is still very similar to the previous problems, a user would only need to slightly change the low-level heuristics used in the previous model in order to adapt to the new problem. This further shows the advantages of a hyper-heuristic methodology.

Another interesting direction would be the integration of the automated planograms with the newly emerging RFID (Radio Frequency Identification) technologies (RFID Journal, 2005). RFID is a new ID technology that may eventually replace bar codes. A typical RFID system consists of a RFID *tag* which is attached to a product, and stores data, and a RFID *reader* which retrieves and updates product information stored on the tag. Two major advantages of RFID over bar codes are: 1. the product information can be retrieved and sent to a computer automatically when the products pass a reader (bar codes need to be scanned manually). 2. the data on a RFID tag can be updated dynamically by the reader while the data on the bar code cannot normally be altered. This functionality is very useful in just-in-time manufacturing, supply chain management and inventory management when one wants to track the physical location of products. RFID technology has been used in many companies in different industries. In retailing, Wal-Mart is a pioneer in adopting this technology. The integration of planogram software with RFID technology will make it possible to automate or semi-automate product replenishment from the storeroom to the shop floor shelves. Currently, this process is completely manual and is not efficient.

From an optimisation perspective, it would be interesting to further improve the current simulated annealing hyper-heuristics in a number of ways. Firstly, it has been observed in this thesis that the deterministic heuristic selection hyper-heuristic approaches (CFSAHH and TSSAHH) do not perform as well as the simulated annealing hyper-heuristics with uniform heuristic selection. The most likely reason is that the probabilistic nature of SA is not complimentary with deterministic heuristic selection strategies. In the future it may be worthwhile investigating some stochastic heuristic selection approaches under the simulated annealing hyper-heuristic framework. Secondly, throughout the three problems, it has been observed that the design of the low-level heuristics is crucial in influencing the performance of the hyper-heuristics. In the current hyper-heuristics the low-level heuristics correspond to some simple neighbourhood move strategies. Several issues are not yet clear when designing these heuristics, such as how to balance the greediness and randomness of a heuristic, and how the low-level heuristics should be designed to help guide the search for promising areas when the objective function fails to guide the search. Further research should be carried out in order to gain a better understanding of these issues. Another interesting research direction is in designing a hyper-heuristic framework which manages several different types of local search explorers (low-level heuristics). These local search explorers do not have to be simple neighbourhood functions. They can be different search strategies (SA, TS and VNS, for example) or the same strategy with different parameters (for example several SA algorithms with different parameters). The local search explorers both compete and cooperate during the process of problem solving. Finally, the proposed simulated annealing hyper-heuristic may be improved by using more complicated temperature cooling strategies (for example, by allowing reheating).

Furthermore, the performance of the current meta-heuristics may be improved by hybridising with some exact methods, such as linear programming, branch-and-bound, dynamic programming, etc. Meta-heuristics are believed to be able explore a large search space within a short time while exact methods can explore a specific small area exhaustively. Hybridisation of them may lead to a better quality solution within reasonable computation time.

REFERENCES

Aarts, E. H. L. and Korst, J., 1998. Simulated Annealing and Boltzman Machines, Wiley.

Aarts, E. H. L. and van Laarhoven, P. J. M., 1985. Statistical Cooling: A General Approach to Combinatorial Optimization Problems, Phillips Journal of Research. 40, 193-226.

Abad, P. L., 1996. Optimal Pricing and Lot-sizing Under Conditions of Perishability and Partial Backordering, Management Science. 42, 1093-1104.

Abadie, J. and Carpentier, J., 1969. Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints. In: Fletcher,R. (Ed.), Optimization, London, Academic Press, pp. 37-47.

Abramson, D., 1991. Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms, Management Science. 37, 98-113.

Ahuja, R. K., Orlin, J. B. and Sharma, D., 2000. Very Large Scale Neighbourhood Search, International Transaction in Operational Research. 7, 301-317.

Ayob, M. and Kendall, G., 2003. A Monte Carlo Hyper-Heuristic to Optimise Component Placement Sequencing for Multi Head Placement Machine, In Proceedings of the International Conference on Intelligent Technologies, InTech'03, Chiang Mai, Thailand, Dec 17-19, 132-141.

Bäck, T., 1996. Evolutionary Algorithms in Theory and Practice, Oxford University Press.

Bäck, T., Fogel, D. and Michalewicz, Z., 1997. Handbook of Evolutionary Computation, Institute of Physics Publishing and Oxford University Press.

Bai, R. and Kendall, G., 2003. An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics, In the proceedings of the 5th Metaheuristics International Conference (MIC 2003), Kyoto, Japan, Aug. 25-28.

Bai, R. and Kendall, G., 2005a. A Model for Fresh Produce Shelf Space Allocation and Inventory Management with Freshness Condition Dependent Demand, Accepted for publication in the INFORMS Journal on Computing.

Bai, R. and Kendall, G., 2005b. An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics, in: Ibaraki, T., Nonobe, K., and Yagiura, M. (Eds.), Metaheuristics: Progress as Real Problem Solvers - (Operations Research/Computer Science Interfaces, Vol. 32), Berlin, Heidelberg, New York, Springer, pp. 87-108.

Bai, R. and Kendall, G., 2005c. A Multi-heuristic Simulated Annealing for the One-dimensional Bin Packing Problem, Submitted to European Journal of Operational Research.

Bai, R. and Kendall, G., 2005d. Heuristic and Meta-heuristics for the Optimisation of Fresh Produce Inventory Control and Shelf Space Allocation Problem, Submitted to Journal of Operations Research Society.

Baker, R. C. and Urban, T. L., 1988. A Deterministic Inventory System with an Inventory-Level-Dependent Demand Rate, Journal of the Operational Research Society. 39, 823-831.

BBC Business, 2003. Asda Overtakes Sainsbury's, http://news.bbc.co.uk/1/hi/business/3112689.stm.

BBC Business, 2005. Tesco Profits Break through 2 Billion Pounds, http://news.bbc.co.uk/1/hi/business/4435339.stm.

Beasley, D., Bull, D. R. and Martin, R. R., 1993. An Overview of Genetic Algorithms: Part 1, Fundamentals, University Computing. 15, 58-69.

Bellman, R., 1957. Dynamic Programming, Princeton University Press.

Belov, G. and Scheithauer, G., 2004. A Branch-and-Cut-and-Price Algorithm for One-Dimensional Stock Cutting and Two-Dimensional Two-Stage Cutting, European Journal of Operational Research, in Press, Available online October 2004.

Ben-Ameur, W., 2004. Computing the Initial Temperature of Simulated Annealing, Computational Optimization and Applications. 29, 369-385.

Ben-Daya, M. and Raouf, A., 1993. On the Constrained Multi-item Single-period Inventory Problem, International Journal of Operations & Production Management. 13, 104-112.

Birrattari, M., Paquete, L., Stutzle, T. and Varrentrapp, K., 2001. Classification of Metaheuristics and Design of Experiments for the Analysis of Components, Technical Report AIDA-01-05, FG Intellektik, FB Informatik, TU Darmstadt, Germany.

Blum, C. and Roli, A., 2003. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, ACM Computing Surveys. 35, 268-303.

Borin, N. and Farris, P., 1995. A Sensitivity Analysis of Retailer Shelf Management Models, Journal of Retailing. 71, 153-171.

Borin, N., Farris, P. W. and Freeland, J. R., 1994. A Model for Determining Retail Product Category Assortment and Shelf Space Allocation, Decision Sciences. 25, 359-384.

Bouleimen, K. and Lecocq, H., 2003. A New Efficient Simulated Annealing Algorithm For the Resource-constrained Project Scheduling Problem and Its

Multiple Mode Version, European Journal of Operational Research. 149, 268-281.

Bremermann, H. J., 1962. Optimisation Through Evolution and Re-Combination. In: Yovits,M., Sawbi,G., and Goldstein,G. (Eds.), Self-Organising Systems, Washington, Spartan.

Bretthauer, K. M. and Shetty, B., 2002. The Nonlinear Knapsack Problem - Algorithms and Applications, European Journal of Operational Research. 138, 459-472.

Bullnheimer, B., Hartl, R. F. and Strauss, C., 1999. A New Rank-based Version of the Ant System, Central European Journal for Operations Research and Economics. 7, 25-38.

Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., and Schulenburg, S., 2003a. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In: Glover F. and Kochenberger,G. (Eds.), Handbook of Meta-Heuristics, Kluwer, pp. 457-474.

Burke, E., Kendall, G., O'Rrien, R., Redrup, D., and Soubeiga, E., 2003b. An Ant Algorithm Hyper-Heuristic, Proceedings of the Fifth Metaheuristics International Conference (MIC 2003), Kyoto Japan, August 25-28.

Burke, E. and Causemacker, P. D., 2003. Practice and Theory of Automated Timetabling IV, Springer-Verlag Lecture Notes in Computer Science, The 4th International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers, Springer LNCS 2740.

Burke, E. and Kendall, G., 1999. Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem, Proceedings of WMC '99 : World Manufacturing Congress, Durham, UK, 27-30 September, 51-57.

Burke, E. and Kendall, G., 2005. Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Kluwer Academic Publishers, Boston, Dordrecht, London.

Burke, E., Kendall, G. and Soubeiga, E., 2003c. A Tabu-Search Hyperheuristic for Timetabling and Rostering, Journal of Heuristics. 9, 451-470.

Burke, E., McCollum, B., Meisels, A., Petrovic, S. and Qu, R., 2006. A Graph-Based Hyper Heuristic for Educational Timetabling Problems, Accepted for Publication in the European Journal of Operational Research.

Burke, E., Petrovic, S. and Qu, R., 2005. Case Based Heuristic Selection for Timetabling Problems, Accepted for Publication in the Journal of Scheduling.

Buttle, F., 1984. Merchandising, European Journal of Marketing. 18, 104-123.

Campo, K., Gijsbrechts, E., Goossens, T. and Verhetsel, A., 2000. The Impact of Location Factors on the Attractiveness and Optimal Space Shares of Product Categories, International Journal of Research in Marketing. 17, 255-279.

Carnevali, L., Coletti, L. and Patarnello, S., 1985. Image Processing by Simulated Annealing, IBM Journal of Research and Development. 29, 569-579.

Carter, M. W., 1986. A Survey of Practical Applications of Examination Timetabling Algorithms, Operations Research. 34, 193-201.

Carter, M. W. and Laporte, G., 1996. Recent Developments in Practical Examination Timetabling. In: Burke,E. and Ross,P. (Eds.), The Practice and Theory of Automated Timetabling: Selected Papers (ICPTAT '95), Lecture Notes in Computer Science, Vol.1153, Springer-Verlag, pp. 3-21.

Chen, S. and Luk, B. L., 1999. Adaptive Simulated Annealing for Optimization in Signal Processing Applications, Signal Processing. 79, 117-128.

Coffman, E. G., Garey, M. R., and Johnson, D. S., 1997. Approximation Algorithms for Bin Packing. In: Hochbaum,D.S. (Ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing, pp. 46-93.

Cohn, H. and Fielding, M., 1999. Simulated Annealing: Searching for an Optimal Temperature Schedule, SIAM Journal on Optimization. 7, 779-802.

Coley, D. A., 1999. An Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific Publishing Co. Pte. Ltd.

Connolly, D. T., 1990. An Improved Annealing Scheme For the QAP, European Journal of Operational Research. 46, 93-100.

Cook, S.A., 1971. The Complexity of Theorem-proving Procedures, In the Proceedings of 3rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 151-158.

Corstjens, M. and Doyle, P., 1981. A Model for Optimaizing Retail Space Allocations, Management Science. 27, 822-833.

Cowling, P. and Chakhlevitch, K., 2003. Hyperheuristics for Managing a Large Collection of Low Level Heuristics to Schedule Personnel, In the Proceeding of the 2003 IEEE Congress on Evolutionary Computation (CEC'2003), Canberra, Australia, 8-12 December, 1214-1221.

Cowling, P., Kendall, G., and Han, L., 2002. An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. Proceedings of Congress on Evolutionary Computation (CEC2002), Hilton Hawaiian Village Hotel, Honolulu, Hawaii, Nov.18-22 pp. 1185-1190.

Cowling, P., Kendall, G., and Soubeiga, E., 2001. A Parameter-free Hyperheuristic For Scheduling a Sales Summit. In the Proceedings of the 4th Metaheuristic International Conference (MIC 2001), Porto, Portugal, July 16-20 pp. 16-20.

Cox, K., 1970. The Effect of Shelf Space Upon Sales of Branded Products, Journal of Marketing Research. 7, 55-58.

Csondes, T., Kotnyek, B. and Szabo, J. Z., 2002. Application of Heuristic Methods for Conformance Test Selection, European Journal of Operational Research. 142, 203-218.

Curhan, R., 1972. The Relationship Between Space and Unit Sales in Supermarkets, Journal of Marketing Research. 9, 406-412.

Curhan, R., 1973. Shelf Space Allocation and Profit Maximization in Mass Retailing, Journal of Marketing. 37, 54-60.

Dantzig, G. B., 1951. Maximization of a Linear Function of Variables Subject to Linear Inequalities. In: Koopmans,T.C. (Ed.), Activity Analysis of Production and Allocation, New York, Wiley, pp. 339-347.

Dantzig, G. B., 1963. Linear Programming and Extensions, Princeton University Press, Princeton.

Davis, L. D., 1987. Genetic Algorithms and Simulated Annealing, Pitman, London.

Davis, L. D., 1991. Handbook of Genetic Algorithms, Van Nostrand Reinhold.

Deneubourg, J.-L., Aron, S., Goss, S. and Pasteels, J.-M., 1990. The Self-organising Exploratory Pattern of the Argentine Ant, Journal of Insect Behavior. 3, 159-168.

Desmet, P. and Renaudin, V., 1998. Estimation of Product Category Sales Responsiveness to Allocated Shelf Space, International Journal of Research in Marketing. 15, 443-457.

Dorigo, M., Vittorio, M. and Alberto, C., 1996. Optimization by a Colony of Cooperating Agents, IEEE Transactions on Systems, Man and Cybernetics - Part B : Cybernetics. 26, 29-41.

Dorigo, Marco, 1992. Optimization, Learning, and Natural Algorithms, PhD Thesis, Politecnico di Milano, Italy.

Dorigo, M. and Gambardella, L. M., 1997a. Ant Colonies for the Traveling Salesman Problem, BioSystems. 43, 73-81.

Dorigo, M. and Gambardella, L. M., 1997b. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transactions on Evolutionary Computation. 1, 53-66.

Dorigo, M. and Maniezzo, V., 1996. The Ant System: Optimisation by a Colony of Cooperating Agents, IEEE Transactions on Systems, Man and Cybernetics - Part B. 26, 29-41.

Dorigo, M. and Stutzle, T., 2003. The Ant Colony Optimisation Metaheuristic: Algorithms, Applications and Advances. In: Glover F. and Kochenberger,G. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 457-474.

Dowsland, K. A., 1993. Some Experiments with Simulated Annealing Techniques for Packing Problems, European Journal of Operational Research. 68, 389-399.

Dowsland, K. A., 1995. Simulated Annealing. In: Reeves,C.R. (Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, pp. 21-69.

Dreze, X., Hoch, S. J. and Purk, M. E., 1994. Shelf Management and Space Elasticity, Journal of Retailing. 70, 301-326.

Falkenauer, E., 1996. A Hybrid Grouping Genetic Algorithm for Bin Pacing, Journal of Heuristics. 2, 5-30.

Falkenauer, E., 1998. Genetic Algorithms and Grouping Problems, John Wiley & Sons Ltd.

Feo, T. A. and Resende, M. G. C., 1989. A Probabilistic Heuristic for a Computational Difficult Set Covering Problem, Operations Research Letters. 8, 67-71.

Feo, T. A. and Resende, M. G. C., 1995. Greedy Randomized Adaptive Search Procedures, Journal of Global Optimization. 6, 109-133.

Fisher, H. and Thompson, G. L., 1961. Probabilistic Learning Combinations of Local Job-shop Scheduling Rules, Factory Scheduling Conference, Carnegie Institute of Technology. May, 10-12.

Fleszar, K. and Hindi, K. S., 2002. New Heuristics for One-dimensional Bin-packing, Computers & Operations Research. 29, 821-839.

Fogel, D. B., 1998. Evolutionary Computation: The Fossil Record, IEEE Press.

Fogel, D. B., 2000. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press.

Forrest, S., 1993. Genetic Algorithms: Principles of Natural Selection Applied to Computation. Science. 261, 872-878.

Fraser, A. S., 1957. Simulation of Genetic Systems by Automatic Digital Computers, Aust. J. Biol. Sci. 10, 484-499.

Fujiwara, O. and Perera, U. L. J. S. R., 1993. EOQ Models for Continuously Deteriorating Products Using Linear and Exponential Penalty Costs, European Journal of Operational Research. 70, 104-114.

Gabriele, G. A. and Ragsdell, K. M., 1977. The Generalized Reduced Gradient Method: A Reliable Tool for Optimal Design, AMSE Journal of Engineering for Industry. 99, 384-400.

Gambardella, L. M. and Dorigo, M., 2000. Ant Colony System Hybridised with a New Local Search for the Sequential Ordering Problems, INFORMS Journal on Computing. 12, 237-255.

Garey, M. R. and Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman.

Gendreau, M., 2002. Recent Advances in Tabu Search. In: Ribeiro,C.C. and Hansen,P. (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, pp. 369-377.

Gendreau, M., Hertz, A. and Laporte, G., 1994. A Tabu Search Heuristic for the Vehicle Routing Problem, Management Science. 40, 1276-1290.

Giri, B. C., Pal, S., Goswami, A. and Chaudhuri, K. S., 1996. An Inventory Model for Deteriorating Items with Stock-dependent Demand Rate, European Journal of Operational Research. 95, 604-610.

Glover F. and Laguna, M., 1995. Tabu Search. In: Reeves,C.R. (Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, pp. 21-69.

Glover, F. and Kochenberger, G. A., 2003. Handbook of Meta-Heuristics, Kluwer, ISBN: 1-4020-7263-5.

Glover, F., Laguna, M., and Marti, R., 2003. Scatter Search and Path Relinking. In: Glover,F. and Kochenberger,G.A. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 1-37.

Glover, F., 1977. Heuristics for Integer Programming using Surrogate Constraints, Decisions Science, 8, 155-166.

Glover, F., 1989. Tabu Search - Part I, ORSA Journal on Computing, 1, 190-206.

Glover, F., 1990. Tabu Search - Part II, ORSA Journal on Computing, 2, 4-32.

Glover, F. and Laguna, M., 1997. Tabu Search, Kluwer Academic Publishers.

Goldberg, D. E., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley.

Goyal, S. K. and Giri, B. C., 2001. Recent Trends in Modeling of Deteriorating Inventory, European Journal of Operational Research. 134, 1-16.

Greene, J. W. and Supowit, K. J., 1986. Simulated Annealing Without Rejected Moves, Transactions on Computer-Aided Design. 5, 221-228.

Gruen, T. and Shah, R., 2000. Determinants and Outcomes of Plan Objectivity and Implementation in Category Management Relationships, Journal of Retailing. 76, 483-510.

Gupta, J. N. D. and Ho, J. C., 1999. A New Heuristic Algorithm for the One-dimensional Bin-packing Problem, Production Planning & Control. 10, 598-603.

Gutin, G., 1999. Exponential Neighbourhood Local Search for the Traveling Salesman Problem, Computer & Operations Research. 26, 313-320.

Han, L., Kendall, G. and Cowling, P., 2002. An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem, 4th Asia-Pacific Conference on Simulated Evolution and Learning [SEAL'02], 267-271.

Hansen, P. and Maldenovic, N., 2001. Variable Neighbourhood Search: Principles and Applications, European Journal of Operational Research. 130, 449-467.

Hansen, P. and Maldenovic, N., 2003. Variable Neighbourhood Search. In: Glover F. and Kochenberger,G. (Eds.), Handbook of Meta-Heuristics, Kluwer, pp. 145-184.

Hart, C. and Davies, M., 1996. The Location and Merchandising of Non-food in Supermarkets, International Journal of Retail & Distribution Management. 24, 17-25.

Hart, E., Ross, P. and Nelson, J. A., 1998. Solving a Real-World Problem Using An Evolving Heuristically Driven Schedule Builder, Evolutionary Computing. 6, 61-80.

Hart, W. E., Krasnogor, N., and Smith, J. E., 2003. Recent Advances in Memetic Algorithms and Related Search Technologies, Springer.

Henderson, D., Jacobson, S. H., and Johnson, A. W., 2003. The Theory and Practice of Simulated Annealing. In: Glover F. and Kochenberger,G. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 287-319.

Hillier, F. S. and Lieberman, G. J., 2005. Introduction to Operations Research, McGraw-Hill, 8th ed. ISBN: 0-07-252744-7.

Holland, J. H., 1975. Adaptation in Natural and Artificial Systems, University of Michigan Press.

Hollier, R. H. and Mak, K. L., 1983. Inventory Replenishing Policies for Deteriorating Items in a Declining Market, International Journal of Production Research. 21, 813-826.

Hwang, H., Choi, B. and Lee, M.-J., 2005. A Model for Shelf Space Allocation and Inventory Control Considering Location and Inventory Level Effects on Demand, International Journal of Production Economics. In Press.

Ibarra, O. H. and Kim, C. E., 1975. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems, Journal of ACM. 22, 463-468.

Ibbotson, J., 2002. Personal Correspondence, Retail Vision.

Jain, K. and Silver, E. A., 1994. Lot Sizing for a Product Subject to Obsolescence or Perishability, European Journal of Operational Research. 75, 287-295.

Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C., 1989. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning, Operations Research. 37, 865-892.

Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C., 1991. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Colouring and Number Partitioning, Operations Research. 39, 378-406.

Johnson, M., 2002. Personal Correspondence, Tesco.

Kaelbling, L. P., Littman, M. L. and Moore, A. W., 1996. Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research. 4, 237-285.

Kar, S., Bhunia, A. K. and Maiti, M., 2001. Inventory of Multi-deteriorating Items Sold From Two Shops Under Single Management with Constraints on Space and Investment, Computers & Operations Research. 28, 1203-1221.

Kendall, G. and Mohamad, M., 2004a. Channel Assignment Optimisation Using a Hyper-heuristic, In the Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems (CIS), Singapore, 1-3 December 2004, 790-795.

Kendall, G. and Mohamad, M., 2004b. Channel Assignment In Cellular Communication Using A Great Deluge Hyper-heuristic, In the Proceedings of the 2004 IEEE International Conference on Network (ICON2004), Singapore, 16-19 November 2004, 769-773.

Kendall, G. and Mohd Hussin, N., 2005. An Investigation of a Tabu Search Based Hyper-heuristic for Examination Timetabling. In: Kendall,G., Burke,E., and Petrovic,S. (Eds.), Selected Papers from MISTA 2003, Kluwer Academic Publisher, 309-328.

Kendall, G. and Mohd Hussin, N., 2004a. An Investigation of a Tabu Search Based Hyper-heuristic for Examination Timetabling. In: Kendall,G., Burke,E., and

Petrovic,S. (Eds.), Selected Papers from MISTA 2003, Kluwer Academic Publisher.

Kendall, G. and Mohd Hussin, N., 2004b. Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at University Technology MARA, In the Proceedings of the 5th international conference on the Practice and Theory of Automated Timetabling (PATAT), Pittsburgh, USA, Aug. 18-20, 199-217.

Kirkpatrick, S., Gellat, C. D. and Vecchi, M. P., 1983. Optimization by Simulated Annealing, Science. 220, 671-680.

Kitano, H., 1990. Designing Neural Networks Using Genetic Algorithms with Graph Generation System, Complex Systems. 4, 461-476.

Kotzan, J. and Evanson, R., 1969. Responsiveness of Drug Store Sales to Shelf Space Allocations, Journal of Marketing Research. 6, 465-469.

Lasdon, L. S., Waren, A. D., Jain, A. and Ratner, M., 1978. Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming, ACM Transactions on Mathematical Software. 4, 34-50.

Levy, M. and Weitz, B., 1992. Retailing Management, Homewood, IL., ISBN: 0-256-05989-6.

Lim, A., Rodrigues, B. and Zhang, X., 2004. Metaheuristics With Local Search Techniques for Retail Shelf-Space Optimization, Management Science. 50, 117-131

Liu, J., 1999. The Impact of Neighbourhood Size on the Process of Simulated Annealing: Computational Experiments on the Flowshop Scheduling Problem, Computers & Industrial Engineering. 37, 285-288.

Liu, L., 1990. (s,S*)* Continuous Review Models for Inventory with Random Lifetimes, Operations Research Letters. 9, 161-167.

Liu, L. and Lian, Z., 1999. Continuous Review Models for Inventory with Fixed Lifetimes, Operations Research. 47, 150-158.

Lourenco, H. R., Martin, O. C., and Stutzle, T., 2003. Iterated Local Search. In: Glover,F. and Kochenberger,G.A. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 321-354.

Lundy, M. and Mees, A., 1986. Convergence of an Annealing Algorithm, Mathematical Programming. 34, 111-124.

Mandal, B. N. and Phaujdar, S., 1989. An Inventory Model for Deteriorating Items and Stock-dependent Consumption Rate, Journal of the Operational Research Society. 40, 483-488.

Martello, S. and Toth, P., 1975. An Upper Bound for the Zero-one Knapsack Problem and a Branch and Bound Algorithm, European Journal of Operational Research. 1, 169-175.

Martello, S. and Toth, P., 1990a. Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, ISBN: 0-471-92420-2.

Martello, S. and Toth, P., 1990b. Lower Bounds and Reduction Procedures for the Bin Packing Problem, Discrete Applied Mathematics. 28, 59-70.

Mazzola, J. B. and Schantz, R. H., 1995. Single-facility Resource Allocation Under Capacity-based Economics and Diseconomies of Scope, Management Science. 41, 669-689.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E., 1953. Equation of State Calculation by Fast Computing Machines, Journal of Chemical Physics. 21, 1087-1091.

Michalewicz, Z., 1996. Genetic Algorithms + Data Structures = Evolution Programs, 3rd. Ed., Springer.

Michalewicz, Z. and Fogel, D. B., 2000. How To Solve It: Model Heuristics, Springer-Verlag.

Mitchell, M., 1996. An Introduction to Genetic Algorithms, Massachusetts Institute of Technology.

Mladenovic, N. and Hansen, P., 1997. Variable Neighbourhood Search, Computers & Operations Research. 11, 1097-1100.

Mockus, J., 1989. Bayesian Approach to Global Optimization, Kluwer Academic Publishers, Dordrecht-London-Boston.

Mockus, J., 1994. Application of Bayesian Approach to Numerical Methods of Global and Stochastic Optimization, Journal of Global Optimization. 4, 347-366.

Mockus, J., 2000. A Set of Examples of Global and Discrete Optimization: Application of Bayesian Heuristic Approach, Kluwer Academic Publishers, Dordrecht-London-Boston.

Mockus, J., Eddy, W., Mockus, A., Mockus, L., and Reklaitis, G., 1997. Bayesian Heuristic Approach to Discrete and Global Optimization, Kluwer Academic Publishers, Dordrecht-Boston-London.

Moscato, P., 1989. On Evolution, Search, Optimisation, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Technical Report Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, California, USA.

Moscato, P. and Cotta, C., 2003. A Gentle Introduction to Memetic Algorithms. In: Glover,F. and Kochenberger,G.A. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 105-144.

Nahmias, S., 1982. Perishable Inventory Theory: A Review, Operations Research. 30, 680-708.

Nandakumar, P. and Morton, T. E., 1990. Near Myopic Heuristic for the Fixed Life Perishability Problem, Management Science. 39, 241-246.

Nareyek, A., 2003. Choosing Search Heuristics by Non-Stationary Reinforcement Learning, Metaheuristics: Computer Decision-Making (Resende, M.G.C., and de Sousa, J.P.ed.), Kluwer, 523-544.

Nemhauser, G. L. and Wolsey, L. A., 1988. Linear and Combinatorial Optimization, Wiley.

O'Grady, P. J. and Harrison, C., 1985. A General Search Sequencing Rule for Job Shop Sequencing, International Journal of Production Research. 5, 961-973.

Ogbu, F. A. and Smith, D. K., 1990. The Application of the Simulated Annealing Algorithm to the Solution of the n*/m/Cmax* Flowshop Problem, Computers & Operations Research. 17, 243-253.

Osman, I. H. and Kelly, J. P., 1996. Meta-Heuristics: Theory and Applications, Kluwer Academic Publishers.

Osman, I. H. and Laporte, G., 1996. Metaheursitics: A Bibliography, Annals of Operations Research. 63, 513-623.

Oxford Dictionary of Computing, 1997. Oxford University Press, Oxford, 4th ed.

Petrovic, S. and Qu, R., 2002. Case-Based Reasoning as a Heuristic Selector in a Hyper-Heuristic for Course Timetabling Problems, In the Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Applied Technologies (KES'02), Milan, Italy, Sep 16-18. 82, 336-375.

Raafat, F., 1991. Survey of Literature on Continuously Deteriorating Inventory Model, Journal of the Operational Research Society. 42, 27-37.

Rajan, A., Rakesh and Steinberg, R., 1992. Dynamic Pricing and Ordering Decisions by A Monopolist, Management Science. 38, 240-262.

Rao, R. L. and Iyengar, S. S., 1994. Bin-packing by Simulated Annealing, Computers & Mathematics with Applications. 27, 71-82.

Rechenberg, I., 1965. Cybernetic Solution Path of an Experimental Problem, Ministry of Aviation, Royal Aircraft Establishment.

Rechenberg, I., 1973. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog (Stuttgart).

Reeves, C., 1995. Modern Heuristic Techniques For Combinatorial Problems, McGraw-Hill, ISBN: 0-07-709239-2.

Reeves, C. R., 1993. Improving the Efficiency of Tabu Search for Machine Sequencing Problems, Journal of the Operational Research Society. 44, 375-382.

Resende, M. G. C. and Ribeiro, C. C., 2003. Greedy Randomized Adaptive Search Procedures. In: Glover F. and Kochenberger,G.A. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 219-249.

RFID Journal, 2005. http://www.rfidjournal.com.

Rolland, E., Pirkul, H. and Glover, F., 1996. Tabu Search for Graph Partitioning, Annals of Operations Research. 63, 209-232.

Ross, P., 2005. Hyper-Heuristics. In: Burke, E. and Kendall, G. (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, pp. 529-556.

Ross, P., Marin-Blazquez, J.G., Schulenburg, S., and Hart, E., 2003. Learning a Procedure That Can Solve Hard Bin-Packing Problems: A New GA-Based Approach to Hyper-heurstics, Proceeding of the Genetic and Evolutionary Computation Conference, GECCO 2003, Chicargo, Illinois, USA, 1295-1306.

Ross, P., Schulenburg, S., Marin-Blazquez, J.G., and Hart, E., 2002. Hyper Heuristics: Learning to Combine Simple Heuristics in Bin-packing Problems, Proceeding of the Genetic and Evolutionary Computation Conference, GECCO2002, New York, US, 942-948.

Sahni, S., 1976. Approximate Algorithms for the 0-1 Knapsack Problem, Journal of ACM. 22, 115-124.

Sarker, B. R., Mukherjee, S. and Balan, C. V., 1997. An Order-level Lot Size Inventory Model with Inventory-level Dependent Demand and Deterioration, International Journal of Production Economics. 48, 227-236.

Sastry, K., Goldberg, D., and Kendall, G., 2005. Genetic Algorithms. In: Burke, E. and Kendall, G. (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Boston, Dordrecht, London, Kluwer Academic Publishers, pp. 97-125.

Scholl, A., Klein, R. and Jurgens, C., 1997. BISON: A Fast Hybrid Procedure for Exactly Solving the One-dimensional Bin Packing Problem, Computers & Operations Research. 24, 627-645.

Schwefel, H-P., Evolutionsstrategie und numerische Optimierung, PhD Thesis, Technische Universität, Berlin 1975.

Sechen, C., Braun, D. and Sangiovanni-Vincetelli, A., 1988. Thunderbird: A Complete Standard Cell Layout Package, IEEE Journal of Solid-State Circuits. 23, 410-420.

Skorin-Kapov, J. and Vakharia, A., 1993. Scheduling a Flow-Line Manufacturing Cell: A Tabu Search Approach, International Journal of Production Research. 31, 1721-1734.

Smith, J., 2002. Co-evolving Memetic Algorithms: Initial Investigations. In: Parallel Problem Solving from Nature VII, PPSN 2002, Lecture Notes in Computer Science, Springer-Verlag, pp. 537-546.

Soriano, P. and Gendreau, M., 1996. Diversification Strategies in Tabu Search Algorithms for the Maximum Clique Problems, Annals of Operations Research. 63, 189-207.

Soubeiga, E., Development and Application of Hyperheuristics to Personnel Scheduling. PhD Thesis 2003.

Steinhofel, K., Albrecht, A. and Wong, C. K., 2003. An Experimental Analysis of Local Minima to Improve Neighbourhood Search, Computers & Operations Research. 30, 2157-2173.

Sutton, R. S. and Barto, A. G., 1998. Reinforcement Learning: An Introduction, MIT Press, Cambridge - MA.

Taillard, E. D., 1994. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, ORSA Journal on Computing. 23, 108-117.

Taillard, E. D., Gambardella, L. M., Gendreau, M. and Potvin, J., 2001. Adaptive Memory Programming: A Unified View of Metaheuristics, European Journal of Operational Research. 135, 1-16.

Thompson, J. M. and Dowsland, K. A., 1998. A Robust Simulated Annealing Based Examination Timetabling System, Computers & Operations Research. 25, 637-648.

Tian, P., Ma, J. and Zhang, D.-M., 1999. Application of the Simulated Annealing Algorithm to the Combinatorial Optimisation Problem with Permutation Property: An Investigation of Generation Mechanism, European Journal of Operational Research. 118, 81-94.

Toth, P., 1980. Dynamic Programming Algorithms for the Zero-one Knapsack Problem, Computing. 25, 29-45.

Tovey, C. A., 1988. Simulated Simulated Annealing, American Journal of Mathematical and Management Sciences. 8, 389-407.

Urban, T., 1998. An Inventory-Theoretic Approach to Product Assortment and Shelf-Space Allocation, Journal of Retailing. 74, 15-35.

Urban, T., 2002. The interdependence of inventory management and retail shelf management, International Journal of Physical Distribution & Logistics Management. 32, 41-58.

Urban, T. L. and Baker, R. C., 1997. Optimal Ordering and Pricing Policies in a Single-period Environment with Multivariate Demand and Markdowns, European Journal of Operational Research. 103, 573-583.

Valerio de Carvalho, J. M., 1999. Exact Solution of Bin-packing Problems Using Column Generation and Branch-and-bound, Annals of Operations Research. 86, 629-659.

Verbeke, W., Farris, P. and Thurik, R., 1998. Consumer Response to the Preferred Brand Out-of-Stock Situation, European Journal of Marketing. 32, 1008-1028.

Voss, S., Martello, S., and Osman, I. H., 1999. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers.

Voudouris, C. and Tsang, E.P.K., 1997. Guided Local Search. Technical Report CSM-247, Department of Computer Science, University of Essex.

Voudouris, C. and Tsang, E. P. K., 2003. Guided Local Search. In: Glover F. and Kochenberger,G. (Eds.), Handbook of Metaheuristics, Kluwer, pp. 185-218.

Wang, C.J. and Tsang, E.P.K., 1991. Solving Constraint Satisfaction Problems Using Neural-networks, Proceedings of the IEE Second International Conference on Artificial Neural Networks, 295-299.

Wang, C. J. and Tsang, E. P. K., 1994. A Cascadable VLSI Design for GENET. In: Delgado-Frias,J.G. and Moore,W.R. (Eds.), VLSI for Neural Networks and Artificial Intelligence, New York, Plenum Press, pp. 187-196.

Widmer, M. and Hertz, A., 1989. A New Heuristic Method for the Flow Shop Sequencing Problem, European Journal of Operational Research. 41, 186-193.

Wolpert, D. and MacReady, W. G., 1997. No Free Lunch Theorems for Optimization, IEEE Transactions on Evolutionary Computation. 1, 67-82.

Xu, H. and Wang, H.-P., 1990. An Economic Ordering Policy Model for Deteriorating Items with Time Proportional Demand, European Journal of Operational Research. 46, 21-27.

Yang, M.-H., 2001. An Efficient Algorithm to Allocate Shelf Space, European Journal of Operational Research. 131, 107-118.

Yang, M.-H. and Chen, W.-C., 1999. A Study on Shelf Space Allocation and Management, International Journal of Production Economics. 60, 309-317.

Yao, X., 1999. Evolutionary Computation: Theory and Applications, World Scientific.

Yeo, A., 1997. Large Exponential Neighbourhoods for the Traveling Salesman Problem, Technical Report: PP-1997-47, University of Southern Demark.

Zhou, Y.-W., Lau, H.-S. and Yang, S.-L., 2003. A New Variable Production Scheduling Strategy for Deteriorating Items with Time-varying Demand and Partial Lost Sale, Computers & Operations Research. 30, 1753-1776.

Zufryden, F., 1986. A Dynamic Programming Approach for Product Selection and Supermarket Shelf-Space Allocation, Journal of Operations Research Society. 37, 413-422.