An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics

Ruibin Bai Graham Kendall

Automated Scheduling, optimisation and Planning (ASAP) Research Group
School of Computer Science & IT, University of Nottingham
Nottingham NG8 1BB, UK
{rzb,gxk}@cs.nott.ac.uk

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 a recent decline of Marks and Spencer, who were the leading high street retailer (in recent years they are stating 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 Sainsbury 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). The recent merger of Asda and Wal-Mart is expected to bring increased pressure on other retailers due to the buying power of Wal-Mart. Finally, there is a battle over Safeways, which is currently 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.

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 [8]. Store layout and space planning focuses on the improvement of the visual effect of the shopping environment and space productivity. Planograms 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 [14]. However, generating planograms is a very 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 problem which is very difficult to solve. The difficulty is further increased when we tackle other merchandise, such as fresh food, clothing and frozen food, due to their special display requirements and not using 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 the historical market shares. However, this approach may lose substantial sales [1] as the display space may have different sales influence with respect to different items [5, 6]. Using the same display space, different items may obtain different sales and hence produce different profits. Some models have been suggested to tackle this problem. Borin et al. [1] used a simulated annealing algorithm to optimise their model in which the demand rate is formulated as an exponential function of the space allocated to an item. Urban developed his model such that the demand function could reflect the occurrence of out-of-stock and a genetic algorithm is proposed to solve the problem [11]. However, a large number of parameters (especially $n \times (n-1)$ cross elasticities for n items) in these two models make them difficult to be put into an application because of the difficulty in obtaining a reliable estimation of their values due to the complicated merchandise relationships. Yang [13] proposed a simpler model, however, the linear objective function does not fit the real world retailing environment very well. All of the three publications reported very good results from their algorithms. However, the performances of these algorithms are largely dependent on the problems. For a different problem, or even a different problem instance, the algorithms may not perform well as the algorithms require substantial knowledge of the problem. 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*". In this approach, there is a set of low-level heuristics that are simple and tailored 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 itself to the new problem, or even a new problem instance. The hyper-heuristics successfully solved a sales summit scheduling problem [4], in which a *choice function* (see section 3.3.2) is used to decide which low-level heuristic to call. A nurse rostering problem and a university course timetabling problem were also solved by a tabu search based hyper-heuristic, in which heuristics compete against one another and a tabu list is maintained to prevent the selection of some heuristics at certain times during the search [2]. Han et al. [7] solved a trainer scheduling problem, in which a genetic algorithm is used to evolve a good sequence of low-level heuristics.

This research has two objectives: Firstly, we present a practical shelf space allocation model that can fit real world problems. Secondly, a new kind of hyper-heuristic approach is proposed. In this approach, simulated annealing is used to provide the suggestion of accepting low-level heuristics instead of finding the solution itself. The results show that our simulated annealing based hyper-heuristic is robust and gives the best solution among other hyper-heuristics, in most cases.

2 Model formulation

The problem we are solving is the assignment of appropriate shelf space to every stock-keeping unit 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 $(l_i, p_i, \beta_i, L_i, U_i)$ 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 [11] demand function and disregard the cross elasticities not only because they are quite small compared with space elasticities but also 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

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

subject to:

$$\sum_{i=1}^{n} l_i x_{ij} \le T_j \quad j = 1, \dots, m \tag{2}$$

$$L_i \le \sum_{j=1}^{m} x_{ij} \le U_i \quad i = 1, \dots, n; \tag{3}$$

$$x_{ij} \in \{1,\ 2,\ 3\ \dots\} \quad i = 1, \dots, n \quad j = 1, \dots, m \tag{4}$$

where *m* is the number of shelves and *n* is the number of items. x_{ij} is 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. Constraint (2) ensures that the length of a shelf is greater than the total length of the facings assigned to this shelf. Constraint (3) ensures that the lower and upper bounds of the number of facings for each item are satisfied. Constraint (4) 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 we let $\beta_i = 1$, the model becomes a multi-knapsack problem.

3 Implementation

3.1 Hyper-heuristic

We employ a new type of hyper-heuristic, a simulated annealing based hyper-heuristic to solve our problem. In this approach, simulated annealing is used to guide the acceptance of the low-level heuristics (or neighbourhood functions) instead of finding a solution. From an initial solution, simulated annealing leads the search in a promising direction by making calls to the appropriate low-level heuristics. Twelve low-level heuristics are used. They are categorised into four types: add product(s), delete product(s), swap and interchange (see section 3.2). Two other types of hyper-heuristics are also applied to our problem and the results are compared.

3.2 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.

We use twelve low-level heuristics in our hyper-heuristics.

- ***Add_random***: add one facing of a random item to the first shelf of S_0.
- ***Add_exact***: search and assign one facing of the biggest possible item to all shelves (begin from the first shelf of S_0) until all shelves cannot hold any more items.
- ***Add_best_contribution***: repeatedly select a shelf from S_0 (begin from the first shelf in S_0), repeatedly search and add as many as possible facings of items (begin from the first item of P_{01}) until the last shelf of S_0.
- ***Add_best_improvement***: select the first shelf of S_0 and allocate one facing space to the item which gives the best improvement to the evaluation function.
- ***Delete_random***: delete one facing of a random item from a random shelf.
- ***Delete_least_contribution1***: delete one facing of the item with the least contribution value (the rightmost item in the list P_{01}) from a random shelf.
- ***Delete_least_contribution2***: delete one facing of the item with the least contribution value (the rightmost item in the list P_{01}) from all shelves.
- ***Delete_least_improvement***: delete one facing of the item that causes least decrease in the objective value from a random shelf.
- ***Swap_random***: randomly delete one facing of an item from a random shelf; add as many as possible facings of another randomly selected item.
- ***Swap_best***: repeatedly select a shelf from S_0, delete one facing of the item with the lowest contribution value (the rightmost item in the list P_{01}), add one facing of another item with a higher/highest contribution value (the leftmost item in the list P_{01}), until the last shelf is swapped.
- ***Interchange_improvement***: randomly select two different items from two random shelves and interchange 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***: randomly select two different items from two random shelves, exchange one facing of the two items.

3.3 Hyper-heuristic approaches

Three types of high-level heuristics are used to guide the selection of low-level heuristics.

3.3.1. Random heuristics

In this hyper-heuristic, a random low-level heuristic is repeatedly selected and applied to the current solution until some stopping criteria are met. Two high-level heuristics, *RHOI* (Random Heuristics Only Improving) and *RHAM* (Random Heuristics All Moves) are used during the experiments. *RHOI* only accepts a move which can improve the current solution, while *RHAM* accepts all moves.

3.3.2. A choice function based heuristics

The choice function based hyper-heuristic was proposed in [4]. In this heuristic, the selection of the low-level heuristics is guided by a *Choice Function*, which is composed of three parts: recent improvement 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). Hence it appears in the form of $CF(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j)$ where α, β and δ are respective weighs of three terms. A more detailed description is given in [4].

3.3.3. A simulated annealing based hyper-heuristics

Simulated annealing has been shown to be a powerful tool in solving a variety of combinatorial optimisation problems[12]. Based on a neighbourhood structure and a cooling schedule, a move is accepted if there is an improvement over the objective function (we are assuming we are trying to maximise the objective function). If the move generates a decrease in the objective function, it is accepted by the probability $\exp(\delta / t)$ [10]. However, in this paper, a simulated annealing heuristic is used to guide the selection and acceptance of the low-level heuristics instead of controlling the moves between neighbourhoods. In this heuristic, we repeatedly apply a random low-level heuristic and accept it based on the Metropolis's probability (simulated annealing would normally only have access to one heuristic function or neighbourhood (e.g. 2-opt in a travelling salesman problem), but we give it access to a set of heuristics). Lundy and Mees's cooling schedule [9] is employed in which the temperature is cooled according to $t \to t/(1+\beta t)$ and at each temperature only one iteration is performed. The starting temperature (t_s) is set to such a value that approximately 75% "bad heuristics" (heuristics which give a decrease in objective function) would be accepted. The stopping temperature (t_f) is 0.1. Suppose we allow $T_{allowed}$ seconds for the search and the average time spent for one iteration is $T_{average}$, we have the total number of the iterations $K = T_{allowed} / T_{average}$. After the mathematical derivation, we have

$$\beta = (t_s - t_f) / K \cdot t_s \cdot t_f = (t_s - t_f) \cdot T_{average} / T_{allowed} \cdot t_s \cdot t_f \quad (5)$$

4 Results

All algorithms were coded in Microsoft Visual C++ version 6.0 and all experiments were run on a PC Pentium IV 1800MHZ with 256MB RAM running Microsoft Windows 2000 professional Version 5. The stopping condition for all algorithms is 650 seconds CPU time. All results are averaged over 5 runs. The products and the shelf data were generated randomly by the same methods as used in [13]. Seven different problem instances, with different degrees of difficulty, were solved by our hyper-heuristic. The problems are also solved by a greedy heuristics (the greedy heuristic repeatedly adds the item with the highest *item_contribution* value). The difficulty degree of a problem is ranked by the ratio of the minimal space requirements to the total shelf space. A ratio of 1.00 means the most difficult problem which usually cannot be solved for the random data. The smaller the ratio is, the less difficult the problem (we don't consider very small value of ratio as it is not practical considering the scarce shelf space). All hyper-heuristic algorithms start from the solution created by the greedy heuristic.

Table 1 shows that our hyper-heuristics give significant improvement over the greedy heuristic. It also appears that our hyper-heuristic is suitable to solve easy problems because it gives more improvement for these problems (that is, problems with smaller ratios) than the more difficult ones. However, when the ratio goes below 0.5, the improvement reduces as well. This is probably because that, in this case, available space is so abundant that the number of the facings of most items has reached the upper bound. Therefore, the low-heuristics are constrained and cannot work efficiently. From the table 1, we also can see that the simulated annealing hyper-heuristic outperforms other hyper-heuristics approaches in most cases. It is only slightly beaten by the choice function based hyper-heuristics in one case (ratio = 0.95).

ratio (*m*, *n*)	**0.95 (12, 54)**		**0.85 (11, 48)**		**0.8 (13, 48)**		**0.7 (15, 48)**		**0.6 (16, 48)**		**0.5 (17, 48)**		**0.4 (22, 48)**	
--	**OV**	**Imp%**	**OV**	**Imp%**	**OV**	**Imp%**	**OV**	**Imp%**	**OV**	**Imp%**	**OV**	**Imp%**	**OV**	**Imp%**
Greedy heuristics	410.81	--	388.24	--	419.06	--	390.47	--	416.75	--	449.55	--	507.96	--
Choice Function Hyper-heuristics	**418.82**	**1.95**	396.50	2.13	432.24	3.15	403.87	3.58	430.63	3.33	467.02	3.89	521.90	2.74
Simulated Annealing Hyper-heuristics	418.80	1.94	**397.94**	**2.50**	**433.35**	**3.41**	**405.25**	**3.79**	**431.39**	**3.51**	**467.68**	**4.03**	**522.57**	**2.88**
RHOI	418.40	1.85	396.44	2.11	396.44	2.62	401.67	2.87	425.86	2.19	458.88	2.07	521.25	2.62
RHAM	418.54	1.88	397.06	2.27	432.49	3.21	404.51	3.60	430.50	3.30	467.01	3.88	521.34	2.63

Table 1 Algorithm performance vs different degrees of problem difficulties
m: the number of shelves, *n*: the number of items, ratio=Minimal space/Total available space
OV: Objective Value, Imp% = $((OV - OV_{greedy}) / OV_{greedy})*100\%$

5 Conclusion

In this paper, we use a more practical shelf space allocation model to generate automatic planograms. Several hyper-heuristic approaches are applied to solve this problem. As an extension of the multi-knapsack problem [13], the planogram problem is difficult to solve. We provide a set of simple low-level heuristics which have been shown to be very successful in bin packing and knapsack problem. The hyper-heuristic operates on the low-level heuristics and adapts its choice decision to the current search space. The experimental results show that the hyper-heuristics used in this paper produced much better results than a greedy heuristic and in the four hyper-heuristics, our simulated annealing based hyper-heuristic produced the best results in most cases.

One of the problems with simulated annealing is defining a suitable cooling schedule. Although our schedule appears to be effective, we plan to investigate tuning the cooling schedule to the problem in hand. We will also investigate on different problems in an attempt to demonstrate the generalisation of this approach.

References

1 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.

2 Burke, E., *A Tabu-Search Hyperheuristic for Timetabling and Rostering*. To Be Appeared

in Journal of Heuristics, 2003.

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

4 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.

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

6 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.

7 Han, L., Kendall, G. and Cowling, P., *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, 2002.

8 Levy, M. and Weitz, B., *Retailing Management*. Homewood, IL., ISBN 0-256-05989-6, 689-692, 1992.

9 Lundy, M. and Mees, A., *Convergence of An Annealing Algorithm*. Math.Prog., 34: 111-124, 1986.

10 Reeves, Colin R., *Simulated Annealing, Modern Heuristic Techniques For Combinatorial Problems.* McGraw-Hill, 1995.

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

12 Vakharia, A. J. and Chang, Y.-L., *A Simulated Annealing Approach to Scheduling a Manufacturing Cell*. NRL, 37: 559-577, 1990.

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

14 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.