Maximising store revenues using Tabu search for floor space optimisation

Jiefeng Xu*, Evren Gul and Alvin Lim

Research and Development Department,
Precima, a NielsenIQ Company,
200 W Jackson Blvd, Chicago,
IL 60606, USA
Email: jiefeng.xu@nielseniq.com
Email: evren.gul@nielseniq.com
Email: alvin.lim@nielseniq.com
*Corresponding author

Abstract: Floor space optimisation (FSO) is a critical revenue management problem commonly encountered by today's retailers. It maximises store revenue by optimally allocating floor space to product categories which are assigned to their most appropriate planograms. We formulate the problem as a connected multi-choice knapsack problem with an additional global constraint and propose a Tabu search-based metaheuristic that exploits the multiple special neighbourhood structures. We also incorporate a mechanism to determine how to combine the multiple neighbourhood moves. A candidate list strategy based on learning from prior search history is also employed to improve the search quality. The results of computational testing with a set of test problems show that our Tabu search heuristic can solve all problems within a reasonable amount of time. Analyses of individual contributions of relevant components of the algorithm were conducted with computational experiments.

Keywords: floor space optimisation; FSO; revenue management; mathematical optimisation; meta-heuristics; Tabu search.

Reference to this paper should be made as follows: Xu, J., Gul, E. and Lim, A. (2021) 'Maximising store revenues using Tabu search for floor space optimisation', *Int. J. Revenue Management*, Vol. 12, Nos. 1/2, pp.56–82.

Biographical notes: Jiefeng Xu is a Principal Data Scientist at Precima, a NielsenIQ company. He received his Master of Applied Science in Industrial Engineering from the University of Toronto and his PhD in Business Administration from the University of Colorado at Boulder. He also received his Bachelor's and Master's degrees from the Shanghai JiaoTong University. He has more than 25 years of experience in research and development and specialises in solving data-driven analytics and optimisation problems arising in many industries, such as retail, supply chain, transportation, telecommunications and others. His research work appeared in *Management Science*, *European Journal of Operational Research*, *Transportation Science*, *INFORMS Journal on Computing*, etc.

Evren Gul is a Data Science Advisor at Precima, a NielsenIQ company. He received his Master of Statistics and PhD in Industrial Engineering with concentration in Statistics from the Georgia Institute of Technology. He also obtained his Bachelor's and Master's degrees from the Middle East Technical University in Turkey. He has ten years of experience in research and development of novel statistical methods for solving complex problems and is currently focusing on solving retail merchandising problems. He is a recipient of the 2020 Statistical Partnerships Among Academe, Industry & Government (SPAIG) Award from the American Statistical Association. His recent research appeared in *Biometrika* and the *Journal of Quality Technology*.

Alvin Lim is the Chief Science Officer at Precima, a NielsenIQ company. He received his PhD in Mathematical Sciences from the Johns Hopkins University. He also has a Bachelor of Science in Mathematics and a Master Science in Applied Mathematics from the University of the Philippines, Diliman. He has over 25 years of research and development experience and specialises in the solution of pricing, promotion, product assortment, and marketing mix optimisation problems in retail and B2B applications. He is a recognised expert in the fields of revenue management, marketing analytics and transportation science.

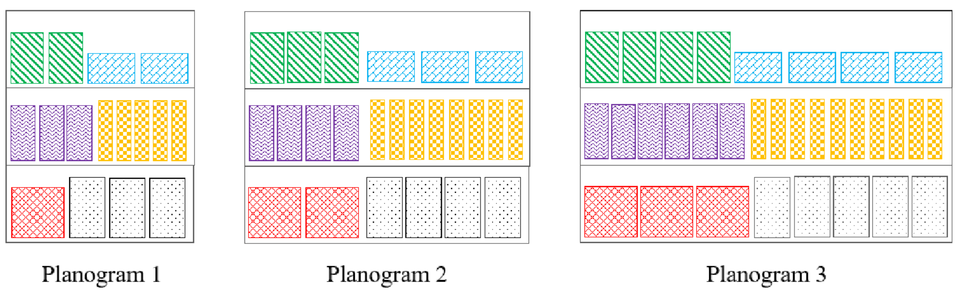
1 Introduction

Floor space is a valuable and scarce asset for retailers. Over the last decade, the number of products competing for limited space increased by up to 30% (EHI Retail Institute, 2014). Thus, the efficient allocation of store floor space to product categories to maximise the total store revenue can provide a significant edge to retailers in an increasingly competitive industry. Consequently, floor space management is considered as one of the vital strategic levers for retail revenue management (Kimes and Renaghan, 2011).

The problem we addressed is additionally motivated by the floor space planning operations of a major grocery chain in Europe. In space planning operations, the grocer benefits from software (e.g., the JDA planogram generator). Space planners of the grocer first create template planograms using software to determine product placement on shelves. A planogram is a visual diagram that shows positions and the number of so-called facings of items (corresponding to visible items). The product mix of categories, merchandising rules, sales patterns and characteristics of display furniture and fixture are considered in the preparation of planograms. An active planogram in a store is replaced by an associated planogram sequence. Figure 1 shows a sequence of three planograms on which pattern and color highlighted rectangles represent the items, their facings and locations on the shelves for a specific category. A store using the set of planograms for the category in Figure 1 can only increase or decrease the space allocation with respect to the order of the planograms.

Next, planners select a planogram for each relevant category and use software to automatically generate store layout. In the final step, planograms are physically replicated on store shelves. Using space planning software, retailers save a substantial amount of time on these space planning operations which otherwise would be carried out manually. Although space planning software provides efficient tools for visualising and preparing of store layout and planograms and day-to-day maintenance and reporting activities, they are limited in incorporating the effect of space on sales and neglect to fully take advantage of mathematical optimisation in floor space decisions. Planners decide the assignment of planograms to categories according to their experience and by employing heuristics (based on criteria sometimes called 'proportional-to-market share' and 'proportional-to-profit share'). However, these decisions can be far from optimal (Desmet and Renaudin, 1998). First of all, space elasticity, the ratio of change in sales to change in space, is not considered. Secondly, the predicted revenue associated with various planograms is not known and not used to determine the best combination of planograms in a store to maximise total revenue.

Figure 1 Sequence of planograms for a category with item facings and locations on shelves (see online version for colours)



Therefore, in our work we created an integrated solution using advanced mathematical modelling techniques in order to maximise revenue of each store. Our approach for FSO includes two steps:

1. develop a statistical model to measure the space elasticity
2. formulate and solve an optimisation problem for each store to determine the optimal assignment of planograms to maximise total revenue subject to certain business constraints.

Our statistical model is able to predict revenue of a product category for a given planogram, and in turn, the mathematical optimisation efficiently determines the best assignment of planograms to categories. In this paper, our focus is on the design and implementation of a solution approach based on Tabu search (TS) to solve the optimisation problem in step 2.

In the early work on space modelling, the emphasis was placed on establishing a relation between space and sales. Indeed, the positive impact of space allocation on sales has been documented by several studies (Curhan, 1972; Corstjens and Doyle, 1981; Bultez et al., 1989; Borin et al., 1994; Dreze et al., 1994; Desmet and Renaudin, 1998). In light of research on space modelling, we developed a statistical model to measure the effect of allocated space in a planogram on category sales focusing on the solution of the space management optimisation problem specified in step 2 mentioned above. We refer the reader to Hübner and Kuhn (2012), Eisend (2014), Kök et al. (2015) and

Bianchi-Aguiar et al. (2018) for holistic reviews of approaches to modelling of space effects.

In the second step of FSO, we maximise store revenues by optimally assigning planograms in stores. Modern revenue management launched with the airline industry and it advanced with the applications in two other traditional industries: hotels and car rental companies (Chiang et al., 2007). The success in these traditional industries attracted others and revenue management has since been applied in industries like restaurants, cargo, cruise, subscription services, theme parks and retail. More recently, revenue management has been introduced to newer industries such as cloud computing, home delivery, rideshare and e-commerce (Klein et al., 2020). Books by Talluri and Van Ryzin (2006) and Phillips (2005) provide introduction to concepts and aspects of revenue management including pricing, capacity allocation, network management, overbooking and markdown management and review papers by Chiang et al. (2007) and Klein et al. (2020) provide comprehensive surveys on the developments in revenue management over the last 40 years.

In the revenue management literature, FSO is considered as an application in the retail industry. Early applications of revenue management in retail started with seasonal items which are analogous to perishability of airline seat inventory. The value of the seasonal items diminish significantly after the selling season. Therefore, Coulter (2001) proposed the maximisation of revenue by applying optimal discount pricing to seasonal items. In another study, Aviv and Pazgal (2005) investigated the dynamic pricing for fashion-like goods for a seasonal retailer. In their second work Aviv and Pazgal (2008), they extended their work to optimal pricing of seasonal goods in the presence of strategic customers. Practitioners Hawtin (2003) and Lippman (2003) provided overall guidelines on the implementation of revenue management systems in grocery retail outlets. They mainly focused on pricing strategy and discussed potential benefits and challenges in systems integration. In the recent years, revenue management for online retailers has been studied. Agatz et al. (2013) identified differentiation in price and delivery options as a key for revenue maximisation. Belavina et al. (2017) analyzed the effect of subscription and per-order delivery revenue models on sales and environment for online grocery retailers.

As mentioned earlier, Kimes and Renaghan (2011) emphasised that space is the third strategic lever in revenue management, along with the price and time levers. They pointed out that space management is less studied in the literature, but is equally important as management of price and time. The aforementioned literature for retail revenue management mainly focuses on 'price' and 'time' levers and only includes 'space' implicitly in some of the studies, i.e., none of the research works explicitly studied the effects of space allocation in optimising revenues. Even though retailers manage time and price well, the performance will be sub-optimal in the context of revenue management if they do not manage their space well (Kimes and Renaghan, 2011). In our problem, all the levers are taken into account in the space-effect and optimisation model where we explicitly optimise space by maximising revenue per linear meter of a grocery store space for the implementation time period. FSO is an essential and integral component of any revenue management system that will enable retailers to achieve their revenue potential.

In sum, the floor space optimisation (FSO) problem considered here involves the optimal allocation of the available planograms in a store to maximise total predicted revenue. The problem is subject to constraints due to planogram sequencing, store layout, furniture and fixture characteristics. The first constraint requires that, for each existing planogram in a store, a replacement planogram should be chosen from its planogram sequence. This constraint ensures that every product category currently in the store is reassigned a planogram after the optimisation. Space planners group the planograms with respect to furniture and fixture requirements and location in the store. These groups are called planogram worlds (PWs). The second constraint requires that the sum of lengths of planograms in each PW is bounded by lower and upper total length limits. Finally, the third constraint arises from the fact that a store can accommodate space expansion up to a certain level and requires space above some threshold. Therefore, this constraint imposes lower and upper limits on the total length of all planograms in the store (additional background on the connection of FSO to other developments is given in Appendix).

Intuitively, the FSO problem seems similar to the maximisation version of the well-known knapsack problem (KP), as well as its variants such as multiple KPs, or multiple choice KPs, since we precalculate the expected revenues of planograms. If we view a planogram as an item, and a PW as a knapsack or bin, the FSO problem clearly contains the resource constraints (like the length limits) which are similar to knapsack constraints. For a comprehensive review on KP and its variants, we refer readers to Hiremath (2008). However, the additional global store length constraint of our FSO problem adds more complexities to the already complicated KP.

This paper develops a TS metaheuristic algorithm that exploits the specific neighbourhood structures of the FSO problem. Section 2 defines the mathematical formulation of the FSO problem, and then provides a relevant literature review. In Section 3, we describe the TS algorithm specifically. The computational results are included in Section 4. Finally, we summarise our findings in Section 5.

2 Mathematical formulation for FSO

The space model predicts revenue for a given space and other predictor variables of a product category in a planogram. Thus, we can define the predicted revenue of product category i when assigned planogram j, R_{ij} as

$$R_{ij} = h(s_{ij}, \mathbf{u}_{ij}),$$

where $h(\cdot)$ represents the statistical model, s_{ij} is the space assigned to category i with planogram j, and $\mathbf{u}_{ij}$ is the vector of values for other predictor variables. Note that for the purposes of this study, the R_{ij} values are precomputed and therefore assumed to be constants. The FSO problem can then be formulated as a mixed integer programming problem as follows.

2.1 Index

I Set of product categories.

J Set of planograms.

K Set of planogram worlds.

J_i Set of planograms that can be assigned to category i where $\bigcap_{i \in I} J_i = \emptyset$.

I_k Set of categories belonging to PW k.

J_k Set of planograms belonging to PW k where $J_k = \bigcup_{i \in I_k} J_i$.

2.2 Constants

R_{ij} The revenue of category i if assigned to planogram $j \in J_i$.

L_{ij} The length (shelf space) for category i if assigned to planogram $j \in J_i$.

LL_k The lower bound of the total length for PW k.

UL_k The upper bound of the total length for PW k.

LS The lower bound on sum of all planogram lengths in the entire store.

US The upper bound on sum of all planogram lengths in the entire store.

2.3 Decision variables

x_{ij} The binary variable with value 1 if category i is assigned to planogram j, and 0 otherwise.

2.4 Model

$$\text{Maximise} \quad \sum_{i \in I} \sum_{j \in J_i} R_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j \in J_i} x_{i,j} = 1, \qquad \forall i \in I, \tag{2}$$

$$\sum_{i \in I_k} \sum_{j \in J_i} L_{ij} x_{ij} \geq LL_k, \quad \forall k \in K, \tag{3}$$

$$\sum_{i \in I_k} \sum_{j \in J_i} L_{ij} x_{ij} \leq UL_k, \quad \forall k \in K, \tag{4}$$

$$\sum_{i \in I} \sum_{j \in J_i} L_{ij} x_{ij} \geq LS, \tag{5}$$

$$\sum_{i \in I} \sum_{j \in J_i} L_{ij} x_{ij} \leq US, \tag{6}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in I, j \in J_i. \tag{7}$$

The objective function (1) maximises the total store revenue which is the sum of revenues of all categories placed on planograms. The constraint (2) stipulates each category should be assigned to a planogram. The constraints (3) and (4) establish the lower and upper length limits for each PW, and constraints (5) and (6) enforce the lower and upper limits of total planogram length for the store. The constraint (7) defines the binary variable for x_{ij}. Without the presence of constraints (5) and (6), the problem would be equivalent to solving $|K|$ multiple-choice KPs.

It is well known that the KP is NP-hard (Karp, 1972). The literature on KP and its variants is rich. Exact methods have focused on employing branch and bound and dynamic programming approaches (Martello and Toth, 1985, 1997, 2003; Martello et al., 1999; Pisinger, 1995, 1997, 1999a, 1999b). A variety of heuristics and meta-heuristics have been designed for solving practical problems quickly, including those based on genetic algorithm (Chu and Beasley, 1997; Raidl, 1998), TS (Glover and Kochenberger, 1996; Lokketangen and Glover, 1998), ant colony algorithms (Shi, 2006), simulated annealing (Liu et al., 2006), global harmony search (Zou et al., 2011), etc. In addition to the papers that have proposed algorithms, Pisinger (2005) conducted an interesting study on how to design test problems that appeared to be hard for several exact methods. Since FSO embeds a multiple choice KP-like NP hard subproblem, it is natural to develop a metaheuristic-based approach such as the TS algorithm for solving this practical problem.

3 The TS algorithm for FSO

The TS algorithm is a well known metaheuristic for solving a large number of both theoretical and practical optimisation problems. It employs adaptive memory to overcome the limitation of conventional search methods such as hill-climbing, which terminate (and hence become 'trapped') in a locally optimal solution within the current neighbourhood. The common mechanisms TS employs include short-term memory, long-term memory, aspiration rule, intensification and diversification strategies. For a more comprehensive compendium of TS and its advanced strategies, we refer readers to Glover and Laguna (1997).

Our TS algorithm for FSO (denoted as TSFSO) starts from an initial solution and evaluates the objective function by calculating the revenues from planogram assignments and penalties from all length violations. Our method employs multiple simple neighbourhood structures, and determines the moves based on these neighbourhoods at each iteration through a scenario-based control mechanism (controller). To improve the efficiency of the TS and reduce the effort spent on examining inferior solutions, we devise a learning-based candidate list strategy which benefits from the statistics collected from the search history. These components are elaborated in the next few subsections.

3.1 Initial solution and objective function evaluation

Our initial solution is constructed based on the following three simple rules:

1 Least length rule: Each category is assigned to the planogram in which it occupies the least length.

  This rule ensures that violations of all upper limit length constraints will be minimised, but violations of the lower limit length constraints may occur.

2 Highest revenue rule: Each category is assigned to the planogram where it will yield the highest revenue.

  This rule will produce a solution that achieves an upper bound on the revenue that can be obtained by an optimal FSO solution. However, violations to length constraints may occur coming from both upper and lower limits of the length constraints.

3 Balanced rule: Each category is assigned to the planogram that yields the maximal revenue per length unit.

This rule will generate a more balanced solution by considering both revenue and length requirements, though it may still cause violations of length constraints.

Let $G_k(x)$ denote the total length occupied by the current assignment x, that is $G_k(x) = \sum_{i \in I_k} \sum_{j \in J_i} L_{ij} x_{ij}$. Then we combine the revenue and violation into a single objective function in the TSFSO as follows:

$$\begin{aligned} f(x) = & \sum_{i \in I} \sum_{j \in J_i} R_{ij} x_{ij} \\ & - P \left(\sum_{k \in K} (\max(0, G_k(x) - UL_k) + \max(0, LL_k - G_k(x))) \right. \\ & \left. + \max\left(0, \sum_{k \in K} G_k(x) - US\right) + \max\left(0, LS - \sum_{k \in K} G_k(x)\right) \right) \end{aligned}$$

In the above objective, P is a large positive number. When the assignment x_{ij} is changed at each iteration, the value of $f(x)$ is recalculated accordingly. Upon termination, the x_{ij} that produced the maximum value $f(x)$ is considered as the best solution. If the violation term associated with a solution is zero, then the solution is feasible.

3.2 Neighbourhood moves

The core decision of the FSO is to assign a planogram for each category to its corresponding PW. The neighbourhood move is performed by assigning different planograms to categories iteratively. We design the five basic neighbourhood moves as follows:

Level 1 move

Select a category and move it from its current planogram to another planogram. In PW k_1, let i_1 be the category under consideration, let j_1 ($j_1 \in J_{i_1}$) be the planogram that i_1 is currently assigned to, and j_2 is the new planogram ($j_2 \neq j_1$, $j_2 \in J_{i_1}$). Then the level 1 move changes the assignment $x_{i_1 j_1} = 1$x,x$_{x i_1 j_2} = 0$ to $x_{i_1 j_1} = 0, x_{i_1 j_2} = 1$. Such a move results in the following changes in objective function evaluation:

$$\begin{aligned} \Delta f(x) = & R_{i_1 j_2} - R_{i_1 j_1} - P(\max(0, G_{k_1}(x) + L_{i_1 j_2} - L_{i_1 j_1} - UL_{k_1}) \\ & + \max(0, LL_{k_1} - G_{k_1}(x) - L_{i_1 j_2} + L_{i_1 j_1}) + \max\left(0, \sum_{k \in K} G_k(x) \right. \\ & \left. + L_{i_1 j_2} - L_{i_1 j_1} - US\right) + \max\left(0, LS - \sum_{k \in K} G_k(x) - L_{i_1 j_2} + L_{i_1 j_1}\right)\Bigg) \end{aligned}$$

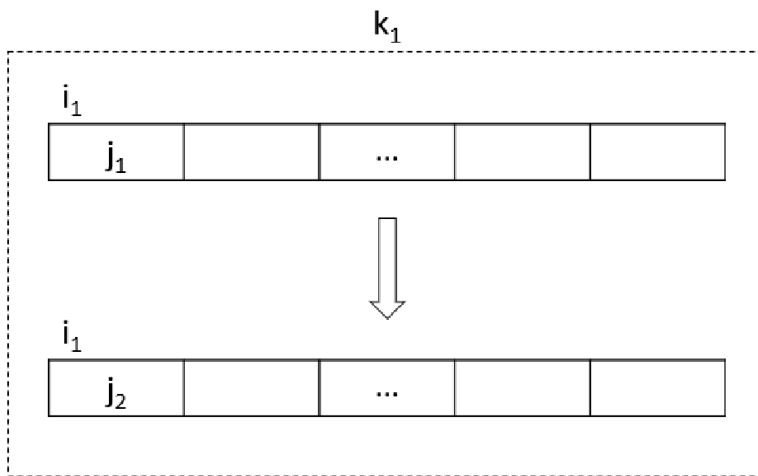
An example of a level 1 move is visually illustrated in Figure 2.

Once a level 1 move is performed, i.e., the category i_1 (in PW k_1) is moved from planogram j_1 to j_2, a Tabu restriction is applied to prevent i_1 from being moved back to j_1 within a specific number of iterations. The duration (in iterations) of such a restriction is called the $Tabu\ tenure$ of the move, and is customarily selected randomly between a lower and upper bound. After a level 1 move (moving i_1 from j_1 to j_2), the reversed move (changing $x_{i_1j_1} = 0$ to $x_{i_1j_1} = 1$ is made $Tabu$ for all iterations starting from the current iteration, $currIter$, through a last iteration $T(i, j)$ for $i = i_1$ and $j = j_1$ by the assignment:

$$T(i_1, j_1) = currIter + random(TL1, TH1)$$

where the value $random(TL1, TH1)$ identifies the Tabu tenure based on the lower and upper bounds, $TL1$ and $TH1$. The value $T(i_1, j_1)$ may be called the (short-term) $Tabu$ $memory$ for the move.

Figure 2 An example of level 1 move



Level 2 move

Select two categories from their current planograms (in PW k_1) to different planograms in the same PW. Let i_1, i_2 ($i_1 \neq i_2$) be the two categories under consideration, let j_1, j_2 ($j_1 \in J_{i_1}, j_2 \in J_{i_2}$) be the two currently assigned planograms for i_1 and i_2, and j_3 and j_4 be the new planograms ($j_3 \neq j_1$, $j_3 \in J_{i_1}$, $j_4 \neq j_2$, $j_4 \in J_{i_2}$) for i_1 and i_2. Then the level 2 move changes the decision variable values from $x_{i_1j_1} = 1$, $x_{i_2j_2} = 1$, $x_{i_1j_3} = 0$, $x_{i_2j_4} = 0$ to $x_{i_1j_1} = 0$, $x_{i_2j_2} = 0$, $x_{i_1j_3} = 1$, $x_{i_2j_4} = 1$. Such a move results in the following changes in objective function evaluation:

$$\begin{aligned}\Delta f(x) = R_{i_1j_3} + R_{i_2j_4} - R_{i_1j_1} - R_{i_2,j_2} - P(\max(0, G_{k_1}(x) + L_{i_1j_3} + L_{i_2j_4} \\ - L_{i_1j_1} - L_{i_2j_2} - UL_{k_1}) + \max(0, LL_{k_1} - G_{k_1}(x) - L_{i_1j_3} - L_{i_2j_4} \\ + L_{i_1j_1} + L_{i_2j_2}) + \max\Bigg(0, \sum_{k\in K} G_k(x) + L_{i_1j_3} + L_{i_2j_4} - L_{i_1j_1} - L_{i_2j_2}\end{aligned}$$

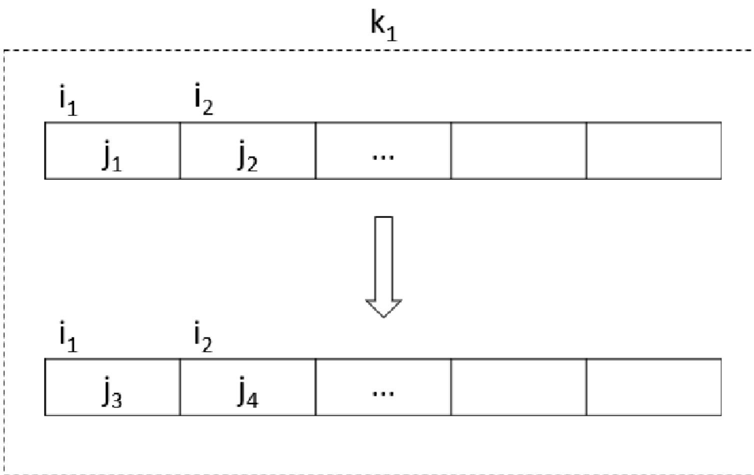
$$-US) + \max\left(0, LS - \sum_{k \in K} G_k(x) - L_{i_1 j_3} - L_{i_2 j_4} + L_{i_1 j_1} + L_{i_2 j_2}\right)\Bigg)$$

Figure 3 shows an example of a level 2 move.

The Tabu memory for a level 2 move (that moves category i_1 from planogram j_1 to j_3, and category i_2 from planogram j_2 to j_4) is to enforce $x_{i_1 j_1} = 0$ for all iterations less than or equal to $T(i_1, j_1)$ and $x_{i_2 j_2} = 0$ for all iterations less than or equal to $T(i_2, j_2)$. After such a level 2 move, the Tabu memory is updated as:

$$T(i_1, j_1) = currIter + random(TL2, TH2)$$
$$T(i_2, j_2) = currIter + random(TL2, TH2)$$

Figure 3 An example of level 2 move



Level 3 move

Reassign three categories from their current planograms (in PW k_1) to different planograms in the same PW. Let i_1, i_2 and i_3 ($i_1 \neq i_2$, $i_2 \neq i_3$, $i_3 \neq i_1$ be the three categories under consideration, let j_1, j_2 and j_3 ($j_1 \in J_{i_1}$, $j_2 \in J_{i_2}$, $j_3 \in J_{i_3}$) be the three currently assigned planograms for i_1, i_2 and i_3, j_4, j_5 and j_6 be the new planograms ($j_4 \neq j_1$, $j_5 \neq j_2$, $j_6 \neq j_3$, $j_4 \in J_{i_1}$, $j_5 \in J_{i_2}$, $j_6 \in J_{i_3}$) for these categories. Then the level 3 move changes the decision variable values from $x_{i_1 j_1} = 1$, $x_{i_2 j_2} = 1$, $x_{i_3 j_3} = 1$, $x_{i_1 j_4} = 0$, $x_{i_2 j_5} = 0$, $x_{i_3 j_6} = 0$ to $x_{i_1 j_1} = 0$, $x_{i_2 j_2} = 0$, $x_{i_3 j_3} = 0$, $x_{i_1 j_4} = 1$, $x_{i_2 j_5} = 1$, $x_{i_3 j_6} = 1$. Such a move results in the following changes in the objective function evaluation:

$$\begin{aligned}\Delta f(x) = {} & R_{i_1 j_4} + R_{i_2 j_5} + R_{i_3 j_6} - R_{i_1 j_1} - R_{i_2 j_2} - R_{i_3 j_3} - P(\max(0, G_{k_1}(x) \\ & + L_{i_1 j_4} + L_{i_2 j_5} + L_{i_3 j_6} - L_{i_1 j_1} - L_{i_2 j_2} - L_{i_3 j_3} - UL_{k_1}) \\ & + \max(0, LL_{k_1} - G_{k_1}(x) - L_{i_1 j_4} - L_{i_2 j_5} - L_{i_3 j_6} + L_{i_1 j_1} + L_{i_2 j_2}\end{aligned}$$

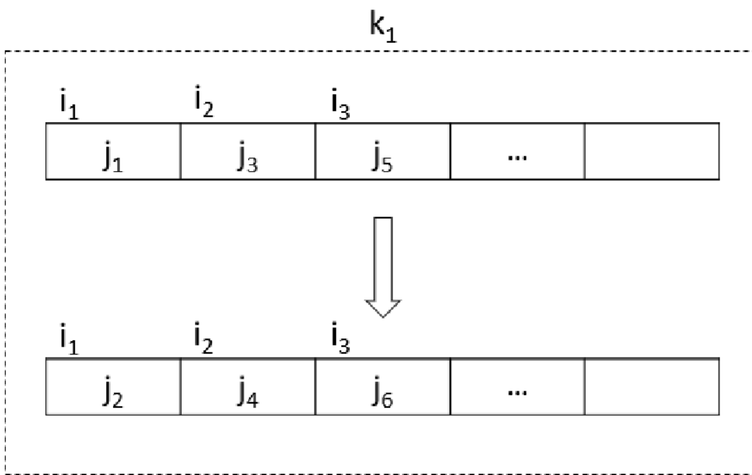
$$+ L_{i_3 j_3}) + \max\left(0, \sum_{k \in K} G_k(x) + L_{i_1 j_4} + L_{i_2 j_5} + L_{i_3 j_6} - L_{i_1 j_1} - L_{i_2 j_2}\right.$$
$$\left. - L_{i_3 j_3} - US\right) + \max\left(0, LS - \sum_{k \in K} G_k(x) - L_{i_1 j_4} - L_{i_2 j_5} - L_{i_3 j_6}\right.$$
$$\left. + L_{i_1 j_1} + L_{i_2 j_2} + L_{i_3 j_3})\right)$$

Figure 4 displays an example of a level 3 move.

Similarly, after a level 3 move (that moves the category i_1 from planogram j_1 to j_4, and category i_2 from planogram j_2 to j_5, and category j_3 from planogram j_3 to j_6) , the Tabu memory is updated by setting:

$$T(i_1, j_1) = currIter + random(TL3, TH3)$$
$$T(i_2, j_2) = currIter + random(TL3, TH3)$$
$$T(i_3, j_3) = currIter + random(TL3, TH3)$$

Figure 4 An example of level 3 move



Level 4 move

Construct a composite move that performs the two level 1 moves simultaneously for two PWs. Let i_1 (in PW k_1) and i_2 (in PW k_2) be the two categories under consideration, let j_1 ($j_1 \in J_{i_1}$) and j_2 ($j_2 \in J_{i_2}$) be the planograms that hold i_1 and i_2, respectively. Finally, let j_3 and j_4 be the new planograms for i_1 and i_2 ($j_3 \neq j_1$, $j_3 \in J_{i_1}$, $j_4 \neq j_2$, $j_4 \in J_{i_2}$) for i_1 and i_2. Then the level 4 move changes the decision variable values from $x_{i_1 j_1} = 1$, $x_{i_1 j_3} = 0$, $x_{i_2 j_2} = 1$, $x_{i_2 j_4} = 0$ to $x_{i_1 j_1} = 0$, $x_{i_1 j_3} = 1$, $x_{i_2 j_2} = 0$, $x_{i_2 j_4} = 1$. Such a move results in the following changes in the objective function evaluation:

$$\Delta f(x) = R_{i_1 j_3} - R_{i_1 j_1} + R_{i_2 j_4} - R_{i_2 j_2} - P(\max(0, G_{k_1}(x) + L_{i_1 j_3}$$
$$- L_{i_1 j_1} - UL_{k_1}) + \max(0, G_{k_2}(x) + L_{i_2 j_4} - L_{i_2 j_2} - UL_{k_2})$$

$$+ \max(0, LL_{k_1} - G_{k_1}(x) - L_{i_1j_3} + L_{i_1j_1}) + \max(0, LL_{k_2}$$
$$- G_{k_2}(x) - L_{i_2j_4} + L_{i_2j_2}) + \max\left(0, \sum_{k\in K} G_k(x) + L_{i_1j_3}\right.$$
$$- L_{i_1j_1} + L_{i_2j_4} - L_{i_2j_2} - US) + \max\left(0, LS - \sum_{k\in K} G_k(x)\right.$$
$$- L_{i_1j_3} + L_{i_1j_1} - L_{i_2j_2} + L_{i_2j_4}))$$

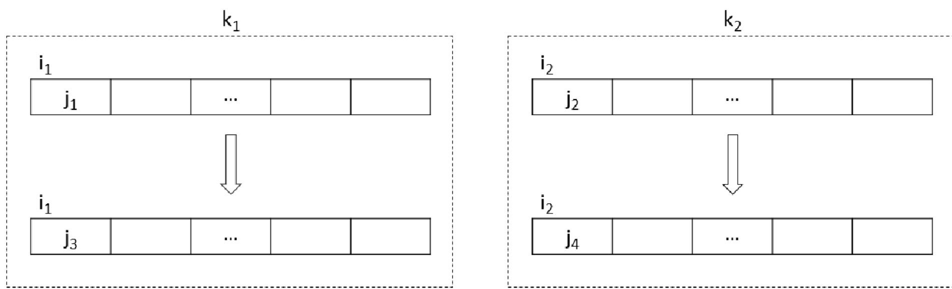
Figure 5 depicts an example of a level 4 move where categories i_1 and i_2 are assigned to new planograms in the two PWs.

The Tabu memory for a level 4 move (that moves category i_1 from planogram j_1 to j_3, and category i_2 from planogram j_2 to j_4) is updated as follows.

$$T(i_1, j_1) = currIter + random(TL4, TH4)$$
$$T(i_2, j_2) = currIter + random(TL4, TH4)$$

The difference between this and a level 2 move is that in the latter, the two categories i_1 and i_2 are located in the same PW, while in a level 4 move, they are located in the different PWs.

Figure 5 An example of level 4 move



Level 5 move

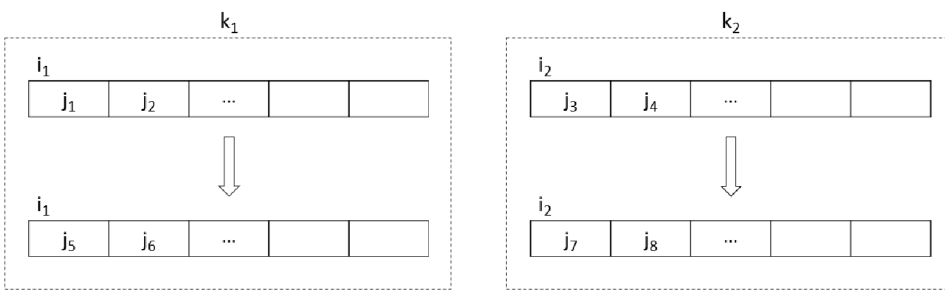
Like the level 4 move, this move involves two PWs, in each PW two categories change their current planograms to the new planograms. Let i_1, i_2 (in PW k_1) and i_3, i_4 (in PW k_2) be the categories under consideration, and let j_1 ($j_1 \in J_{i_1}$), j_2 ($j_2 \in J_{i_2}$), j_3 ($j_3 \in J_{i_3}$) and j_4 ($j_4 \in J_{i_4}$) be the planograms that hold i_1, i_2, i$_3$ and i_4, respectively. Finally, let j_5 and j_6 be the new planograms for i_1 and i_2 ($j_5 \neq j_1$, $j_5 \in J_{i_1}$, $j_6 \neq j_2$, $j_6 \in J_{i_2}$), and let j_7 and j_8 be the new planograms for i_3 and i_4 ($j_7 \neq j_3$, $j_7 \in J_{i_3}$, $j_8 \neq j_4$, $j_8 \in J_{i_4}$). Then the level 5 move changes the decision variable values from $x_{i_1j_1} = 1$, $x_{i_1j_5} = 0$, $x_{i_2j_2} = 1$, $x_{i_2j_6} = 0$, $x_{i_3j_3} = 1$, $x_{i_3j_7} = 0$, $x_{i_4j_4} = 1$, $x_{i_4j_8} = 0$ to $x_{i_1j_1} = 0$, $x_{i_1j_5} = 1$, $x_{i_2j_2} = 0$, $x_{i_2j_6} = 1$, $x_{i_3j_3} = 0$, $x_{i_3j_7} = 1$, $x_{i_4j_4} = 0$, $x_{i_4j_8} = 1$. The changes in the objective function value are identified by:

$$\Delta f(x) = R_{i_1j_5} - R_{i_1j_1} + R_{i_2j_6} - R_{i_2j_2} + R_{i_3j_7} - R_{i_3j_3} + R_{i_4j_8} - R_{i_4j_4}$$
$$- P(\max(0, G_{k_1}(x) + L_{i_1j_5} - L_{i_1j_1} + L_{i_2j_6} - L_{i_2j_2} - UL_{k_1})$$

$$+ \max(0, G_{k_2}(x) + L_{i_3j_7} - L_{i_3j_3} + L_{i_4j_8} - L_{i_4j_4} - UL_{k_2})$$
$$+ \max(0, LL_{k_1} - G_{k_1}(x) - L_{i_1j_5} + L_{i_1j_1} - L_{i_2j_6} + L_{i_2j_2})$$
$$+ \max(0, LL_{k_2} - G_{k_2}(x) - L_{i_3j_7} + L_{i_3j_3} - L_{i_4j_8} + L_{i_4j_4})$$
$$+ \max\left(0, \sum_{k\in K} G_k(x) + L_{i_1j_5} - L_{i_1j_1} + L_{i_2j_6} - L_{i_2j_2} + L_{i_3j_7}\right.$$
$$\left. - L_{i_3j_3} + L_{i_4j_8} - L_{i_4j_4} - US\right) + \max\left(0, LS - \sum_{k\in K} G_k(x)\right.$$
$$- L_{i_1j_5} + L_{i_1j_1} - L_{i_2j_6} + L_{i_2j_2} - L_{i_3j_7} - L_{i_3j_3}$$
$$\left. - L_{i_4j_8} - L_{i_4j_4}\right))$$

Figure 6 exhibits an example of a level 5 move where categories (i_1, i_2) and (i_3, i_4) are assigned to new planograms in the two PWs.

Figure 6 An example of level 5 move



Similarly, the Tabu memory for a level 5 move (that moves category i_1 from planogram j_1 to j_5, category i_2 from planogram j_2 to j_6, category i_3 from planogram j_3 to j_7, and category i_4 from planogram j_4 to j_8) is updated as:

$$T(i_1, j_1) = currIter + random(TL5, TH5)$$
$$T(i_2, j_2) = currIter + random(TL5, TH5)$$
$$T(i_3, j_3) = currIter + random(TL5, TH5)$$
$$T(i_4, j_4) = currIter + random(TL5, TH5)$$

Note that in TSFSO, once a type of neighbourhood move is determined at each iteration, all legal moves from its respective neighbourhood space are evaluated, and the move with the best evaluation subject to the corresponding Tabu restrictions. If a better solution than the best solution found so far is detected, an *aspiration rule* permits the move to be performed regardless of the Tabu memory restrictions. Other types of aspiration rules are also sometimes used, but the preceding rule is most often favoured.

The type of move selected in TSFSO is determined by a scenario-based control mechanism described next.

3.3 The scenario-based controller

Our use of multiple neighbourhood moves, where each neighbourhood provides a special structure for moving from one solution to another, is a type of strategy that is a critical element in meta-heuristics such as variable neighbourhood search (VNS) (see Mladenović and Hansen, 1997). However, multiple neighbourhoods strategies had already been successfully implemented in TS applications before the first VNS paper was published (see Glover et al., 1984; Xu et al., 1996; Xu and Kelly, 1996). In contrast to the VNS mechanism that allows the search to iterate over different neighbourhoods, in such TS applications, a special control mechanism is designed to determine when to apply a specific neighbourhood most efficiently and effectively. An example of this relevant to our current algorithm also appears in Xu et al. (1998).

In our five level neighbourhoods, the lower level neighbourhood moves are more effective for intensification to improve the current solution in regions around local optima, while the higher level neighbourhood moves cause greater structural changes that move farther from a current solution and induce diversification. On the other hand, the number of move alternatives associated with the lower level neighbourhoods is smaller than the number of alternatives associated with the higher level neighbourhoods, and so the evaluation of the former is faster than that of the latter. Based on these observations, we design the following rules for the scenario-based controller by considering the balance between intensification and diversification, together with the trade-offs for efficiency. In the rules described below, a lower numbered rule can be overwritten by a higher numbered rule if both of them are valid for a specific scenario:

Rule 1 During the first stage of the search (T_1 iterations), the level 2 neighbourhood is used to quickly improve the solution.

Rule 2 During the second stage of the search (T_2 iterations), the lower level neighbourhood moves are used probabilistically based on the probabilities as follows: p_1 for level 1 neighbourhood move, p_2 for level 2 neighbourhood move, and p_3 for level 3 neighbourhood move.

Rule 3 During the second stage of the search, whenever a new best ever solution is found, we 'downgrade' neighbourhood move type for the next iteration to reduce the current neighbourhood move by one level (this forces the search to focus more fully on intensification in an attempt to find an even better solution than the current new best solution).

Rule 4 At any iteration, if the current best solution could not be improved after a certain number of iterations denoted by T_3 (which often signals a search deadlock that requires a diversification strategy to break the impasse), a high level neighbourhood (level 4 or 5) move is required. The selection of the neighbourhood is based on the probabilities p_4 for a level 4 neighbourhood and p_5 for a level 5 neighbourhood.

Rule 5 If level 4 or 5 is selected per rule 4, then this level cannot be performed for more than T_4 consecutive iterations. Once T_4 consecutive iterations of level 4 or 5 moves are used, then the search must be switched back to a lower level neighbourhood (level 1, 2, or 3, the exact type to use is determined probabilistically by rule 2) for at least T_5 iterations (rule 5 prevents the search from focusing too long on diversification strategies when these strategies are not successful in improving the best solution. This allows the search to strategically oscillate between the intensification and diversification strategies).

Rule 6 The search terminates either when the maximum number of iterations is reached ($T_0 = T_1 + T_2$), or when the search cannot improve the current best feasible solution for T_6 iterations.

In rules 2 and 4, a randomisation-based probabilistic selection method is used for iterating between the different types of neighbourhoods. This allows the search to overcome the strong local optimality tendency produced by adhering to a single neighbourhood and amplifies the diversification effect. The scenario-triggered rules (which are invoked when a quick improvement is required during early search; a new best solution is found; the search fails to find a new best solution for a certain time, etc.) are applied to enhance the ability to exploit the special neighbourhood structures of various neighbourhoods, while accounting for issues of effectiveness and efficiency.

3.4 The learning-based candidate list strategy

To further improve the efficiency of TSFSO, we consider the fact that a good neighbourhood move may contain attribute values similar to those of good neighbourhood moves performed in the past. For this, we employ a *candidate list* to guide our choice of moves based on keeping track of attributes contained in good past moves.

We collect statistics from the moves performed at each of the first T_7 iterations in the search history and count the frequency of the PW/category combinations of the moves executed at each iteration. Let U_{ki} be the number of times that PW k and category i are involved in the moves performed so far, where initially U_{ki} is set to zero, and U_{ki} is updated at each iteration as follows.

Table 1 Move attributes statistics collection

Performed move type	*Involved PW/category*	*Update U_{ki}*
Level 1	(k, i)	$U_{ki} = U_{ki} + 1$
Level 2	(k, i_1)	$U_{ki_1} = U_{ki_1} + 1$
	(k, i_2)	$U_{ki_2} = U_{ki_2} + 1$
Level 3	(k, i_1)	$U_{ki_1} = U_{ki_1} + 1$
	(k, i_2)	$U_{ki_2} = U_{ki_2} + 1$
	(k, i_3)	$U_{ki_3} = U_{ki_3} + 1$
Level 4	(k_1, i_1)	$U_{k_1i_1} = U_{k_1i_1} + 1$
	(k_2, i_2)	$U_{k_2i_2} = U_{k_2i_2} + 1$
Level 5	(k_1, i_1)	$U_{k_1i_1} = U_{k_1i_1} + 1$
	(k_1, i_2)	$U_{k_1i_2} = U_{k_1i_2} + 1$
	(k_2, i_3)	$U_{k_2i_3} = U_{k_2i_3} + 1$
	(k_2, i_4)	$U_{k_2i_4} = U_{k_2i_4} + 1$

After the statistics U_{ki} are collected from the first T_7 iterations, we construct a candidate list at each iteration thereafter for each type of neighbourhood move performed at that iteration. The updating of the U_{ki} values is continued, however, until the end of the search. For categories I_k of PW k, let n_k be the number of categories in I_k that are included in the moves performed so far ($n_k \leq |I_k|$), and let $\bar{U}_k = \{i'_1, i'_2, ..., i'_{n_k}\}$ be a sorted list of category indices such that $U_{ki'_1}, \geq U_{ki'_2} \geq ... \geq U_{ki'_{n_k}} > 0$.

We further introduce a list $\bar{U}_k(n_k)$ containing the first n_k category indices from $\bar{U}_k$ ($p \leq n_k$) (with n_k identifying the highest U_{ki} value category). The candidate list construction is described in Table 2.

In TSFSO, the ratios r_1 and r_2 indicate the degree of reduction from the full neighbourhoods. The smaller the values for r_1 and r_2, the more compressed is the neighbourhood used for the respective candidate list.

4 Computational results

TSFSO algorithm is implemented for the real floor space planning problem arising in a leading grocery chain in Europe. To investigate the effectiveness of TSFSO, we need to execute our TSFSO on a series of test problems. Since there are no known open benchmark problems available for FSO, we first design a method to generate a series of test problems. We utilise store space configuration data from our real world implementation, as well as ideas for designing hard test problem instances for relevant KPs in literature. We describe the test problem generation method in the next subsection.

Table 2 Candidate list construction

Performed move type	*Involved PW/category*	*Candidate list*
Level 1	(k, i)	No candidate list; use the full neighbourhood, i.e., i is selected from I_k.
Level 2	(k, i_1) (k, i_2)	For PW k, if $\lvert\bar{U}_k\rvert > 0$, then i_1, i_2 are selected from the category list $\bar{U}_k(n_k * r_1)$; Otherwise, i_1, i_2 are selected from I_k.
Level 3	(k, i_1) (k, i_2) (k, i_3)	For PW k, if $\lvert\bar{U}_k\rvert > 0$, then i_1, i_2, i_3 are selected from the category list $\bar{U}_k(n_k * r_1)$; Otherwise, i_1, i_2, i_3 are selected from I_k.
Level 4	(k_1, i_1) (k_2, i_2)	For PW k_1, if $\lvert\bar{U}_{k_1}\rvert > 0$, then i_1 is selected from the category list $\bar{U}_{k_1}(n_{k_1} * r_2)$; Otherwise, i_1 is selected from IA_{s_1}. For PW k_2, if $\lvert\bar{U}_{k_2}\rvert > 0$, then i_2 is selected from the category list $\bar{U}_{k_2}(n_{k_2} * r_2)$; Otherwise, i_2 is selected from I_{k_2}.
Level 5	(k_1, i_1) (k_1, i_2) (k_2, i_3) (k_2, i_4)	For PW k_1, if $\lvert\bar{U}_{k_1}\rvert > 0$, then i_1, i_2 are selected from the category list $\bar{U}_{k_1}(n_{k_1} * r_2)$; Otherwise, i_1, i_2 are selected from I_{k_1}. For PW k_2, if $\lvert\bar{U}_{k_2}\rvert > 0$, then i_3, i_4 are selected from the category list $\bar{U}_{k_2}(n_{k_2} * r_2)$; Otherwise, i_3, i_4 are selected from I_{k_2}.

4.1 Test problem generation

We choose a common store in the grocery chain with 9 PWs and 194 planograms and use its actual PW/planogram configuration as a basis for our test problem generation. To protect confidential information such as planogram length and revenue data for the company, we generated associated length (L_{ij}) and revenue (R_{ij}) data using an approach which aims to create difficult test instances for KPs based on existing research in literature as described in Pisinger (2005). More specifically, we use a method to create the so-called weakly correlated spanner instances with span (2, 10) described in Pisinger (2005) to generate a series of coordinated pairs of length and revenue data as follows. We first generated a basic spanner set (l_κ, r_κ) for $\kappa \in \{1, 2\}$ by setting $l_\kappa = random(1, 10^7)$, and $r_\kappa = random(l_\kappa - 10^6, l_\kappa + 10^6)$ for $\kappa \in \{1, 2\}$. If the resulting $r_\kappa < 1$, we regenerate r_κ until it satisfies $r_\kappa \geq 1$. Then we normalise the spanner set by setting $l_\kappa = \lceil l_\kappa/5 \rceil, r_\kappa = \lceil r_\kappa/5 \rceil$ for $\kappa \in \{1, 2\}$. Then each pair of length and revenue numbers (L_{ij}, R_{ij}) is generated by repeatedly taking one pair (l_κ, r_κ) from the spanner set, and multiplying it by a value randomly generated from [1, 10].

In particular, we initialise the counter $k = 0$ and for each valid combination of (i, j), we identify the spanner set index by $\kappa = k \bmod 2 + 1$, $\alpha = random(1, 10)$. The length and revenue numbers are then calculated as $L_{i,j} = \alpha l_k$ and $R_{ij} = \alpha r_k$, followed by incrementing the counter k by setting $k = k + 1$. We repeat the process until all possible length and revenue values (L_{ij}, R_{ij}) are generated for the required 194 planograms.

Unlike the method suggested in Pisinger (2005) where the length bounds are generated based on the generated lengths using different ratios across test instances (e.g., bounds are increased by a fixed percentage for each instance), however, we generated the lower and upper bounds for PWs and store levels for our test problems by multiplying fixed ratios (0.85 for lower bound and 1.15 for upper bound) with total (generated) lengths calculated based on the current planogram assignment. This approach may reduce the difficulty of the test problems but ensures the generated problems are feasible and bounds are consistent with the current store FSO planning practice as well. Nevertheless, the test problems we generated are still computationally more difficult than the real problem we encountered in practice, indicating that they are good problems to stress-test our TSFSO heuristic and to have some confidence that our heuristic can cover realistic problems we have yet to encounter.

We repeat the above process using different random seeds until we generate 100 test problems. To execute our TSFSO on these test problems, we use the parameter values described in the next subsection.

4.2 Parameter setting

We set the values for the parameters based on a priori knowledge, as well as on the results from limited brute-force experiments. Applying a systematic parameter fine-tuning method based on statistical analysis and experiment design techniques may significantly improve the heuristic performance (see Xu et al., 1998 for an example).

First, the penalty P for length violations (at both the PW level and store level) is set to –20,000. In this application, the penalty function is implemented as a static value, heavily emphasising feasibility over solution quality by strongly favouring feasible neighbourhood moves, and hence focusing more intensification than diversification. We plan in a future work to design a dynamic, self-adaptive penalty parameter, to better explore the interplay between intensification and diversification.

Tabu tenure parameters are set as follows: we unify such parameters for various neighbourhood move types by designating a single lower bound and a single upper bound. Such bounds are related to the respective neighbourhood spaces so they can be resiliently applied to small, medium or large neighbourhoods. Specifically, for a given PW k, we set the lower bound value $TL = TL1 = TL1 = TL3 = TL4 = TL5 = \max\{4, |I_k|/2\}$, and the upper bound value $TH = TH1 = TH1 = TH3 = TH4 = TH5 = TL + \min\{7, |I_k|/7\}$.

Several parameters govern the controller and the search progress. The entire search consists of 1,200 iterations (T_0 = 1,200), while the first stage of the search starts from iteration 1 to iteration T_1 = 120 and the second stage then invokes for the reset of T_2 = 1,080 iterations. The parameter T_3, which triggers the condition for using high level neighbourhood moves, is set to 20 iterations. The high level neighbourhood moves can be performed no more than 2 consecutive iterations (T_4 = 2) and the next 10 iterations (T_5 = 10) will be dedicated to low level neighbourhood moves after 2 consecutive iterations for high level neighbourhood moves. The TSFSO terminates either when the iteration counter reaches T_0 = 1,200, or when the current (feasible) best solution cannot be improved within the most recent $T_6 = 0.8 * T_0$ iterations.

The probabilities used for selecting the different types of neighbourhood moves are: p_1 = 0.2, p_2 = 0.5, p_3 = 0.3, p_4 = 0.6, p_5 = 0.4. Note that $p_1 + p_2 + p_3$ = 1.0 and $p_4 + p_5$ = 1.0.

Lastly, the candidate list strategy collects statistics from attributes of moves performed during the first 100 (T_7 = 100) iterations. Consequently, the candidate list strategy is initiated at the 101st iteration. The two ratios used to reduce the neighbourhood spaces, are set to r_1 = 0.5, r_2 = 0.8, respectively.

4.3 Computational experiments and results analysis

The TSFSO is implemented using the Python language and is executed on a Linux machine on cloud. The CPU model of the machine is Intel® Xeon® CPU E5-2686 v4 @ 2.30GHz. To compare the effectiveness of TSFSO, we first use Gurobi, one of the leading commercial solver packages for mixed integer programming, to solve the 100 tests problems to optimality using the same computational environment. We limit the computation time to 200 seconds for each problem. Gurobi either finds an optimal solution before reaching the time limit or terminates with a best solution obtained at the time limit, which we designate it as the *de facto* optimal solution, instead of classifying it as an optimal solution.

To evaluate the different time effect of TSFSO, we set the maximum number of iterations (T_0 in the four test runs of TSFSO to be 300, 600, 900, and 1,200). The associated test runs are accordingly denoted TSFSO-300, TSFSO-600, TSFSO-900, TSFSO-1,200. All other parameters remain the same as previously described with the following features implemented:

- used the balanced rule as the initial solution rule
- used all five level moves coordinated by the scenario-based controller
- used candidate list.

In summarising the results from across the 100 problems, we report the number of problems for which we obtain an optimal or *de facto* optimal solution (as OPTNUM), the average percentage of the optimality gap across all 100 problems (as AVGGAP), maximum percentage of the optimality gap (as MAXGAP), and the total CPU time in seconds (as CPUTM) in Table 3.

Table 3 Computational results on 100 test problems

RUN	*OPTNUM*	*AVGGAP (%)*	*MAXGAP (%)*	*CPUTM*
TSFSO-300	35	0.23	0.96	59
TSFSO-600	68	0.08	0.52	154
TSFSO-900	83	0.04	0.52	217
TSFSO-1,200	100	0.00	0.00	312
Gurobi	100	0.00	0.00	12,345

Table 4 Comparison of initial solution rules

RUN	*OPTNUM*	*AVGGAP (%)*	*MAXGAP (%)*	*CPUTM*
TSFSO-LL	39	0.22	0.91	310
TSFSO-HR	26	0.33	1.40	314
TSFSO-1,200	100	0.00	0.00	312

Table 5 Comparison of neighbourhoods and candidate list strategy

RUN	*OPTNUM*	*INFNUM*	*AVGGAP (%)**	*MAXGAP (%)**	*CPUTM*
TSFSO-1,200	100	0	0.00	0.00	312
TSFSO-1	0	46	3.25	19.20	4
TSFSO-2	0	100	N/A	N/A	109
TSFSO-3	0	100	N/A	N/A	743
TSFSO-4	0	25	1.81	19.40	99
TSFSO-5	0	100	N/A	N/A	8,272
TSFSO-NC	91	0	0.02	0.48	2,731

Note: *applies to feasible cases only.

Table 3 clearly demonstrates with the reported reasonable computational times that our TSFSO algorithm can overcome the computational intractability of the test problems and obtain exceedingly high quality solutions using just a fraction of the time required by the Gurobi solver. Although for iteration limits below 1,200, not all optimal solutions were attained, the maximum percentage gap is quite small at less than 1%, and these solutions were obtained by our TSFSO at significantly less computational time versus Gurobi. This further confirms that TSFSO can be used as an effective practical optimisation method for solving FSO problems without relying on commercial proprietary solvers such as Gurobi. It should be noted that our TSFSO was implemented in Python, which is generally considered slower in computational speed as compared to the C language that Gurobi was implemented on.

Next, we run experiments on our TSFSO to determine the effects of the three different initial solution rules described in Subsection 3.1. We denote the TSFSO using the least length rule, the highest revenue rule, and the balanced rule as TSFSO-LL, TSFSO-HR, and TSFSO-1,200, respectively. All such TSFSO variants use 1,200 iterations which offers sufficient time to improve the solution quality using TS.

The results in Table 4 clearly show that TSFSO with different initial solution rules can find very high quality solutions for FSO. Compared to the optimal (or *de facto* optimal) solutions, the TSFSO using the balanced rules for generating initial solutions yields all optimal solutions for all cases, and the one using the least length rule for initial solutions obtains 39 optimal solutions out of 100 cases. The TSFSO using the highest revenue rule-based initial solutions lags behind by obtaining 26 optimal solutions; however, the worst solution is only 1.4% away from optimality (on average it is 0.33%), indicating it can still produce reliable high quality solutions for practical applications.

By carefully monitoring and comparing the search progress, we cannot conclude that the computation time is significantly sensitive to the choice of initial solution method. No matter which initial solution method is adapter, the TSFSO beats Gurobi by an obvious edge in computation time.

We use TSFSO-1,200 as a basis for examining the effects of using multiple neighbourhoods in TSFSO by comparing to the tests from those using a single type of neighbourhood for level 1, level 2, level 3, level 4 and level 5 moves, denoting these tests by TSFSO-1, TSFSO-2, TSFSO-3, TSFSO-4 and TSFSO-5 in Table 5. We also design a new test, designated TSFSO-NC, that deactivates the learning-based candidate list in TSFSO-1,200. In addition to the values OPTNUM, AVGGAP, MAXGAP and CPUTM reported in the previous table, we also show in the column INFNUM the number of cases in which the runs fail to find a feasible solution when the algorithm terminated.

It is obvious in Table 5 that the TSFSO version that uses multiple neighbourhoods performs better than the versions equipped with a single type of neighbourhood. None of the single neighbourhood versions can compete with TSFSO-1,200 in terms of the number of optimal/feasible solution obtained. As shown, the level 1 neighbourhood move is the fastest but is able to obtain feasible solutions only for 54 cases. Moreover, it yields not a single optimal solution, and the average optimality gap of 3.25%. Level 4 neighbourhood moves, which perform two level 1 moves simultaneously, provide significant improvements in term of the feasible solutions obtained. The levels 2, 3 and 5 neighbourhood moves, each of which changes more than one assignment for one or two PWs, can hardly find feasible solutions by solely relying on their own effort. This confirms that each type of move has its own unique merits and disadvantages for handling special problem structures. In combination, they complement each other to find superior solutions, as shown in our baseline TSFSO-1,200 version.

Surprisingly, the high level level 4 neighbourhood moves perform slightly faster than the lower level level 2 moves. We attribute this mainly to the fact that the level 4 moves enable the algorithm to reach a feasible local optimal solution quickly in 75 cases. In each iteration, it may move to the first improving best solution without examining the whole neighbourhood space. In contrast, the level 2 move version struggles to achieve feasibility, so it consequently searches the entire neighbourhood space at each iteration, and finally terminates without finding a feasible solution.

The findings from the comparisons between TSFSO-1,200 and TSFSO-NC also confirm that the learning-based candidate list strategy can effectively improve efficiency using only 11.4% of the computation time required by the version without learning-based candidate list strategy (312 seconds versus 2,731 seconds). It is also interesting to note that the full neighbourhood search performed in TSFSO-NC does not always yield optimal solutions, though in the nine cases it obtained solutions with exceedingly small optimality gaps.

Next, we examine the effects of Tabu memory in TSFSO. We again use TSFSO-1,200 as the base case and obtain varying cases to experiment as follows:

a no Tabu memory is applied

b all Tabu tenures are set to 4

c all Tabu tenures are set to 7

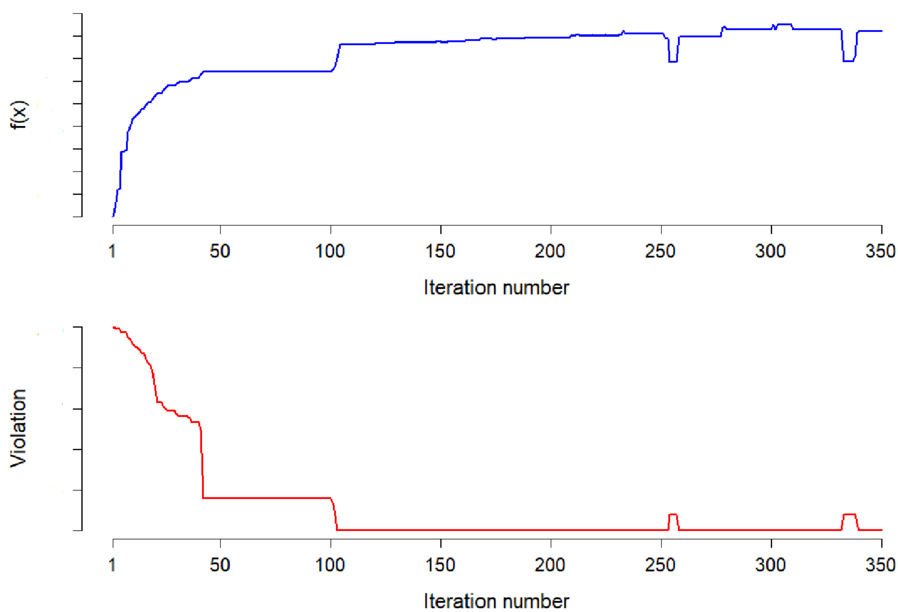
d $TL = TH = (\max\{4, |I_k|/2\} + \min\{7, |I_k|/7\})/2$.

The corresponding new tests are denoted TSFSO-a, TSFSO-b, TSFSO-c and TSFSO-d. Table 6 contains the results from these tests along with the baseline run TSFSO-1,200.

Table 6 Comparison of different Tabu memories

RUN	*OPTNUM*	*AVGGAP (%)*	*MAXGAP (%)*	*CPUTM*
TSFSO-1,200	100	0	0	312
TSFSO-a	0	0.87	2.5	458
TSFSO-b	16	0.39	1.5	377
TSFSO-c	32	0.25	1.05	339
TSFSO-d	47	0.0.21	1.93	324

Figure 7 A search progress example (see online version for colours)



As shown in Table 6, all tests obtain feasible solutions with marginal optimality gaps. This can be partially attributed to the efficiency of the multiple neighbourhood search method designed for TSFSO. The contribution of the Tabu memory to this outcome is quite noticeable. However, when the Tabu memory is removed (TSFSO-a), the algorithm fails to obtain any optimal solutions in any of the 100 cases. Upon implementing various Tabu memory methods (TSFSO-b, TSFSO-c, TSFSO-d), the number of optimal solutions improved and the optimality gaps diminished for non-optimal cases. This demonstrates the value of Tabu memory for improving an already good multiple neighbourhood procedure by providing the ability to escape from strong locally optimal solutions. The baseline version of our algorithm (TSFSO-1,200) which determines the Tabu tenure as indicated in Subsection 4.2 clearly outperforms the others.

In addition, based on the results shown in Table 6, Tabu memory not only plays a critical role in obtaining optimal or near optimal solutions, but also is helpful in improving the efficiency with reduced computational time. This finding is consistent with our observation that those TSFSO variants without Tabu memory or with simple implementation of Tabu memory intend to use more complicated and time consuming neighbourhood move types (i.e., type 3 or type 5 moves), since they may encounter search impasse more frequently. An appropriate design of Tabu memory is one of the key factors for practical FSO applications that requires obtaining high quality solutions within reasonable amount of computing time.

Finally, we demonstrate an example of the search progression of TSFSO for a selected run from our practical implementation in Figure 7. In this figure, the top and bottom charts show the value of the objective function $f(x)$ as well as the sum of the length constraint violations of the move performed at each iteration (a zero violation indicates a feasible solution). The horizontal axis represents the iteration number, and the values of the vertical axis [$f(x)$ and violations] are transformed and re-scaled artificially (without loss of trends) for better illustrative purposes. Note the maximum number of iterations T_0 is set to 350 in this run.

Figure 7 demonstrates that the TSFSO can rapidly improve solution quality after starting from an initial solution with a large feasibility violation. With the help of the large penalty, the solution becomes feasible at 103rd iteration, and continues to improve via the multiple neighbourhood moves. A locally optimal solution is obtained at the 233rd iteration. Then the search continues, escaping the local optimum with the assistance of Tabu memory. The search falls into an infeasible region at the 254th iteration, and moves back into a feasible region at the 258th iteration. The algorithm finds two new best solutions at the 278th and 301st iterations, and then again enters an infeasible region between iterations 333 and 340. No new best solutions were found throughout the remainder of the search, which terminated at the 350th iteration.

5 Conclusions

We studied the FSO problem arising in retail store revenue management by providing a mathematical formulation, conducting a review of relevant research and proposing a new solution approach based on TS. Our TS algorithm contains several innovative components, including a scenario-based controller for combining the multiple neighbourhoods, and a candidate list strategy that utilises learning based on statistics collected from prior search history.

We applied our TSFSO on 100 test problems that combined the practical configuration from real world applications as well as results from computational studies in literature. We reported results demonstrating that our method is highly effective and efficient in handling the test problems. Analysis of the experiment results further confirms the value of our scenario-based controller and our learning-based candidate list strategy. The use of the TSFSO in a practical application to a grocery chain improved its predicted annual revenue by around 1%, which amounts to approximately 80 million Euros.

Future research work will continue in two directions: model improvement and algorithm improvement. First, we plan to enrich the space-effect and optimisation models by considering planogram orientations, relative positions and layout. Mowrey et al. (2018, 2019) studied the impact of layout and orientation of racks in a store on customer's navigation and exposure which provide guidance on how to allocate planograms in a store to maximise revenue. Because space is relatively fixed and scarce in a store, the inclusion of these additional aspects will improve the forecasting from the space-effect model and provide additional depth to the optimisation model. Secondly, we will further improve the efficacy of TSFSO. Among the improvements envisioned, we plan to introduce a self-adaptive penalty function to replace the current reliance on a fixed penalty value that penalises feasibility violations. We also plan to study other potential improvements such as the analysis of the patterns of moves that prove most effective in order to further improve the search efficiency, the investigation of more advanced mechanisms to control the interplay between intensification and diversification, and the exploration of path relinking strategies to construct and exploit search trajectories between good solutions.

Acknowledgements

The authors thank Professor Fred Glover for reviewing an early draft of this paper and providing helpful comments. The authors also thank the editor and two anonymous reviewers for their detailed and helpful comments.

References

Agatz, N., Campbell, A.M., Fleischmann, M., Van Nunen, J. and Savelsbergh, M. (2013) 'Revenue management opportunities for internet retailers', *Journal of Revenue and Pricing Management*, Vol. 12, No. 2, pp.128–138.

Aviv, Y. and Pazgal, A. (2005) 'A partially observed markov decision process for dynamic pricing', *Management Science*, Vol. 51, No. 9, pp.1400–1416.

Aviv, Y. and Pazgal, A. (2008) 'Optimal pricing of seasonal products in the presence of forward-looking consumers', *Manufacturing & Service Operations Management*, Vol. 10, No. 3, pp.339–359.

Belavina, E., Girotra, K. and Kabra, A. (2017) 'Online grocery retail: revenue models and environmental impact', *Management Science*, Vol. 63, No. 6, pp.1781–1799.

Bianchi-Aguiar, T., Silva, E., Guimarães, L., Carravilla, M.A. and Oliveira, J.F. (2018) 'Allocating products on shelves under merchandising rules: multi-level product families with display directions', *Omega*, Vol. 76, No. 4, pp.47–62.

Bianchi-Aguiar, T., Silva, E., Guimarães, L., Carravilla, M.A., Oliveira, J.F., Amaral, J.G., Liz, J. and Lapela, S. (2016) 'Using analytics to enhance a food retailer's shelf-space management', *Interfaces*, Vol. 46, No. 5, pp.424–444.

Borin, N., Farris, P.W. and Freeland, J.R. (1994) 'A model for determining retail product category assortment and shelf space allocation', *Decision Sciences*, Vol. 25, No. 3, pp.359–384.

Bultez, A., Naert, P., Gijsbrechts, E. and Abeele, P.V. (1989) 'Asymmetric cannibalism in retail assortments', *Journal of Retailing*, Vol. 65, No. 2, p.153.

Chiang, W-C., Chen, J.C. and Xu, X. (2007) 'An overview of research on revenue management: current issues and future research', *International Journal of Revenue Management*, Vol. 1, No. 1, pp.97–128.

Chu, P.C. and Beasley, J.E. (1997) 'A genetic algorithm for the generalised assignment problem', *Computers & Operations Research*, Vol. 24, No. 1, pp.17–23.

Corstjens, M. and Doyle, P. (1981) 'A model for optimizing retail space allocations', *Management Science*, Vol. 27, No. 7, pp.822–833.

Coulter, K.S. (2001) 'Decreasing price sensitivity involving physical product inventory: a yield management application', *Journal of Product & Brand Management*, Vol. 10, No. 5, pp.301–317.

Curhan, R.C. (1972) 'The relationship between shelf space and unit sales in supermarkets', *Journal of Marketing Research*, Vol. 9, No. 4, pp.406–412.

Desmet, P. and Renaudin, V. (1998) 'Estimation of product category sales responsiveness to allocated shelf space', *International Journal of Research in Marketing*, Vol. 15, No. 5, pp.443–457.

Dreze, X., Hoch, S.J. and Purk, M.E. (1994) 'Shelf management and space elasticity', *Journal of Retailing*, Vol. 70, No. 4, pp.301–326.

EHI Retail Institute (2014) *Retail Data 2014: Structure, Key Figures and Profiles of International Retailing*, White Paper, EHI Retail Institute, Cologne.

Eisend, M. (2014) 'Shelf space elasticity: a meta-analysis', *Journal of Retailing*, Vol. 90, No. 2, pp.168–181.

Flamand, T., Ghoniem, A., Haouari, M. and Maddah, B. (2018) 'Integrated assortment planning and store-wide shelf space allocation: an optimization-based approach', *Omega*, Vol. 81, No. 12, pp.134–149.

Ghoniem, A., Flamand, T. and Haouari, M. (2016) 'Optimization-based very large-scale neighborhood search for generalized assignment problems with location/allocation considerations', *INFORMS Journal on Computing*, Vol. 28, No. 3, pp.575–588.

Glover, F. and Kochenberger, G.A. (1996) 'Critical event Tabu search for multidimensional knapsack problems', in Osman, I.H. and Kelly, J.P. (Eds): *Meta-heuristics*, pp.407–427, Springer, Boston, MA.

Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer Academic Publishers, Boston/Dordrecht/London.

Glover, F., McMillan, C. and Glover, R. (1984) 'A heuristic programming approach to the employee scheduling problem and some thoughts on 'managerial robots'', *Journal of Operations Management*, Vo. 4, No. 2, pp.113–128.

Hansen, P. and Heinsbroek, H. (1979) 'Product selection and space allocation in supermarkets', *European Journal of Operational Research*, Vol. 3, No. 6, pp.474–484.

Hawtin, M. (2003) 'The practicalities and benefits of applying revenue management to grocery retailing, and the need for effective business rule management', *Journal of Revenue and Pricing Management*, Vol. 2, No. 1, pp.61–68.

Hiremath, C. (2008) *New Heuristic and Metaheuristic Approaches Applied to the Multiplechoice Multidimensional Knapsack Problem*, PhD thesis, Wright State University.

Hübner, A.H. and Kuhn, H. (2012) 'Retail category management: state-of-the-art review of quantitative research and software applications in assortment and shelf space management', *Omega*, Vol. 40, No. 2, pp.199–209.

Hübner, A. and Schaal, K. (2017) 'An integrated assortment and shelf-space optimization model with demand substitution and space-elasticity effects', *European Journal of Operational Research*, Vol. 261, No. 1, pp.302–316.

Irion, J., Lu, J-C., Al-Khayyal, F. and Tsao, Y-C. (2012) 'A piecewise linearization framework for retail shelf space management models', *European Journal of Operational Research*, Vol. 222, No. 1, pp.122–136.

Karp, R.M. (1972) 'Reducibility among combinatorial problems', in Miller, R.E., Thatcher, J.W. and Bohlinger, J.D. (Eds.): *Complexity of Computer Computations*, pp.85–103, The IBM Research Symposia Series. Springer, Boston, MA.

Kimes, S.E. and Renaghan, L.M. (2011) 'The role of space in revenue management', in Yeoman, I. and McMahon-Beattie, U. (Eds.): *Revenue Management*, pp.17–28, Palgrave Macmillan, London.

Klein, R., Koch, S., Steinhardt, C. and Strauss, A.K. (2020) 'A review of revenue management: recent generalizations and advances in industry applications', *European Journal of Operational Research*, Vol. 284, No. 2, pp.397–412.

Kök, A.G., Fisher, M.L. and Vaidyanathan, R. (2015) 'Assortment planning: review of literature and industry practice', in Agrawal, N. and Smith, S.A. (Eds.): *Retail Supply Chain Management*, pp.175–236, Springer, New York, NY.

Lim, A., Rodrigues, B. and Zhang, X. (2004) 'Metaheuristics with local search techniques for retail shelf-space optimization', *Management Science*, Vol. 50, No. 1, pp.117–131.

Lippman, B.W. (2003) 'Retail revenue management – competitive strategy for grocery retailers', *Journal of Revenue and Pricing Management*, Vol. 2, No. 3, pp.229–233.

Liu, A., Wang, J., Han, G., Wang, S. and Wen, J. (2006) 'Improved simulated annealing algorithm solving for 0/1 knapsack problem', in *Sixth International Conference on Intelligent Systems Design and Applications*, IEEE, Vol. 2, pp.1159–1164.

Lokketangen, A. and Glover, F. (1998) 'Solving zero-one mixed integer programming problems using Tabu search', *European Journal of Operational Research*, Vol. 106, Nos. 2–3, pp.624–658.

Martello, S., Pisinger, D. and Toth, P. (1999) 'Dynamic programming and strong bounds for the 0–1 knapsack problem', *Management Science*, Vol. 45, No. 3, pp.414–424.

Martello, S. and Toth, P. (1985) 'Algorithm 632: a program for the 0–1 multiple knapsack problem', *ACM Transactions on Mathematical Software (TOMS)*, Vol. 11, No. 2, pp.135–140.

Martello, S. and Toth, P. (1997) 'Upper bounds and algorithms for hard 0–1 knapsack problems', *Operations Research*, Vol. 45, No. 5, pp.768–778.

Martello, S. and Toth, P. (2003) 'An exact algorithm for the two-constraint 0–1 knapsack problem', *Operations Research*, Vo. 51, No. 5, pp.826–835.

Mladenović, N. and Hansen, P. (1997) 'Variable neighborhood search', *Computers & Operations Research*, Vol. 24, No. 11, pp.1097–1100.

Mowrey, C.H., Parikh, P.J. and Gue, K.R. (2018) 'A model to optimize rack layout in a retail store', *European Journal of Operational Research*, Vol. 271, No. 3, pp.1100–1112.

Mowrey, C.H., Parikh, P.J. and Gue, K.R. (2019) 'The impact of rack layout on visual experience in a retail store', *INFOR: Information Systems and Operational Research*, Vol. 57, No. 1, pp.75–98.

Phillips, R.L. (2005) *Pricing and Revenue Optimization*, Stanford University Press, Stanford, CA.

Pisinger, D. (1995) 'A minimal algorithm for the multiple-choice knapsack problem', *European Journal of Operational Research*, Vol. 83, No. 2, pp.394–410.

Pisinger, D. (1997) 'A minimal algorithm for the 0–1 knapsack problem', *Operations Research*, Vol. 45, No. 5, pp.758–767.

Pisinger, D. (1999a) 'Core problems in knapsack algorithms', *Operations Research*, Vol. 47, No. 4, pp.570–575.

Pisinger, D. (1999b) 'An exact algorithm for large multiple knapsack problems', *European Journal of Operational Research*, Vol. 114, No. 3, pp.528–541.

Pisinger, D. (2005) 'Where are the hard knapsack problems?', *Computers & Operations Research*, Vol. 32, No. 9, pp.2271–2284.

Raidl, G.R. (1998) 'An improved genetic algorithm for the multiconstrained 0–1 knapsack problem', in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, IEEE, pp.207–211.

Shi, H. (2006) 'Solution to 0/1 knapsack problem based on improved ant colony algorithm', in *2006 IEEE International Conference on Information Acquisition*, IEEE, pp.1062–1066.

Talluri, K.T. and Van Ryzin, G.J. (2006) *The Theory and Practice of Revenue Management*, Springer-Verlag. New York.

Urban, T.L. (1998) 'An inventory-theoretic approach to product assortment and shelf-space allocation', *Journal of Retailing*, Vol. 74, No. 1, pp.15–35.

Xu, J. and Kelly, J.P. (1996) 'A network flow-based Tabu search heuristic for the vehicle routing problem', *Transportation Science*, Vol. 30, No. 4, pp.379–393.

Xu, J., Chiu, S.Y. and Glover, F. (1996) 'Using Tabu search to solve the steiner tree-star problem in telecommunications network design', *Telecommunication Systems*, Vol. 6, No. 1, pp.117–125.

Xu, J., Chiu, S.Y. and Glover, F. (1998) 'Fine-tuning a Tabu search algorithm with statistical tests', *International Transactions in Operational Research*, Vol. 5, No. 3, pp.233–244.

Yang, M-H. (2001) 'An efficient algorithm to allocate shelf space', *European Journal of Operational Research*, Vol. 131, No. 1, pp.107–118.

Zou, D., Gao, L., Li, S. and Wu, J. (2011) 'Solving 0–1 knapsack problem by a novel global harmony search algorithm', *Applied Soft Computing*, Vol. 11, No. 2, pp.1556–1564.

Zufryden, F.S. (1986) 'A dynamic programming approach for product selection and supermarket shelf-space allocation', *Journal of the Operational Research Society*, Vol. 37, No. 4, pp.413–422.

Appendix

In the literature, the closest analogue to FSO in terms of optimisation model is the shelf space allocation problem (SSAP). In SSAP, the aim is to optimally allocate space to items in shelves subject to facing constraints for items and space constraints for shelves. Although the problems are analogous, they have noticeable differences. In FSO, we do not make decisions at the item level, but rather at the planogram level as we transform the problem from product categories to planograms. Therefore, FSO results are store-wide, managerially useful, immediately implementable, and fully integrated with the current operations of the grocer. On the other hand, in SSAP, the outcome of every optimisation creates new item allocations which translate into new planograms. Retailers can carry up to 60,000 products in a store (EHI Retail Institute, 2014). These products are carried in thousands of candidate planograms which are created with several criteria. New planogram creation and implementation is a time and labour-intensive task, and planners can only prepare a limited number of new planograms in each planning cycle; hence, retailers prefer to utilise already available planograms.

Moreover, the demand model in SSAP is generally characterised deterministically with nonlinear terms. In turn, along with additional effects such as cross space elasticity and integer decision variables (representing the number of facings allocated to an item), the problem can be exceedingly hard to solve. Heuristics or linearisation techniques are employed to simplify the problem, and as a result, the quality of solutions produced and/or the structure of the problem instance considered are restricted.

In early work, Hansen and Heinsbroek (1979) proposed a generalised Lagrange multiplier technique to solve the problem and implemented it on a large problem instance. Corstjens and Doyle (1981) used geometric programming and Zufryden (1986) developed a dynamic programming framework to tackle the problem on small problem instances. Borin et al. (1994) and Urban (1998) were the first to utilise meta-heuristics. While a dynamic programming algorithm was used by Borin, a genetic algorithm and a generalised reduced gradient algorithm were developed by Urban to solve problems with up to 54 items. Yang (2001) assumed a predefined linear profit per facing for each item and formulated SSAP as a linear multi-KP and solved the problem for simple instances. Later Lim et al. (2004) built their approach on Yang's linear model and solved the problem with meta-heuristics for sizes ranging up to one hundred items. In a different approach, Irion et al. (2012) used a piecewise linearisation framework for approximating the nonlinear model and then transformed the problem into a mixed integer program (MIP). Their MIP solution provided near optimal solutions for single category-shelf space optimisation with relevant practical sizes. In two associated studies, Bianchi-Aguiar et al. (2016, 2018) solved SSAP with different problem characteristics using MIP-based heuristics. Recently, Hübner and Schaal (2017) assumed stochastic demand and formulated SSAP as a MIP and then solved the problem with a multistage heuristic up to 2,000 items.

None of the aforementioned studies addressed the store-wide space optimisation problem. In contrast, Ghoniem et al. (2016) and Flamand et al. (2018) treated the category space allocation problem at the store-wide level. These two studies do not consider an in-depth space model; instead, they both generate a linear profitability index depending on some metrics and allocate space to categories continuously. Ghoniem et al. framed the problem as a generalised assignment problem and tackled the problem with a neighbourhood search algorithm. In yet another approach, Flamand et al. solved the store-wide MIP problem with an optimisation-based heuristics.