Chapter 4

AN INVESTIGATION OF AUTOMATED PLANOGRAMS USING A SIMULATED ANNEALING BASED HYPER-HEURISTIC

Ruibin Bai and Graham Kendall
Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
School of Computer Science & IT, University of Nottingham
Nottingham NG8 1BB, UK Email: rzb/gxk@cs.nott.ac.uk

Abstract: This paper formulates the shelf space allocation problem as a non-linear function of the product net profit and store-inventory. We show that this model is an extension of multi-knapsack problem, which is itself an NP-hard problem. A two-stage relaxation is carried out to get an upper bound of the model. A simulated annealing based hyper-heuristic algorithm is proposed to solve several problem instances with different problem sizes and space ratios. The results show that the simulated annealing hyper-heuristic significantly outperforms two conventional simulated annealing algorithms and other hyper-heuristics for all problem instances. The experimental results show that our approach is a robust and efficient approach for the shelf space allocation problem.

Key words: hyper-heuristics, simulated annealing, shelf space allocation, planograms

1. INTRODUCTION

The retailing sector in the UK is an extremely competitive arena. We only need to consider some high profile companies to show that this is the case. A particular example is provided by the recent decline of Marks and Spencer, who were the leading high street retailer (and in recent years they are starting to show an improvement in their profitability). A further example is given by C&A's decision to close all of its high street 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 17% of market share, while Sainsbury's had slipped from 17.1% to 16.2%. Tesco retains the top spot with 27% of the overall market share. Finally, there was a battle over Safeways, which was recently up for sale.

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.

Figure 4-1. An example of a simple planogram

Several factors are used to influence consumers' purchases, including product assortment (which merchandise to sell), store layout and space planning, merchandise pricing, services offered, advertising and other promotional programs [21]. Store layout and space planning focuses on the improvement of the visual effect of the shopping environment and space productivity. Planograms (see figure 4-1 for an example) are used to show exactly where and how many facings of each item should physically be placed onto the store shelves. Due to the limited shelf space, planograms are one of the most important aspects that are used to improve financial performance [31]. Electronic planograms can be also used for inventory control and vendor relation improvement [21]. 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 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) but it involves significant human interaction and does not provide any guidance or suggestions in deciding a good quality layout) and the shelf space allocation is mainly based on historical market share. However, this approach may lose substantial sales [2] as the display space may have different sales influence with respect to different items [11, 12, 14, 20]. Using the same display space, different items may obtain different sales and hence affect the profits of the organisation.

A recent publication [31] conducted a survey of the area. This work demonstrated 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.

At present, commercial systems use simple heuristic rules to allow retailers to plan their shelf space allocation [32]. Some research [2, 26] has proposed models which integrate product assortment, inventory control management and shelf space allocation. However, these models are too complicated to put into practice. Yang [30] used a model based on a knapsack problem and presented a heuristic to solve some numerical instances of shelf space allocation problems. However, the linear objective function assumption does not fit well with the real world retailing environment.

Planograms are a subset of the wider domain of space planning which includes more well known research areas such as bin packing and knapsack problems [31]. Some of techniques that have already been successfully applied to problems within this wider domain may also be promising to shelf space allocation problems.

2. RELATED WORK

2.1 Experiments and studies

Due to the scarcity of space within stores, some 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 depended on the type of items [11, 12, 14, 16, 20]. Earlier, in 1969, Kotzan and Evanson [20] 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 [11] found that this relationship was very weak and depended on the category of products. However, his experimental results may be affected by limited experiment brand samples (only two brands are measured for each of the two categories). Curhan [12] defined the space elasticity as "*the ratio of relative change in unit sales to relative change in shelf space*" and argued that there existed a small, positive space elasticity for each item but the value depended on the products, stores and in-store layout [13]. Drèze et al. [16] carried out a series of experiments to evaluate the effectiveness of shelf space management and cross-category merchandise reorganisation. The shelf space manipulation included changing product facings, deletion of slow moving items, changes of shelf height, etc. Cross-category merchandise reorganisation included manipulation to enhance the complementary shopping within merchandise assortment and easier shopping. 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. On the contrary, recent research [14] showed that direct space elasticities were significantly non-zero and varied considerably through different categories. Costume jewellery, fruit and vegetables, underwear, shoes were among the highest space elasticities while textile, 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 has no effect on its sales [2]. 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 [28]. On the other hand, the purchase of some merchandise could increase the possibility of buying other merchandise with the 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 [7]. Borin et al. [2] and Urban [26] also used cross elasticities in their models but with different definitions. In Borin et al.'s model, the cross elasticities were in the range [–1, 0]. Urban extended the range of cross elasticities to [-1, 1]. It was positive if two items were complementary and negative if they could be substituted for each other. 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 disregarded it in their models [14, 27].

Display location is another factor that has been studied. Campo et al. [5] 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 the store.

2.2 Shelf space allocation models and searching methods

Several space allocation models have been proposed by market researchers. Most 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 [1]:

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

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 shape 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 parameters can then be estimated by a simple linear regression) and its *richness*. Corstjens and Doyle [7] formulated their model as a multiplicative form and incorporated both the space elasticities and cross elasticities. The inventory and handling cost effects were also considered. The model allowed different product profit margins corresponding to different locations and hence captured the location impact on the sales. However, due to the characteristic of the polynomial function, this model tends to scatter the facings of each item into different locations in order to obtain a larger objective function value. Based on this model, some non-space factors were also taken into account by Zufryden's model [32], such as price, advertisement, promotion, store characteristics, etc. A dynamic programming approach was proposed to solve this model. However, this approach ignored the integer nature of the number of facings of the items and hence only produced suboptimal solutions.

Some integrated models were also proposed based on the correlation of retailing decision process [2, 26]. Borin et al. [2] developed a different model which tried to maximise the category return on inventory. This model was supposed to help the retailer to decide which products to stock (products 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 of 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. A heuristic procedure, based on simulated annealing, was employed to optimise the model. The neighbourhood was defined by exchanging one facing of two random items. The result 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 to foresee the demand quantity of that item. However, the effect of part-stocked items (some facings are missing) was not explicitly reflected. Urban [26] replaced the number of facings with average inventory in his model which integrated existing inventory-control model, product assortment model and 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 scaling down the number of facings of each item and their order quantities. However, the GA operations (crossover and mutation) were applied to only produce a good product assortment based on the given objective function. No procedure was carried out to evolve a good space allocation decision. The drawback mainly results from the fact that the model includes many parameters and is difficult to be optimised by current AI search techniques. In fact, Yang [30] 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 Zufryden's work [32], by assuming that total net profit was linear with the number of facings of an item. However, this is unrealistic for the real world retail environment. 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 was easily trapped in local optima.

2.3 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 [18, 24]. Most of these applications require a thorough study of the problem and a fixed problem definition. Many of the publications also reported that these algorithms perform very well in solving their specific problems. However, once the problem changes (even slightly), the performance of the already developed meta-heuristic may decrease dramatically for the new problem. Significant parameter tuning may need to be carried out for the purpose of adapting the algorithms to the new problem or the new problem instance. It should also be recognised that real-world problems are subject to changes due to them reflecting changes in the business requirements either by management decisions or other factors, such as trading conditions, research and development, employing new technology etc.

The "No Free Lunch Theorem" [29] showed that there is no one algorithm that could beat all other algorithms in all classes of problems. If an algorithm outperforms other algorithms on a specific class of problems, there must exist another class of problems on which this algorithm is worse than the others. Hence, a good way to raise the generality of meta-heuristics is to apply different (meta-)heuristics at different times of the search. In this context, a generalised approach (termed hyper-heuristics) is proposed [3] which "*broadly describes the process of using (meta-)heuristics to choose (meta-)heuristics to solve the problem in hand*". This approach differs from the more usual meta-heuristic approach, which operates over the solution space directly. A hyper-heuristic approach operates over the solution space indirectly by searching the heuristic space. In this approach, there is a set of "low-level heuristics" that are designed for the problem to be solved. Another heuristic, a high-level heuristic, operates over the low-level heuristics. From an initial solution, the high-level heuristic leads the search to a good quality solution by making calls to the low-level heuristics. The benefit of this approach is two-fold. Firstly, once the high-level heuristic algorithm has been developed, a new problem can be solved by replacing the set of low-level heuristics and the objective function, which is used to evaluate the quality of the solutions. Secondly, the high-level heuristic can adapt itself in order to tune to the new problem, or even a new problem instance. The application of hyper-heuristic approaches can be traced back to the 1960's although the term "hyper-heuristic" was not used. Fisher and Thompson [17] used an unbiased random process to combine two

rules/heuristics to solve a job-shop scheduling problem and the results showed that even this simple combination of rules produced much better results than any of them used separately. Recently, Hart et al. [19] solved a real world chicken factory scheduling problem using a GA based hyper-heuristic. The problem involved scheduling the collection and delivery of chicken from farms to its processing factories. The problem was decomposed into two stages and two separate GAs were used to tackle the problem. 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 suitable sized tasks and another set of heuristics to assign these tasks to the different "catching squads". The GA was used to evolve a strategy of building a good solution instead of finding the solution directly. The experimental results showed this approach is fast and robust and easy to implement. Another GA based hyper-heuristic framework was also proposed by Cowling et al. [9] in solving 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. Recently, Ross et al. [25] proposed a different type of hyper-heuristic based on a genetic algorithm. The problem is one dimensional bin packing. Instead of working on feasible solutions as the hyper-heuristics mentioned above do, the proposed hyper-heuristic in this paper operates on a partial solution and gradually constructs the solution using different rules (heuristics) until a feasible solution is obtained. The heuristic selection is based on the state of current partial solution. Each state associates a rule or heuristic whose relationship with solution states is evolved by a genetic algorithm. The chromosomes of their GA are defined as a set of blocks and each block contains a set of parameters which is used to define a solution state and its corresponding heuristics. The algorithm is firstly trained on parts of benchmark problems. After the training, the fittest chromosome is then applied to every benchmark problem, 80% of which are solved to optimality. Yet another kind of hyper-heuristic used the ideas of reinforcement learning to guide the choice of the heuristics during the search [8, 23]. In [8], 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 performed better than applying the heuristics randomly. Nareyek [23] used a non-stationary

reinforcement learning procedure to choose heuristics in solving the Orc Quest problem and the Logistics Domain problem. The author discussed the advantages of the hyper-heuristic approach, especially in solving complex real-world problems in which the computational cost is expensive. Burke et al. [4] applied a tabu search based hyper-heuristic to a nurse rostering problem and a university course timetabling problem, in which the set of heuristics were ranked according to their performances in the search history. A tabu list is also incorporated to prevent the selection of some heuristics at certain points in the search.

The remainder of this paper is organised as follows: in the section 3, we give a description of the problem and formulate it as a non-linear combinatorial maximisation model. Section 4 will focus on the implementation of our simulated annealing hyper-heuristic. Section 5 gives the experimental results, together with some analysis. Section 6 concludes the paper.

3. MODEL FORMULATION

3.1 Model formulation

The problem we are solving is the assignment of appropriate shelf space to every stock-keeping unit (SKU) in a given product category, without violating the given constraints, whilst maximising the overall profit. 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 length (respectively, profit, space elasticity, lower bounds, upper bounds) of item *i*. The length of shelf *j* is denoted by T_j. We assume that: 1) Retailers prevent out-of-stock occurrences. 2) The total profit of item *i* is proportional to its unit profit p_i. 3) We ignore the physical constraints in the other two dimensions (height and depth).

We employ Urban's [26] demand function and disregard the cross elasticities not only because they are quite small compared with space elasticities but also because it is quite difficult to obtain a reliable estimation of them. Based on the assumptions we discussed earlier, we have the following space allocation model

$$\text{Maximise} \quad P = \sum_{i=1}^{n} (p_i \alpha_i x_i^{\beta_i}) \tag{2}$$

subject to:

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

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

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

where m is the number of shelves and n is the number of items. 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 $\alpha_i > 0$. Constraint (3) ensures that the length of a shelf is greater than the total length of the facings assigned to this shelf. Constraint (4) ensures that the lower and upper bounds of the number of facings for each item are satisfied. Constraint (5) ensures that the number of facings for each item is an integer. The objective is to maximise the overall profit without violating the given constraints. The model is a non-linear, multi-constraints optimisation problem. If $\beta_i \to 1$, the model degenerates into a multi-knapsack problem.

3.2 Upper bound of the model

As shelf space allocation cannot be solved to optimality in polynomial time [2], 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 [30] 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 requires $U^{m \times n}$ iterations to find the optimal solution using an exhaustive search. Even for a small problem instance: n=6, m=3, U=6, this could take around 40 years, an extremely unrealistic computing time for a practical application. Another common method is to relax the problem to a

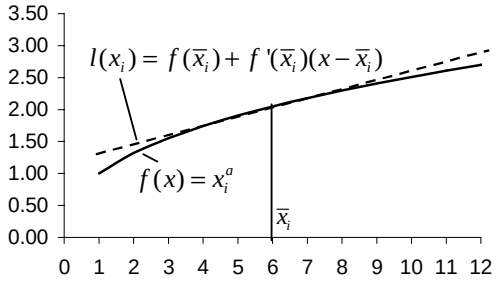

Figure 4-2. Approximate objective with a linear function

simpler one and obtain the upper bound of the objective. In this paper, we use a two-stage relaxation to get an upper bound of the model:

Stage 1: we first relax our non-linear model to be a linear model. This is accomplished by applying a linear Taylor expansion at the point $\overline{x}_i$ ($L_i \le \overline{x}_i \le U_i$) (as illustrated in figure 4-2). The model hence becomes an integer programming (IP) problem:

$$\text{maximise} \quad P_{IP} = \sum_{i=1}^{n} p_i \cdot \alpha_i \cdot (\beta_i \cdot \overline{x}_i^{(\beta_i - 1)} \cdot (x_i - \overline{x}_i) + \cdot \overline{x}_i^{\beta_i}) \tag{6}$$

or

$$\text{maximise} \quad P_{IP} = \sum_{i=1}^{n} (A_i x_i + B_i) \tag{7}$$

subject to the constraints (3), (4) and (5), where $A_i = p_i \cdot \alpha_i \cdot \beta_i \cdot \overline{x}_i^{(\beta_i - 1)}$ and $B_i = p_i \cdot \alpha_i \cdot \overline{x}_i^{\beta_i} - p_i \cdot \alpha_i \cdot \beta_i \cdot \overline{x}_i^{(\beta_i - 1)} \cdot \overline{x}_i$. Suppose $X^* = (x_1^*, x_2^*, ..., x_n^*)$ is the optimal solution for the original model (2) and P^* is its corresponding optimal objective value. P_{IP}^* is the optimal objective value for the IP model (7). From figure 4-2, we have:

$$\begin{aligned} P^* &= \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i} = \sum_{i=1}^{n} (A_i x_i^* + B_i) - [\sum_{i=1}^{n} (A_i x_i^* + B_i) - \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i}] \\ &\le P_{IP}^* - [\sum_{i=1}^{n} (A_i x_i^* + B_i) - \sum_{i=1}^{n} p_i \alpha_i (x_i^*)^{\beta_i}] \\ &= P_{IP}^* - \sum_{i=1}^{n} p_i \alpha_i [\overline{x}_i^{\beta_i} + (x_i^* - \overline{x}_i) \beta_i \overline{x}_i^{(\beta_i - 1)} - (x_i^*)^{\beta_i}] \quad \le P_{IP}^* \end{aligned} \tag{8}$$

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

$$G_1 = \sum_{i=1}^{n} p_i \alpha_i [\overline{x}_i^{\beta_i} + (x_i^* - \overline{x}_i) \beta_i \overline{x}_i^{(\beta_i - 1)} - (x_i^*)^{\beta_i}] \tag{9}$$

From equation (9), we can see 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).

Stage 2: based on the approximation from stage 1, we ignore the integer constraint (5) in the IP model and the model becomes a linear programming (LP) model. We use "lp_solve" (a free LP software package) to obtain the optimal objective (denoted by P_{LP}^*) of this LP model. We take this value as the relaxed upper bound of our shelf space allocation model P^{ub} , i.e. $P^{ub} = P_{LP}^*$.

4. SIMULATED ANNEALING HYPER-HEURISTIC

4.1 Simulated annealing

Simulated annealing is a local search method inspired by the process to simulate the physical cooling process. From an initial solution, SA repeatedly generates a neighbour of the current solution and transfers to it according to some strategy with the aim of improving the objective function value (we are assuming we are trying to maximise the objective function). During this process, SA has the possibility to visit worse neighbours in order to escape from a local optimum. Specifically, a parameter, called temperature *t*, is used to control the possibility of moving to a worse neighbour solution. The algorithm, starting from a high temperature, repeatedly decreases the temperature in a strategic manner (usually called a cooling schedule) until the temperature is low enough or some other stopping criteria is satisfied. In each iteration, the algorithm accepts all "good" moves and some of the "bad" moves according to the Metropolis probability, defined by $\exp(-\delta/t)$ where δ is the decrease in the objective function value. Simulated annealing has been shown to be a powerful tool in solving a variety of combinatorial optimisation problems [15]. However, the drawback of SA is that the algorithms performance is sensitive to the parameters and problem instance. Many experiments need to be carried out in order to tune the parameters to the problem.

4.2 Simulated annealing hyper-heuristic

Hyper-heuristics were proposed to be a more general approach for most combinatorial optimisation problems and they have the ability to adapt themselves to different problems or problem instances. Here, we propose another type of hyper-heuristic: a simulated annealing based hyper-heuristic. The basic idea behind this approach is that we use simulated annealing to guide the selection and acceptance of the low-level heuristics (or neighbourhood functions, see section 4.3), instead of controlling the moves between neighbours. From an initial solution, simulated annealing leads the search in a promising direction by making calls to the appropriate low-level heuristics. Specifically, for a maximisation problem**,** the algorithm works as follows:

Define an objective function *f* and a set of heuristics *H*;
Define a cooling schedule: starting temperature $t_s > 0$, a temperature reduction function φ and a number of iterations for each temperature *nrep* ;

Select an initial solution s_0 ;
Repeat
  Randomly select a heuristic $h \in H$;
  iteration_count = 0;
  Repeat
    iteration_count++;
    Applying *h* to s_0 , get a new solution s_1 ;
    $\delta = f(s_1) - f(s_0)$
    if ($\delta \geq 0$) then $s_0 = s_1$;
    else
      Generate a random *x* uniformly in the range (0,1);
      if $x < \exp(\delta / t)$ then $s_0 = s_1$;
  Until *iteration_count* = *nrep*;
  set $t = \varphi(t)$;
Until the stopping criteria = *true*.

It should be noted that simulated annealing would normally only have access to one heuristic or neighbourhood function (e.g. 2-opt in a travelling salesman problem), but here we give it access to a set of heuristics which allows it to adapt itself to a given problem (instance) by utilising different heuristics.

In the above algorithm, we have stated that we need to define a cooling schedule. In fact, we would like to do this automatically so that we have an adaptively parameterised algorithm. Compared with a geometric cooling, Lundy and Mees's cooling schedule [22] has one less parameter because at each temperature only one iteration was performed. The temperature is reduced according to $t \to t/(1+\beta t)$. Hence, we use this cooling function in this paper. Suppose we allow $T_{allowed}$ seconds for the search and the average time spent for one iteration was $T_{average}$, we have the total number of the iterations $K = T_{allowed} / T_{average}$. After the mathematical derivation, we have

$$\beta = (t_s - t_f) / K \cdot t_s \cdot t_f \;=\; (t_s - t_f) \cdot T_{average} / T_{allowed} \cdot t_s \cdot t_f \qquad (9)$$

where t_s (respectively t_f) is the starting temperature (respectively stopping temperature). In this paper, the algorithms stops when the temperature (t_f) decreases to 0.1. Two different methods were used to determine the starting temperature (corresponding to two kinds of simulated annealing hyper-heuristics, denoted as SAHH and SAHH_adpt respectively) in order to investigate the sensitivity of parameters in our simulated annealing based hyper-heuristic. In SAHH, after preliminary experiments, we let $t_s = 0.3 f(s_0)$ where $f(s_0)$ is the objective function value of the initial solution. To automate the decision of t_s , in SAHH_adpt , we use a similar

method described in [6]. $K/100$ random solutions were sampled from the initial solution to approximately determine the maximum objective difference $\delta_{\max}$. The starting temperature was then set to a value such that 85% of "bad moves would be accepted. According to the Metropolis probability function, we have $t_s = -\delta_{\max} / \ln(0.85)$.

4.3 Low-level heuristics

Before we describe the low-level heuristics which are used in the hyper-heuristics, we first 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.

Twelve low-level heuristics are used. They are categorised into four types: add product(s), delete product(s), swap and interchange:

- ***Add_random*:** this heuristic adds one facing of a random item to the first shelf of S_0.
- ***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 as possible facings of an item 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 evaluation function.
- ***Delete_random*:** this heuristic deletes one facing of a random item from a random shelf.
- ***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.

- ***Swap_best*:** this heuristic repeatedly selects a shelf from S_0, deletes one facing of the item with the lowest contribution value, 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 and interchanges one facing or multiple facings of 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.

Note that each of above low-level heuristic is enforced to generate a feasible solution from the incumbent solution. If a low-level heuristic cannot produce a new feasible solution, the incumbent solution is returned.

5. 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. The length of the products conformed to a uniform distribution between 25 and 60. The net profit of the products were created randomly by a normal distribution in the same way as described in [30]. 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 [30] experimental results which show that the problem size is a potential factor affecting algorithm performance, in this paper, five problem instances with different problem sizes were generated to test this relationship. We also take into account the influence of space availability in the performance of the algorithms. Because each item has a lower bound and an upper bound of facings, the available shelf space of a problem must be greater than a minimal space value to satisfy the lower bound of facings and meanwhile it should not exceed a maximal space value in case that all items' facings reach the upper bounds 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 with different *r_min* and *r_max* values were also generated to test the corresponding algorithms performance.

Two simple hyper-heuristics, RHOI (Random Heuristics Only Improving) and RHAM (Random Heuristics All Moves), were also applied to the problems for the purpose of comparison. RHOI repeatedly selects a random low-level heuristic and applies it to the current solution until some stopping criteria is met, during which only those heuristics that can improve the objective function value are accepted. RHAM works in the similar way but all moves are accepted. We also experimented with a "Choice Function" based hyper-heuristic which was proposed in [10]. In this approach, the selection of the low-level heuristics is guided by a "Choice Function", which considers 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). Overall, the function is defined as

$$CF(h_j) = \pi_1 f_1(h_j) + \pi_2 f_2(h_k, h_j) + \pi_3 f_3(h_j) \quad (10)$$

Both f_1 and f_2 are used as a method to intensify the search and f_3 is used as a diversification strategy. π_1, π_2 and π_3 are scaling parameters to weight the different terms. Values of these parameters are changed adaptively according to the magnitude of recent improvement in the objective function. A more detailed description is provided in [10]. 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 is used in SAHH but utilising different neighbourhood structures. In SA_swap, the neighbourhood structure was defined by randomly swapping one facing of two different items on a random shelf. However, 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.

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 a solution produced by a greedy heuristic (the greedy heuristic repeatedly adds the item with the largest possible *item_contribution* value) and allowed 600 seconds computation time for a fair comparison. The algorithms' performance was evaluated by the ratio of best objective value (P^h) obtained by the different algorithms to the relaxed upper bound (P^{ub}). All results were averaged over 5 runs.

In the first round experiments, seven problem instances with different space ratios were solved by the algorithms. Figure 4-3 shows the results. We can

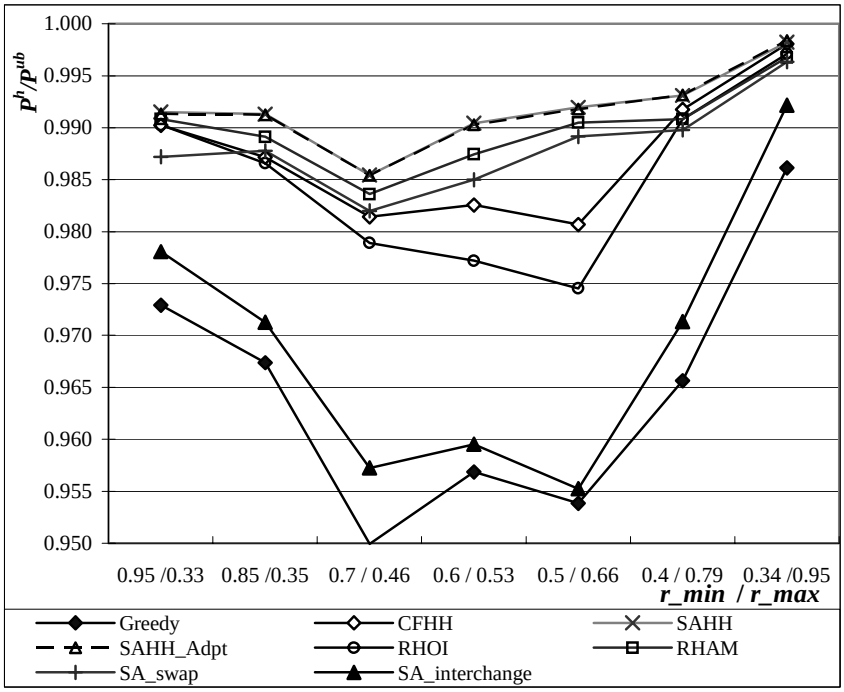

Figure 4-3. Algorithms performance for different space ratios

see that all types of hyper-heuristics 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 is greatly dependent on the neighbourhood structure. We can also see that our simulated annealing based hyper-heuristics outperformed all other algorithms in all cases with surprising high solution quality. Both types of simulated annealing hyper-heuristics obtained over 98.5% of the upper bound (calculated by the two-stage relaxation). The performance of SA based hyper-heuristic slightly decreased when *r_min* and *r_max* reached the middle of their ranges. This is probably because that, when the *r_min* is large while *r_max* is small, the shelf space is very scare, 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, the space is so ample 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.

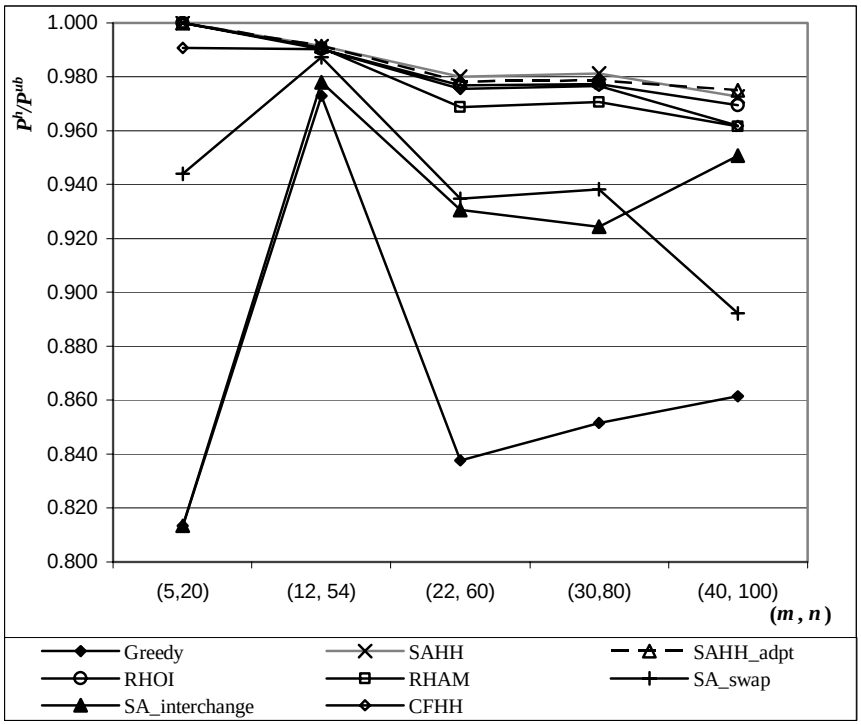

Figure 4-4. Algorithms performance for different problem sizes

To test the influence of the problem size, we also generated five problem instances with different problem sizes in terms of the number of the shelves and the number of the items. In case of the influence of the space availability, we let all the problems have almost the same space ratio (*r_min* = 0.95, *r_max* = 0.24). Figure 4-4 and table 4-1 show the corresponding experimental results and comparison. We can see that SAHH and SAHH_adpt outperformed all other algorithms, including the two simple simulated annealing algorithms. The results also show that our simulated annealing hyper-heuristic performed slightly worse when the problem size increased but still obtained more than 97% of the relaxed upper bound for a very large problem (*m*=40, *n*=100). From both figure 4-3 and table 4-1, we can see that SAHH and SAHH_adpt had almost the same performance. This shows that our simulated annealing hyper-heuristic is not sensitive to the change of the starting temperature and hence is a robust framework. In contrast, in figure 4-4, we can see that both SA_swap and SA_interchange are very sensitive to the change of the problem instances. For small problem sizes, SA_swap performed better than SA_interchange. However, for the large problem sizes, SA-interchange performed better than SA_swap. This demonstrates that, for conventional SA, a good neighbourhood structure for

a given problem instance does not guarantee good performance for another problem instance. However, SA based hyper-heuristics can synergise several neighbourhood functions (or low-level heuristics) according to the characteristics of different problem instances.

Table 4-1. Algorithms performance for different problem sizes

(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	1077.38
Greedy	P^h P^h/P^{ub}	151.73 0.813	410.81 0.973	511.51 0.838	753.35 0.852	928.07 0.861
CFHH	P^h P^h/P^{ub}	184.79 0.991	418.13 0.990	595.74 0.976	863.93 0.977	1036.23 0.962
SAHH	P^h P^h/P^{ub}	**186.53** **1.000**	**418.68** **0.992**	**598.50** **0.980**	**868.15** **0.981**	1048.04 0.973
SAHH_adpt	P^h P^h/P^{ub}	**186.53** **1.000**	418.60 0.991	597.41 0.978	865.86 0.979	**1050.60** **0.975**
RHOI	P^h P^h/P^{ub}	**186.53** **1.000**	418.14 0.990	596.60 0.977	864.59 0.977	1044.60 0.970
RHAM	P^h P^h/P^{ub}	**186.53** **1.000**	418.38 0.991	491.65 0.969	858.56 0.971	1036.01 0.962
SA_swap	P^h P^h/P^{ub}	176.09 0.944	416.85 0.987	570.78 0.935	830.09 0.938	961.19 0.892
SA_ interchange	P^h P^h/P^{ub}	151.73 0.813	412.99 0.978	568.37 0.931	817.71 0.924	1024.42 0.951

6. CONCLUSION

In this paper, we have used a practical shelf space allocation model to generate automatic planograms. Several hyper-heuristic approaches were applied to solve this problem. As an extension of the multi-knapsack problem, the planogram problem is difficult to solve. We provided a set of simple low-level heuristics which have been shown to be very successful in bin packing and knapsack problems. A simulated annealing based hyper-heuristic framework was proposed to solve the problem. In this approach, simulated annealing was used to guide the selection and acceptance of the appropriate heuristics at different search stages instead of controlling moves among neighbours. To give a better evaluation of the solution quality

obtained by different algorithms, the upper bound of the objective function was also derived by a two-stage relaxation. The experimental results show that the simulated annealing based hyper-heuristics used in this paper produced high quality solutions in different problem situations and outperformed three other hyper-heuristics and two versions of the conventional simulated annealing algorithms. The simulated annealing hyper-heuristic does not seem parameter sensitive, which has always been a problem for the conventional simulated annealing algorithms.

Simulated annealing hyper-heuristic is a very promising technique for combinatorial optimisation problems. In the future, we will also investigate different problems in an attempt to demonstrate the generalisation of this approach.

REFERENCES

1 Baker, R. C. and Urban, T. L., *A Deterministic Inventory System with an Inventory-Level-Dependent Demand Rate*. Journal of the Operational Research Society, 39(9): 823-831, 1988.

2 Borin, N., Farris, P. W. and Freeland, J. R., *A Model for Determining Retail Product Category Assortment and Shelf Space Allocation*. Decision Sciences, 25(3): 359-384, 1994.

3 Burke, E., Hart, E., Kendall, G, Newall, J., Ross, P, and Schulenburg, S., "*Hyper-Heuristics: An Emerging Direction in Modern Search Technology*" in Handbook of Meta-Heuristics (Glover F. and Kochenberger, G. eds.), Kluwer, ISBN: 1-4020-7263-5, 457-474, 2003.

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

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

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

7 Corstjens, M. and Doyle, P., *A Model for Optimaizing Retail Space Allocations*. Management Science, 27(7): 822-833, 1981.

8 Cowling, P., Kendall, G. and Soubeiga, E., *Adaptively Parameterised Hyperheuristics for Sales Summit Scheduling*. 4th Metahuristics International Conference [MIC 2001], 2001.

9 Cowling, P., Kendall, G. and Han, L., *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, 1185-1190, 2002.

10 Cowling, P., Kendall, G. and Soubeiga, E., *A Parameter-free Hyperheuristic for Scheduling a Sales Summit*. Proceedings of the 4th Metaheuristic International Conference[MIC 2001], 127-131, 2001.

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

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

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

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

15 Dowsland, Kathryn A., "*Simulated Annealing*" in Modern Heuristic Techniques for Combinatorial Problems (Reeves, C. R. ed.), McGraw-Hill, ISBN: 0-07-709239-2, 21-69, 1995.

16 Drèze, X., Hoch, S. J. and Purk, M. E., *Shelf Management and Space Elasticity*. Journal of Retailing, 70(4): 301-326, 1994.

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

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

19 Hart, E., Ross, P. and Nelson, J. A., *Solving a Real-World Problem Using An Evolving Heuristically Driven Schedule Builder*. Evolutionary Computing, 6(1): 61-80, 1998.

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

21 Levy, Michael and Weitz, Barton, *Retailing Management*, Homewood, IL., ISBN: 0-256-05989-6, 1992.

22 Lundy, M. and Mees, A., *Convergence of An Annealing Algorithm.* Mathematical Programming, 34: 111-124, 1986.

23 Nareyek, A., *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, 2003.

24 Reeves, Colin R., *Modern Heuristic Techniques For Combinatorial Problems*, McGraw-Hill, ISBN: 0-07-709239-2, 1995.

25 Ross, P., Marin-Blazquez, J. G., Schulenburg, S. and Hart, E., *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, GECCO2003, Berlin, Germany: 1295-1306, 2003.

26 Urban, T., *An Inventory-Theoretic Approach to Product Assortment and Shelf-Space Allocation*. Journal of Retailing, 74(1): 15-35, 1998.

27 Urban, T., *The interdependence of inventory management and retail shelf management.* International Journal of Physical Distribution & Logistics Management, 32(1): 41-58, 2002.

28 Verbeke, W., Farris, P. and Thurik, R., *Consumer Response to the Preferred Brand Out-of-Stock Situation.* European Journal of Marketing, 32(11/12): 1008-1028, 1998.

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

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

31 Yang, M.-H. and Chen, W.-C., *A Study on shelf Space Allocation and Management.* International Journal of Production Economics, 60(61): 309-317, 1999.

32 Zufryden, F., *A Dynamic Programming Approach for Product Selection and Supermarket Shelf-Space Allocation.* Journal of Operations Research Society, 37(4): 413-422, 1986.