www.palgrave-journals.com/jors

# Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation

R Bai*, EK Burke and G Kendall
*University of Nottingham, Nottingham, UK*

The allocation of fresh produce to shelf space represents a new decision support research area which is motivated by the desire of many retailers to improve their service due to the increasing demand for fresh food. However, automated decision making for fresh produce allocation is challenging because of the very short lifetime of fresh products. This paper considers a recently proposed practical model for the problem which is motivated by our collaboration with Tesco. Moreover, the paper investigates heuristic and meta-heuristic approaches as alternatives for the generalized reduced gradient algorithm, which becomes inefficient when the problem size becomes larger. A simpler single-item inventory problem is firstly studied and solved by a polynomial time bounded procedure. Several dynamic greedy heuristics are then developed for the multi-item problem based on the procedure for the single-item inventory problem. Experimental results show that these greedy heuristics are much more efficient and provide competitive results when compared to those of a multi-start generalized reduced gradient algorithm. In order to further improve the solution, we investigated simulated annealing, a greedy randomized adaptive search procedure and three types of hyper-heuristics. Their performance is tested and compared on a set of problem instances which are made publicly available for the research community.

*Journal of the Operational Research Society* advance online publication, 8 August 2007
doi:10.1057/palgrave.jors.2602463

**Keywords:** meta-heuristics; hyper-heuristics; shelf space allocation; inventory; fresh produce

## Introduction

Increasing health concerns have made people more aware of the importance of healthier foods in their diets (such as fresh vegetables, fruits and organic food) rather than processed produce. This has led retailers to provide a much wider choice in this category of goods. In this paper, we are concerned with inventory control and shelf management for fresh produce such as vegetables, fruits and fresh meats. Fresh produce is different from other produce in that it usually has a very short shelf-life and its utility or condition of freshness continuously decays throughout its lifetime. The fresh produce inventory problem belongs to a wider domain of deteriorating inventory problems which have been intensively studied, with a large number of models being proposed in the literature. Comprehensive reviews on deteriorating inventory can be found in Nahmias (1982), Raafat (1991), Goyal and Giri (2001). However, most of these models treated fresh produce as a special case of perishable items with a fixed deterioration rate and non-decaying utilities before their expiration dates. Based on this principle, different ages of items capture the same demand, however fresh they are, as long as they are not completely spoilt. This is unreasonable as freshness is one of the main criteria for measuring a product's quality and could dramatically impact its demand if its condition is poor. In fact, many retailers have adopted strict temperature control and intelligent inventory and shelf management systems in order to improve their financial performance. In developing countries, it has been observed that retailers usually separate fresh items and items which are less fresh and sell them in different shops at different prices (Kar *et al*, 2001). In addition, some inventory models (Mandal and Phaujdar, 1989; Giri *et al*, 1996) assume that all stock could be displayed on the shelves. This situation, however, seldom happens in most supermarkets because the shelf space for fresh food is normally limited, and expensive, due to the low temperature requirements. Therefore, only a part of the inventory can be displayed on the shelves. Furthermore, when we consider a range of goods rather than single items, the shelf space allocation among different items is especially important. The importance of shelf space allocation for non-perishable merchandise is underlined in several previous studies (Kotzan and Evanson, 1969; Curhan, 1972; Borin *et al*, 1994; Urban, 1998; Yang and Chen, 1999, Bai and Kendall, 2005).

This research is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) and a leading UK retailer (Tesco). Colleagues at Tesco presented us with

*Correspondence: R Bai, Automated Scheduling, Optimisation and Planning (ASAP) Research Group, School of Computer Science & IT, University of Nottingham, Nottingham NG8 1BB, UK.*
E-mail: rzb@cs.nott.ac.uk

the problem addressed in this paper which is concerned with the shelf space allocation for fresh food. This is an area in which they have a particular interest due to increasing market competition. The problem is difficult because of the situation discussed above concerning the deterioration of fresh food and the difficulty in managing inventory and shelf space allocation for such short-life products. What they are seeking is a new approach that can be quickly and easily implemented to improve their current methodologies. Such an approach is presented here.

In this paper, we consider a practical fresh produce inventory control and shelf space allocation model that was built upon the situation as described by Tesco and which was proposed in an earlier paper (Bai and Kendall, 2007). The model can simultaneously decide the ordering policy as well as allocating shelf space among different items, together with the ability to consider utility deterioration. The goal of this research is to help retailers to move towards a high-quality automated ordering and shelf allocation decision system for the retail of fresh produce where the aim is to maximize the overall profit achieved during a given time period. In a similar manner to the way in which other inventory problems were tackled (Urban, 1998; Kar *et al*, 2001), a generalized reduced gradient algorithm (GRG) is used to solve the problem in (Bai and Kendall, 2007). Although good results were reported, GRG may not be the most efficient approach for this problem. Firstly, GRG algorithms cannot handle integer variables which are necessary in inventory control and shelf space allocation problems. As some rounding heuristics have to be used to convert the results obtained by GRG to integer values, the resulting solution can only be a local optimum in most cases and the quality of the solution may be poor. Secondly, as mentioned in Bai and Kendall (2007), a general GRG algorithm can only find the local optimum in the vicinity of the initial solution from where the GRG algorithm started. Although a multi-start version of GRG can eliminate this problem, it can be computationally expensive for large problem instances because each call to GRG is generally slow. As an extension of non-linear knapsack problems, this paper investigates heuristic and meta-heuristic approaches for the problem.

## Problem model

The problem we address in this paper can be described as follows. Suppose a retailer is selling $n$ types of fresh items (or *stock-keeping-units*). Each item has three decision variables $q_i$, $s_i$ and $r_i$, representing the procurement quantity, the number of allocated shelf facings and the surplus of item $i$, respectively. The inventory level of item $i$ changes periodically over time according to the curve shown in Figure 1. At the beginning of the period (corresponding to time 0), the retailer places an order of quantity $q_i$ for item $i$ (assuming a zero lead time). Initially $s_i$ facings of these items are displayed on the shelf and $(q_i - s_i)$ quantities of item $i$ are kept in the storage room. As sales are made, the items in the storage room are

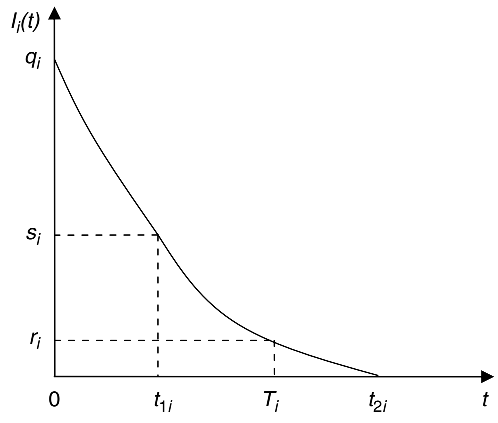

**Figure 1** Graphical representation of inventory level changes over time. *Source*: (Bai and Kendall, 2007).

moved to the shelf to fill the vacated space. From time $t_{1i}$, stock in the backroom reaches zero and the shelf is only partially stocked. Once the time reaches point $T_i$, the surplus of item $i$, represented by $r_i$, is removed from the shelf and is assumed to be sold at a discount price, $pd_i$, immediately. A new order of quantity $q_i$ is placed for item $i$ (time $t_{2i}$ corresponds to the point when the inventory becomes zero if no discounts are made). The next period starts and the inventory changes in the same way as it did in the last period. Note that every new order arrives at point $T_i$ although it might be possible that $T_i = t_{2i}$ (when $r_i = 0$).

Bai and Kendall (2007) proposed a practical non-linear model for the problem which considered both the effect of allocated shelf space and an item's freshness on the demand. The objective is to maximize the overall profit per unit time for all items. In this paper, we use the same model which is also briefly described here for completeness:

$$\text{maximize} \quad \sum_{i=1}^{n} M_i(s_i, q_i, r_i) \tag{1}$$

subject to

$$\sum_{i=1}^{n} s_i A_i \leqslant W \tag{2}$$

$$L_i \leqslant s_i \leqslant U_i, \qquad i = 1, 2, \ldots, n \tag{3}$$

$$r_i \leqslant s_i \leqslant q_i, \qquad i = 1, 2, \ldots, n \tag{4}$$

$$r_i < q_i, \qquad i = 1, 2, \ldots, n \tag{5}$$

$$0 < T_i \leqslant E_i, \qquad i = 1, 2, \ldots, n \tag{6}$$

$$s_i, q_i \in \{1, 2, 3, \ldots\}, \qquad i = 1, 2, \ldots, n \tag{7}$$

$$r_i \in \{0, 1, 2, \ldots\}, \qquad i = 1, 2, \ldots, n \tag{8}$$

where $M_i(s_i, q_i, r_i)$ (denoted by $M_i$ in short) is the average profit of item $i$ per unit time, given by (total income deducing total cost):

$$M_i = \frac{1}{T_i}[p_i(q_i - r_i) + pd_i r_i - ca_i q_i - Co_i - HC_{1i} - HC_{2i}] - c_s s_i A_i \qquad (9)$$

The following is a list of the notations used in the paper:

- $q_i$ is the procurement quantity of item $i$
- $s_i$ is the number of the facings (displayed item quantity) assigned to item $i$
- $r_i$ is the surplus of item $i$ at the end of the cycle
- $p_i$ is the unit selling price of item $i$
- $pd_i$ is the unit discount price of item $i$
- $ca_i$ is the unit acquisition cost of item $i$
- $Co_i$ is the constant order cost of item $i$ (independent of the order quantity $q_i$)
- $ch_i$ is the unit holding cost of item $i$, including the costs caused by inventory loses, damage, maintenance, interest, insurance, etc
- $I_i(t)$ is the inventory level of item $i$ at time $t$. It was defined in Bai and Kendall (2007) and is reproduced here for completeness.

$$I_i(t) = \begin{cases} q_i + \dfrac{\alpha_i s_i^{\beta_i}}{\sigma_i}(\mathrm{e}^{-\sigma_i t} - 1) & 0 \leqslant t \leqslant t_{1i} \\ \left[\dfrac{\alpha_i(1-\beta_i)}{\sigma_i}\mathrm{e}^{-\sigma_i t} + K_i\right]^{\frac{1}{(1-\beta_i)}} & t_{1i} < t \leqslant t_{2i} \end{cases}$$

where

$$K_i = [q_i - \beta_i(q_i - s_i)]s_i^{-\beta_i} - \frac{\alpha_i(1-\beta_i)}{\sigma_i}.$$

- $HC_{1i}$ is the total holding cost during period $[0,\ t_{1i}]$. Note that $HC_{1i} = ch_i \int_0^{t_{1i}} I_i(t)\,\mathrm{d}t = ch_i[(q_i - (\alpha_i s_i^{\beta_i}/\sigma_i)t_{1i} + (1 - \mathrm{e}^{-\sigma_i t_{1i}})\alpha_i s_i^{\beta_i}/\sigma_i^2]$.
- $c_s$ is the shelf cost per unit space.
- $\alpha_i$ is scale coefficient in the demand function of item $i$ and $\alpha_i > 0$
- $\beta_i$ is the space elasticity of item $i$ and $0 < \beta_i < 1$
- $\sigma_i$ is the constant decaying rate of item $i$ and $\sigma_i > 0$
- $HC_{2i}$ is the total holding cost during $[t_{1i}, T_i]$ and $HC_{2i} = ch_i \int_{t_{1i}}^{T_i} I_i(t)\,\mathrm{d}t \approx ch_i[s_i + r_i](T_i - t_{1i})/2$
- $A_i$ is the space required for one facing of item $i$
- $E_i$ is the lifetime of item $i$ after which the item is rotten (ie cannot be sold)
- $W$ is the total shelf space available
- $L_i$ is the lower bound of the number of facings of item $i$
- $U_i$ is the upper bound of the number of facings of item $i$
- $T_i$ is the length of the cycle period of item $i$

Constraint (2) makes sure that the total allocated shelf space to each item is no more than the total available shelf space. Constraint (3) ensures that the space allocated to each item is within an upper bound and a lower bound. Constraints (4) and (5) make sure that the order quantity of each item must be greater than the number of facings which itself should be greater than the surplus. Constraint (6) ensures that the span of one cycle period must be less than the product validity period. Constraints (7) and (8) ensure that the number of facings, order quantity and the value of the surplus must be integers.

For a problem with $n$ stock-keeping-units, the total number of variables ($q_i$, $s_i$ and $r_i$) is $3 \times n$. According to Bai and Kendall (2007), all variables have upper and lower bounds, which are $0 < r_i \leqslant s_i$, $L_i < s_i \leqslant U_i$ and $s_i \leqslant q_i \leqslant q_i^{\mathrm{ub}}$, where

$$q_i^{\mathrm{ub}} = \left\lfloor \frac{1}{(1-\beta_i)} r_i^{(1-\beta_i)} s_i^{\beta_i} + \frac{\alpha_i}{\sigma_i} s_i^{\beta_i} - \frac{\beta_i}{(1-\beta_i)} s_i - \frac{\alpha_i}{\sigma_i}\mathrm{e}^{-\sigma_i E_i} s_i^{\beta_i} \right\rfloor \qquad (10)$$

Following the standard notation, $\lfloor x \rfloor$ denotes the function which rounds down a real value $x$ to its closest integer. See Bai and Kendall (2007) for the derivation of equation (10).

## Addressing the single-item inventory problem

Let us first consider a single-item problem: for a given shelf space decision $s(L < s \leqslant U)$, the problem is to search for a pair of order quantity and the amount of surplus ($q$ and $r$) such that the unit space profit function (9) is maximized, subject to the constraints (4–8).

Figures 2 and 3 illustrate a typical relationship between the profit function (9) and the decision variables $q$, $s$ and $r$. It can be seen that, for this item, the profit function is more sensitive to the changes of facings $s$ than the order quantity $q$ and the surplus quantity $r$. This suggests that retailers should take special care when deciding on displayed facings. A bad decision could result in a large profit loss.

As stated above, all decision variables have lower and upper bounds. A simple way to solve the single-item inventory problem is to consider all combinations that satisfy the constraints (4–8). Suppose the shelf space allocated to an item is $s'$, the time complexity of the algorithm is $O(s' q^{\mathrm{ub}})$. However, from Figures 2 and 3, it seems that the profit function (9) displays unimodality over integer variables $q$, $s$ and $r$ and there is only one maximal value. Although, of course, this evidence does not provide a proof, all of our experiments have demonstrated this property. Note that the search space defined over continuous variables is not unimodal since GRG does not converge to the same continuous solution when starting from different initial solution (Bai and Kendall, 2007). Alternatively, a more efficient binary search is used in this paper to obtain a good order quantity $q$ value. Note that this binary search procedure cannot guarantee an optimal $q$ value if the above assumption does not hold, in which case, one may use complete enumeration. Meanwhile because $r$ is relatively small (in most UK supermarkets, the number of facings of an item $s$ is generally less than 12 and $r < s$), an enumeration method can be used in the search for the optimal value of $r$.

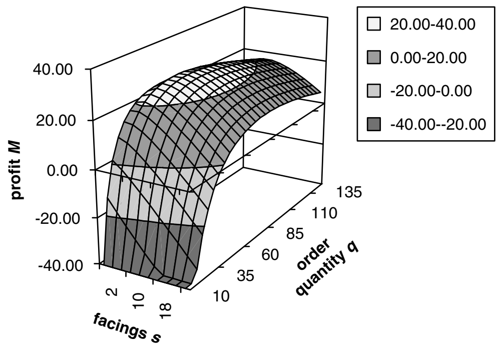

**Figure 2** Graphical representation of an item's profit function with respect to facings $s$ and order quantity $q$ (surplus $r = 0$).

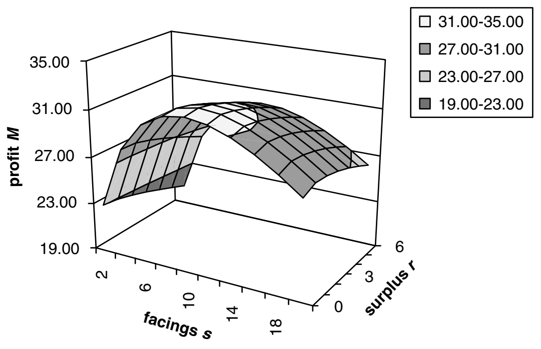

**Figure 3** Graphical representation of an item's profit function with respect to facings $s$ and surplus $r$ (order quantity $q = 90$).

Figure 4 presents the pseudocode for the binary search algorithm which, for simplicity, is denoted by proc_qr($s'$). Suppose the shelf space allocated to an item is $s'$, for each possible value of $r$, a lower bound and an upper bound of $q$ were calculated (denoted by $q^l$ and $q^r$ respectively). The algorithm then divides the range $[q^l, q^r]$ into two equal parts (ie $q=(q^l+q^r)/2$) and checks in which half the optimal order quantity $q'_{\mathrm{opt}}$ lies. If $q'_{\mathrm{opt}}$ lies in the left half, it sets $q^r = q$, otherwise it sets $q^l = q$. This process is repeated until the length of the range $[q^l, q^r]$ decreases to 1 and the optimal order quantity $q'_{\mathrm{opt}}$ is one of the range boundaries (ie $q^l$ or $q^r$). The total number of iterations of this procedure is no more than $s' \log_2 q^{\mathrm{ub}}$ where $q^{\mathrm{ub}}$ is the upper bound of order quantity. Because it is difficult to calculate the derivative of function (9), we used the method below to determine on which side the optimal order quantity $q'_{\mathrm{opt}}$ lies. Let $M$ (respectively $M^l$, $M^r$) be the profit when order quantity is $q$ (respectively $q^l$, $q^r$) and $M^{-\varepsilon}$ be the profit when we decrease $q$ by a very small value $\varepsilon$ (see Figures 5 and 6). If $M^{-\varepsilon} > M^l$, $q'_{\mathrm{opt}}$is on the left hand side of $q$, otherwise, $q'_{\mathrm{opt}}$ is on the right-hand side of $q$.

In order to clearly demonstrate the solution procedure proc_qr($s'$), we present an illustrative example. The problem parameters are taken from the numerical instance BORIN94/6 (item 1) in Bai and Kendall (2007) where

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

**Figure 4** The pseudocode of the procedure proc_qr($s'$).

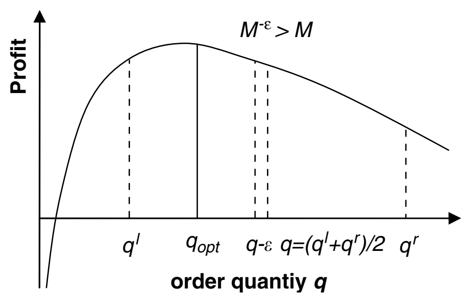

**Figure 5** The relationship between order quantity and its unit time profit function ($q > q_{\mathrm{opt}}$).

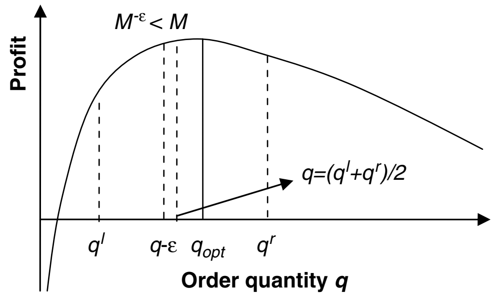

**Figure 6** The relationship between order quantity and its unit time profit function ($q < q_{\mathrm{opt}}$).

$a_i = 0.028$, $p_i = 5.03$, $c_{ai} = 2.46$, $c_{hi} = 0.19$, $p_{di} = 1.23$, $C_o = 34.3$, $\alpha_i = 28.53$, $\beta_i = 0.1532$, $\sigma_i = 0.06$, $c_s = 5.0$, $L_i = 1$, $U_i = 12$, $T_{ei} = 7$. For an input $s = 2$, according to Figure 6,

**Table 1** A detailed illustrations of the iterations used by proc_qr(s′) for the illustrative example

| *Iteration* | $q^l$ | $q^r$ | $M^l$ | $M^r$ | $q=\lfloor (q^l+q^r)/2 \rfloor$ | $M$ | $M^{-\varepsilon}$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 180 | −391.298 | 45.440 | 91 | 54.509 | 54.513 |
| 2 | 2 | 91 | −391.298 | 54.509 | 46 | 50.322 | 50.287 |
| 3 | 46 | 91 | 50.322 | 54.509 | 68 | 54.319 | 54.312 |
| 4 | 68 | 91 | 54.319 | 54.509 | 79 | 54.706 | 54.705 |
| 5 | 79 | 91 | 54.706 | 54.509 | 85 | 54.672 | 54.674 |
| 6 | 79 | 85 | 54.706 | 54.672 | 82 | 54.707 | 54.708 |
| 7 | 79 | 82 | 54.706 | 54.707 | 80 | 54.7109 | 54.7106 |
| 8 | 80 | 82 | 54.7109 | 54.707 | **81** | **54.7111** | |

we have:

$$r=0, q^l=2, q^r=q^{\text{ub}}=\left\lfloor \frac{0^{(1-0.1532)}\times 2^{0.1532}}{(1-0.1532)}+\frac{28.53}{0.06}\times 2^{0.1532}-\frac{0.1532}{(1-0.1532)}\times 2-\frac{28.53}{0.06}\times \mathrm{e}^{-0.06\times 7}2^{0.1532}\right\rfloor = 180, \varepsilon=0.1$$

After 8 iterations (see Table 1) we have $q'_{\text{opt}}=q_r=81$, $M'_{\text{opt}}=54.711$. The same procedures were carried out for $r=1$ and $r=2$ and the final solution for $s=2$ is: $q_{\text{opt}}=81$, $r_{\text{opt}}=0$ $M'_{\text{opt}}=54.711$. The optimal results are shown in bold in Table 1.

## Addressing multi-item problems

In the previous section, we developed a sub-procedure proc_qr($s'$) to obtain a high quality solution for a single-item inventory problem, with constant shelf space $s'$ being allocated to the item. For the multi-item problem model (1) subject to constraints (2)–(9), we can determine the following property.

**Property** For a given shelf space allocation decision $s_i^*$ ($1 \leqslant i \leqslant n$) that satisfies constraints (2) and (3), the maximum of the overall average profit of all the items is equal to the sum of the maximal average profit of each item, that is $\max \sum_{i=1}^n M_i(s_i^*, q_i, r_i) = \sum_{i=1}^n [\max M_i(s_i^*, q_i, r_i)]$.

**Proof** From function (9) it can be seen that, for a given shelf space allocation decision $s_i^*$, the average profit ($M_i$) of item $i$ is only dependent on the corresponding values of $q_i$ and $r_i$ and is not affected by the changes of other decision variables. Therefore, the maximal overall average profit can be obtained by maximizing the average profit of each item.

Although this property appears straightforward, it is very helpful to reduce the search space of the problem model. The property implies that once the shelf space allocation decisions for each item are made, the problem can be solved by independently searching for a pair consisting of order quantity ($q_i$) and surplus ($r_i$) for every item $i$ ($1 \leqslant i \leqslant n$). Therefore, the problem can be decomposed into two sub-problems: (a) optimize $n$ shelf space allocation variables ($s_i$), (b) search for the optimal values of ordering quantity ($q_i$) and surplus ($r_i$), for the given space allocation decisions made in the first sub-problem. Because the second sub-problem can be efficiently solved by the sub-procedure proc_$qr(s')$ with polynomial computational time, the search space is cut down to searching for $n$ shelf space allocation decisions, compared with a search space of $n \times 3$ dimensional vectors without decomposition. Once the optimal shelf space allocation variables are found, the corresponding optimal order quality and surplus can be decided efficiently. In this sense, the problem can be reduced to a nonlinear bounded knapsack problem, which is still NP-Hard (Bretthauer and Shetty, 2002). In the following sections, some heuristic and meta-heuristic approaches will be adapted to the problem and their computational performance will be reported and compared against each other.

### *Greedy heuristics for the problem*

In this section, and the next, we shall consider the original problem (model (1) subject to constraints (2–9)) where there are multiple items in the inventory with limited shelf space on which to display them. The items have to compete against each other for the shelf space so that the total profit is maximized. Once the amount of shelf space allocated to each item is determined, the procedure proc_qr($s'$) can be applied to every item to find the corresponding order quantity and the number of surplus items. There are many ways to allocate shelf space among items. A common sense rule to accomplish this would be to allocate the space in favour of more profitable items. The problem, in fact, degenerates into a problem similar to a nonlinear bounded knapsack problem. Note that this particular knapsack problem is different from the linear ones discussed in Martello and Toth (1990). In linear knapsack problems, the profits of the items are constants and, therefore, each item's unit-space profit (ie profit/space) is constant as well. However, the space allocation problem in this paper is more difficult because the unit-space profit of every item is changing with the change of allocated shelf space. This section introduces four greedy heuristics for this problem.

The basic idea behind the algorithms is that, for a given amount of shelf space, each item represents an intelligent entity optimizing its own inventory variables ($q$ and $r$) using the procedure proc_qr($s'$). However, with the limited shelf space resources, these items have to compete and cooperate with each other so that their total profit is maximized. Items that make less profit per unit of shelf space must release part of their space to those which could make more profit if given the same amount of shelf space.

Two functions were used to rank the profitability of different items with respect to the shelf space (denoted by $F_1$ and $F_2$ respectively). The first function is an item's unit space profitability, defined by $F_1 = M_i(s_i)/(a_i s_i)$. The second function is a differential coefficient profitability, defined by $F_2 = (M_i(s_i) - M_i(s_i - \varepsilon))/(\varepsilon a_i)$ where $\varepsilon$ is a small positive value (the derivative value is an ideal criterion but is difficult to calculate in this case). Note that $M_i(s_i)$ is the optimal unit profit value of item $i$ returned by the procedure proc_qr($s'$) for the given input $s_i$. Because the profit function (9) is nonlinear with respect to the facings $s_i$, both $F_1$ and $F_2$ are not constant and will change with the changes of $s_i$. Therefore, both profitability values $F_1$ and $F_2$ need to be recalculated at each solution construction step. This is the dynamic part of the algorithm.

There are two possible points from which the greedy heuristics can start. They can start from a solution that has met the minimal space requirements and can then repeatedly add a facing to the shelf according to the ranking functions $F_1$ or $F_2$ without violating the constraint (2). They can also start from a point where the number of facings of each item is equal to its upper bound and then repeatedly delete a facing according to the functions $F_1$ or $F_2$ until the space constraint (2) is satisfied. Therefore, there are a total of four combinations, denoted by GH$_1$, GH$_2$, GH$_3$ and GH$_4$ respectively and described as follows.

GH$_1$ (Greedy_Fwd): This heuristic starts from a shelf space allocation decision that satisfies the minimal space requirements of each item and repeatedly adds to the shelf a facing of the item with the largest profitability value according to the criterion $F_1$. The heuristic stops as soon as no more facings can be added to the shelf. During this process, if adding a facing causes a constraint violation, the profitability value of this item is set to a very small value so that the item is of no further consideration. A full description of the algorithm is given in Figure 7.

GH$_2$ (Greedy_Bwd): This heuristic starts from an initial shelf space allocation that is equal to the corresponding upper bounds. Then the heuristic repeatedly deletes a facing of the item with the smallest profitability value of $F_1$ until the shelf space constraint is satisfied. Afterwards, a sub-procedure is executed which tries to add (if possible) as many facings as possible to the shelf according to the criterion of $F_1$ (see Figure 8 for a detailed description).

GH$_3$ (Greedy_Derivative_Fwd): This heuristic is the same as GH$_1$ except that the greedy criterion is $F_2$ instead of $F_1$.

GH$_4$ (Greedy_Derivative_Bwd): This heuristic is the same as GH$_2$ except it uses $F_2$ as the greedy criterion.

```
Step 1:
  For each item $i$ ($1 \le i \le n$)
    $s_i = L_i$ ;
    Call proc_qr($s_i$) to optimize $q_i$ and $r_i$;
    Calculate $F_1$ value for item $i$;
  Endfor
Step 2:
  If (FreeSpace > MinProdSpace)
    Select an item $j$ with largest possible
    profitability value of $F_1$ and whose size is
    smaller than free space and the number of
    facing $s_j$ is less than its upper bound;
    If no such item is available, stop the procedure
    Else
      $s_j = s_j + 1$;
      Call proc_qr($s_j$) to optimize $q_j$ and $r_j$;
      Update $F_1$ for item $j$;
      Go to step 2;
    Endif
  Else
  Stop and output the solution.
Endif
```

**Figure 7** Pseudocode of GH$_1$.

```
Step 1:
  For each item $i$ ($1 \le i \le n$)
    $s_i = U_i$ ;
    Call proc_qr($s_i$) to optimize $q_i$ and $r_i$;
    Calculate $F_1$ value for item $i$;
  Endfor

Step 2:
  While (SpaceUsed > SpaceAvailable)
    Select an item $j$ with the smallest possible
    profitability value of $F_1$ and whose facing ($s_j$)
    has not reached its lower bound;
    $s_j = s_j - 1$;
    Call proc_qr($s_j$) to optimize $q_j$ and $r_j$;
    Update $F_1$ for item $j$ ;
  Loop

Step 3:
  If (FreeSpace > MinProdSpace)
    Select an item $k$ with the largest possible
    profitability value of $F_1$ and whose area is
    smaller than free space and where the number of
    facing $s_k$ is less than its upper bound;
    If no such item is available, stop the procedure
    Else
      $s_k = s_k + 1$;
      Call proc_qr($s_k$) to optimize $q_k$ and $r_k$
      Update $F_1$ for item $k$;
      Go to step 3;
    Endif
  Else
    Stop and output solution.
  Endif
```

**Figure 8** Pseudocode of GH$_2$.

Table 2 gives a comparison of the four greedy heuristics and the multi-start GRG algorithm, on the five test problem instances reported in Bai and Kendall (2007) (with the

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

| | *BORIN94/6* | | *FRESH2* | | *FRESH3* | | *FRESH4* | | *FRESH5* | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | 6 | | 18 | | 32 | | 49 | | 64 | |
| | *obj.* | *cpu(s)* | *obj.* | *cpu(s)* | *obj.* | *cpu(s)* | *obj.* | *cpu(s)* | *obj.* | *cpu (s)* |
| Multi-start GRG | **347.45** | 3.2 | 1129.6 | 73.6 | **2056.46** | 74.3 | **3163.98** | 179.2 | **4387.16** | 209.7 |
| $GH_1$ | 344.55 | 0.03 | 1126.8 | 0.03 | 2042.07 | 0.05 | 3144.02 | 0.05 | 4360.44 | 0.09 |
| $GH_2$ | 344.55 | 0.05 | 1129.09 | 0.13 | 2041.59 | 0.28 | 3147.28 | 0.55 | 4358.96 | 0.91 |
| $GH_3$ | **347.45** | 0.02 | **1131.64** | 0.02 | 2053.71 | 0.03 | 3159.17 | 0.06 | 4384.62 | 0.09 |
| $GH_4$ | 346.90 | 0.06 | 1131.33 | 0.25 | 2054.14 | 0.50 | 3160.91 | 1.06 | 4382.66 | 1.45 |
| GRG relaxed + $GH_3^*$ | 349.05 | 0.5 | 1137.73 | 5.7 | 2063.09 | 18.3 | 3172.96 | 60.0 | 4394.96 | 47.5 |

The bold values represent the best results.
Note that **n** is the number of items; **obj.** is the objective value of the solution obtained by different algorithms (for multi-start GRG, this is the average value of 10 runs); **cpu(s)** is average CPU time consumed by different algorithms (in seconds).
* The objective values of the relaxed problem instances (excluding integrality constraints) by GRG with an initial solution generated by $GH_3$.

best results shown in bold). The relaxed problem instances, in which the integrality constraints of the decision variables were excluded, were also solved by the GRG with initial solutions generated by $GH_3$. The results for these relaxed problem instances are also presented in Table 2. However, once the rounding heuristic is applied to recover the feasibility, we obtained the same quality solutions as those by the multi-start GRG. It can be seen that all greedy heuristics are very fast, compared with the multi-start GRG algorithm. $GH_1$ and $GH_3$ are also faster than $GH_2$ and $GH_4$. This is probably because the facings in the final solution are closer to their lower bound facings than to the upper bound facings for these instances. In terms of the solution quality, $GH_3$ and $GH_4$ perform better than $GH_1$ and $GH_2$ and are even competitive when compared with the multi-start GRG algorithm, which took much longer. Neither $GH_3$ nor $GH_4$ performed better than the other in terms of solution quality. $GH_3$ is better on the instance BORIN94/6, FRESH2 and FRESH5 while $GH_4$ is better than $GH_3$ on the other two instances. However, $GH_3$was computationally less expensive than $GH_4$. For real-world applications that not only stress solution quality but also emphasize algorithm simplicity and speed, heuristic $GH_3$ has obvious advantages over the multi-start GRG.

### *Further improvement*

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

## A GRASP algorithm for the problem

A GRASP (greedy randomized adaptive search procedure) algorithm (Feo and Resende, 1995) has been applied to the problem. GRASP is a multi-start meta-heuristic approach that explores the search space from different starting points. The idea of applying GRASP to this problem is that we have two profitability functions, $F_1$ and $F_2$, available for this problem. The greedy heuristics based on the function $F_2$ produce high-quality solutions. This function can be utilized in the solution construction stage of a GRASP algorithm.

```
For nrep = 1 to max_rep
    /* solution construction phase */
    Start from an empty solution;
    For each item $i$ ( $1 \le i \le n$ )
        $s_i = L_i$ ;
        Call proc_qr($s_i$) to optimize $q_i$ and $r_i$;
        Calculate $F_2$ value for item $i$;
    Endfor
    Initialise candidate list (CL);
    While (FreeSpace > MinProdSpace and $CL \ne \emptyset$ )
        $F_2^{\min} \leftarrow \min\{F_2(i) \mid i \in CL\}$ ;
        $F_2^{\max} \leftarrow \min\{F_2(i) \mid i \in CL\}$ ;
        Construct Restricted Candidate List (RCL) by
        $RCL \leftarrow \{\text{item } i \mid i \in CL \ \&$
            $F_2(i) \ge F_2^{\min} + \theta(F_2^{\max} - F_2^{\min})\}$ ;
        Select an item $j$ from RCL at random;
        $s_j = s_j + 1$;
        Call proc_qr($s_j$) to optimize $q_j$ and $r_j$ ;
        Update the candidate list CL;
        Update $F_2$ value for item $j$;
    Loop

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

**Figure 9** A GRASP algorithm for the problem.

Figure 9 presents the pseudocode of the GRASP algorithm used in this paper. A total of *max_rep* runs were executed and each run consists of a solution construction phase and a local search phase. The solution construction phase is very similar to the greedy algorithm $GH_3$ except that a parameter $\theta$ is introduced to control the degree of randomness and greediness. To do this, a candidate list (CL) is maintained, consisting of all the non-initialized variables $s_i$. A restricted candidate list (RCL) is constructed based on the parameter $\theta$ (see Figure 9). The case $\theta = 0$ corresponds to a random construction process, while $\theta = 1$ is equivalent to the greedy algorithm GH$_3$. The local search phase is a simple hill-climbing algorithm which repeatedly generates a candidate solution by swapping one facing of two random items (ie select two random items $i$ and $j$, set $s_i = s_i - 1$, $s_j = s_j + 1$) and then calling the procedure proc_qr($s'$) to obtain corresponding $q_i$, $r_i$, $q_j$ and $r_j$ after swapping. The candidate solution is accepted if it is better and discarded otherwise. The local search phase stops when the number of total repetitions exceeds a given value *ls_max_rep*. After the preliminary experiments, we set $\theta = 0.85$, $max\_rep = 100$ and $ls\_max\_rep = n^2$.

## A simulated annealing algorithm for the problem

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

## Hyper-heuristic approaches for the problem

Hyper-heuristics (Burke *et al*, 2003a,b; Ross, 2005) represent a search methodology that is receiving some attention in the literature. The term hyper-heuristic has been defined as the procedure of 'using (meta-) heuristics to choose (meta-)heuristics to solve the problem in hand' (Burke *et al*, 2003b). One of the ideas behind hyper-heuristics is that each problem-specific heuristic may have some weaknesses in certain scenarios in which other heuristics may perform better. Better algorithmic performance could be achieved by combining a set of heuristics, instead of using just a single heuristic alone. Hyper-heuristics combine a set of easily implemented, heuristics in a strategic way so that the algorithm is able to tackle not only a specific problem or problem instance but a batch of problems. Unlike most search methodologies (and, indeed, most implementations of meta-heuristics) which search the solution space directly, hyper-heuristics work on the problem indirectly by strategically selecting appropriate heuristics at different times in the search. With hyper-heuristics, the primary domain of exploration is a search space of heuristics rather than a search space of solutions. In recent years, hyper-heuristics have been successfully explored across many scheduling and combinatorial problems (Hart *et al*, 1998; Terashima-Marin *et al*, 1999; Ross *et al*, 2002; Burke *et al*, 2003b; Nareyek, 2003; Ross *et al*, 2003; Bai and Kendall, 2005, Kendall and Mohd Hussin (2005); Rattadilok *et al*, 2005; Bilgin *et al*, 2006; Burke *et al*, 2006; Ozcan *et al*, 2006; Burke *et al*, 2007; Dowsland *et al*, 2007). One can refer to (Burke *et al*, 2003a; Ross, 2005) for more discussion and applications of hyper-heuristics.

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

**Figure 10** The pseudocode of a tabu search based hyper-heuristic for a maximization problem *Source*: (Burke *et al*, 2003b).

### *Tabu search hyper-heuristics*

We firstly applied a recently developed tabu search-based hyper-heuristic approach (TSHH) (Burke *et al*, 2003b) to the problem. The main idea behind this algorithm is the incorporation of a tabu list in the heuristic selection mechanism that forbids the selection of some low-level heuristics at certain stages of the search. For a maximization problem with an objective function $f(x)$, TSHH can be described by Figure 10.

In this application, the parameters were set as follows (these are the same as Burke *et al*, 2003b). Suppose a total of $k$ low-level heuristics were used, the maximal length of the tabu list was set to $max\_tabu\_len = k/2$. The upper and lower bound

```
Initialisation: initial solution $s_0$, temperature $t_s$, cooling
function $\varphi(t)$ and a set of low-level heuristics
$\{H_i \mid i = 1,...n\}$, for given evaluation function $f$:
Repeat
    Repeat
        Randomly select a low-level heuristic $H^* \in \{H_i \mid i = 1,...n\}$;
        Apply H* to the current solution s, resulting in a
        neighbour solution s';
        $\delta = f(s') - f(s)$;
        If $f(s') - f(s) > 0$  $s \leftarrow s'$;
        Else if ( $\exp((f(s') - f(s))/t) > random(0,1)$ )
            $s \leftarrow s'$;
        Endif
        If $f(s) > f(s_{best})$  $s_{best} \leftarrow s$ ;
    Until iteration_count = nrep
    Set $t = \varphi(t)$;
Until the stopping conditions are met.
Output $s_{best}$ as the best solution found.
```

**Figure 11** A simulated annealing hyper-heuristic algorithm for a maximization problem *Source*: (Bai and Kendall, 2005).

of weights for each low-level heuristic were set to $w_u = k$ and $w_l = 0$ respectively. The initial weights of each heuristic were set to their lower bounds and updated after each heuristic call. Once a heuristic's weight exceeded one of its boundaries, it was set to the corresponding boundary.

*Simulated annealing hyper-heuristics*

A simulated annealing hyper-heuristic (SAHH) proposed in (Bai and Kendall, 2005) is also adapted for the problem. The pseudocode of the algorithm is presented in Figure 11.

*Tabu search simulated annealing hyper-heuristics*

A hybrid hyper-heuristic was also implemented, denoted by TSSAHH, which hybridizes the tabu search hyper-heuristic and the simulated annealing hyper-heuristic described above. In this hybrid algorithm, the low-level heuristics are selected using the same rule as in TSHH but candidate solutions are accepted according to the simulated annealing probability. The parameters are the same as the general simulated annealing algorithm described in the previous section. The pseudocode of TSSAHH is given in Figure 12.

All three types of hyper-heuristics used the same four low-level heuristics, which can be outlined as follows:

- *2-opt*: This heuristic removes one facing from a random item and adds one facing to another item, that is, it selects two random items *i* and *j*, and sets $s_i = s_i + 1$, $s_j = s_j - 1$. We call the procedure proc_qr($s'$) to improve the corresponding order quantity and surplus.
- *3-opt1*: This heuristic randomly selects three different items, *i*, *j*, *k*, and sets $s_i = s_i - 1$, $s_j = s_j - 1$, $s_k = s_k + 1$. We call the procedure proc_qr($s'$) to improve the corresponding order quantity and surplus.
- *3-opt2*: This heuristic randomly selects three different items, *i*, *j*, *k* and sets $s_i = s_i + 1$, $s_j = s_j + 1$, $s_k = s_k - 1$. We call the procedure proc_qr($s'$) to improve the corresponding order quantity and surplus.
- *4-opt*: This heuristic selects four different random items, deletes one facing of two random items and adds one facing to the other two items. We call the procedure proc_qr($s'$) to optimize the corresponding order quantity and surplus.

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

**Figure 12** The pseudocode of a TSSAHH for a maximization problem.

Note that each of the low-level heuristics described above is guaranteed to return a feasible solution. If a low-level heuristic cannot produce a new feasible solution, the original solution is returned. All three hyper-heuristics above started from the same solution generated by $GH_3$. The stopping criterion is a computational time limit which was set to be approximately the same value as the amount of time spent by the multi-start GRG algorithm (see Table 2).

## Experimental results

The above algorithms were applied to the same five problem instances from (Bai and Kendall, 2007), each of which were run 30 times. Their computational results are averaged and presented in Table 3 (with the best results highlighted

**Table 3** A comparison of different algorithms on five fresh produce instances

| | *BORIN94/6* | | | *FRESH2* | | | *FRESH3* | | | *FRESH4* | | | *FRESH5* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 6 | | | 18 | | | 32 | | | 49 | | | 64 | | |
| | *av. obj.* | *stdev* | *av. cpu* | *av. obj.* | *stdev* | *av.cup* | *av. obj.* | *stdev* | *av.cup* | *av. obj.* | *stdev* | *av.cup* | *av. obj.* | *stdev* | *av.cup* |
| Initial (GH3) | 347.45 | – | 0.02 | 1131.64 | – | 0.02 | 2053.71 | – | 0.03 | 3159.17 | – | 0.06 | 4384.62 | – | 0.09 |
| Multi-Start GRG | 347.45 | 0 | 3.2 | 1129.6 | 0 | 73.6 | 2056.46 | 0.97 | 74.3 | 3163.98 | 0.51 | 179.2 | 4387.16 | 0.43 | 209.7 |
| GRASP | **347.58** | 0.00 | 2.79 | **1133.51** | 0.00 | 23.64 | 2056.43 | 0.17 | 78.90 | 3164.14 | 0.34 | 245.30 | 4387.23 | 0.31 | 438.99 |
| SA | **347.58** | 0.00 | 2.23 | 1133.22 | 0.36 | 19.11 | 2055.04 | 0.61 | 72.24 | 3163.81 | 0.41 | 226.70 | 4387.16 | 0.51 | 401.20 |
| SAHH | **347.58** | 0.00 | 3.70 | **1133.51** | 0.00 | 61.26 | 2056.93 | 0.27 | 60.62 | 3164.18 | 0.27 | 148.91 | **4387.42** | 0.37 | 185.09 |
| TSHH | 347.56 | 0.05 | 3.20 | 1131.64 | 0.00 | 73.61 | 2053.71 | 0.00 | 74.31 | 3159.23 | 0.33 | 179.21 | 4384.62 | 0.00 | 209.71 |
| TSSAHH | **347.58** | 0.00 | 3.72 | **1133.51** | 0.00 | 62.50 | **2057.09** | 0.71 | 56.31 | **3164.21** | 0.25 | 135.55 | 4387.41 | 0.32 | 185.29 |

The bold values represent the best results.
Note that av. obj. is the average objective value of 30 runs; stdev is the standard deviation of 30 runs; av. cpu is the average CPU time spent.

in bold).

It can be seen that the results obtained by $GH_3$ are very close to the results of the four meta-heuristic algorithms. The largest improvement for the instance BORIN94/6 is only 0.04% ((347.58 − 347.45/347.45) × 100%). The four algorithms have consistently solved this small instance to optimality over 30 runs (the optimal solution of BORIN94/6 was obtained by a complete search). For the instance FRESH2, three algorithms (GRASP, SAHH, TSSAHH) consistently produced the same solution over 30 runs). For the other four instances, the biggest improvements over the initial solutions are 0.17, 0.16, 0.16 and 0.06% respectively. Similar results were obtained even when the algorithms were given much more computational time or more repetitions. This evidence suggests that these results are of a high level of quality. A theoretical study of how close (or not) they may be to optimality by establishing upper bounds represents a direction for future research. Among the algorithms we have investigated, the GRASP algorithm performed well when compared with the multi-start GRG algorithm and the general simulated annealing. It was only marginally outperformed by multi-GRG on instance FRESH3. However, on larger problem instances, both GRASP and simulated annealing consumed more computational time than the multi-start GRG. Comparing the different hyper-heuristics, TSHH was unable to improve the initial solution or only achieved a very small improvement. However, the performance of TSHH was improved when hybridized with a simulated annealing acceptance criterion (corresponding to the algorithm TSSAHH).

In general, meta-heuristics (especially when employed as hyper-heuristics) are not only more efficient than the multi-start GRG when dealing with larger size problem instances, but more importantly, are much easier to implement than the GRG which involves complex mathematical manipulations. Two types of hyper-heuristic performed best among all the algorithms. TSSAHH outperformed all of the other algorithms for four instances and was only marginally beaten by SAHH on the remaining one instance. SAHH performed well and obtained best results on three instances (BORIN94/6, FRESH2 AND FRESH5). Even for the other two instances, it ranked as the second best algorithm and found solutions that are very close to the best known solutions.

## Conclusions

The motivation of this paper is to investigate fast, easy-to-implement approaches as alternatives to the multi-start GRG algorithm used to address a fresh produce inventory control and shelf space allocation problem in Bai and Kendall (2007). We firstly analysed a single item problem and proposed a binary search algorithm to obtain a high quality solution. For the multi-item problem, the problem can be decomposed into two sub-problems and the search space can be substantially reduced. Four greedy heuristic methods are developed which dynamically rank the objective contribution of each item and allocate more space to those items which could make more profit. They also reclaim shelf space from those items if they cannot make as much profit as another item. The experimental results over five test problem instances showed that these greedy algorithms, especially $GH_3$ could produce similar quality solutions to the multi-start GRG but with much less computational time. In practice, where simplicity and computational time are also priorities, this simple heuristic method has advantages over the widely used GRG algorithm from the literature. The paper also investigated several meta-heuristic approaches in order to further improve the solution generated by the greedy algorithms. Two of the hyper-heuristic methods have been able to obtain better quality solutions than the multi-start GRG does while using similar or less computational time.

*Acknowledgements*—We would like to thank the anonymous referees and the editor for their helpful comments. We are also grateful for the support of the Engineering and Physical Sciences Research Council (EPSRC) under grants GR/R60577/01 and GR/S70197/01.

## References

Bai R and Kendall G (2005). An investigation of automated planograms using a simulated annealing based hyper-heuristic. In: Ibaraki T, Nonobe K and Yagiura M (eds). *Metaheuristics: Progress as Real Problem Solvers*, Operations Research/Computer Science Interfaces Series, Vol. 32. Springer: Berlin, Heidelberg, New York, pp 87–108.

Bai R and Kendall G (2007). A model for fresh produce shelf space allocation and inventory management with freshness condition dependent demand. *INFORMS J Comput*, accepted.

Bilgin B, Ozcan E and Korkmaz EE (2006). An experimental study on hyper-heuristics and exam timetabling. *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, Brono, Czech Republic, August 30–September 1, pp 123–140.

Borin N, Farris PW and Freeland JR (1994). A model for determining retail product category assortment and shelf space allocation. *Decis Sci* **25**: 359–384.

Bretthauer KM and Shetty B (2002). The nonlinear knapsack problem—algorithms and applications. *Eur J Opl Res* **138**: 459–472.

Burke EK, Hart E, Kendall G, Newall J, Ross P and Schulenburg S (2003a). Hyper-heuristics: An emerging direction in modern search technology. In: Glover FW and Kochenberger GA (eds). *Handbook of Meta-Heuristics*. Kluwer: Dordrecht, pp 457–474.

Burke EK, Kendall G and Soubeiga E (2003). A tabu-search hyper-heuristic for timetabling and rostering. *J Heuristics* **9**: 451–470.

Burke EK, McCollum B, Meisels A, Petrovic S and Qu R (2007). A graph-based hyper-heuristic for educational timetabling problems. *Eur J Opl Res* **176**: 177–192.

Burke EK, Petrovic S and Qu R (2006). Case based heuristic selection for timetabling problems. *J Scheduling* **9**(2): 99–113.

Curhan R (1972). The relationship between shelf space and unit sales in supermarkets. *J Market Res* **9**: 406–412.

Dowsland KA, Soubeiga E and Burke EK (2007). A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *Eur J Opl Res* **179**: 759–774.

Feo TA and Resende MGC (1995). Greedy randomized adaptive search procedures. *J Global Optim* **6**: 109–133.

Giri BC, Pal S, Goswami A and Chaudhuri KS (1996). An inventory model for deteriorating items with stock-dependent demand rate. *Eur J Opl Res* **95**: 604–610.

Goyal SK and Giri BC (2001). Recent trends in modelling of deteriorating inventory. *Eur J Opl Res* **134**: 1–16.

Hart E, Ross P and Nelson JA (1998). Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolut Comput* **6**(1): 61–80.

Kar S, Bhunia AK and Maiti M (2001). Inventory of multi-deteriorating items sold from two shops under single management with constraints on space and investment. *Comput Opl Res* **28**: 1203–1221.

Kendall G and Mohd Hussin N (2005). An investigation of a tabu search based hyper-heuristic for examination timetabling. In: Kendall G, Burke EK and Petrovic S (eds). *Multidisciplinary Scheduling: Theory and Application*. Springer: Berlin, pp 309–328.

Kotzan J and Evanson R (1969). Responsiveness of drug store sales to shelf space allocations. *J Market Res* **6**: 465–469.

Lundy M and Mees A (1986). Convergence of an annealing algorithm. *Math Prog* **34**: 111–124.

Mandal BN and Phaujdar S (1989). An inventory model for deteriorating items and stock-dependent consumption rate. *J Opl Res Soc* **40**: 483–488.

Martello S and Toth P (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons: New York.

Nahmias S (1982). Perishable inventory theory: A review. *Opns Res* **30**: 680–708.

Nareyek A (2003). Choosing search heuristics by non-stationary reinforcement learning. In: Resende MGC and de Sousa JP (eds). *Metaheuristics: Comp Decision-Making*. Kluwer: Dordrecht, pp 523–544.

Ozcan E, Bilgin B and Korkmaz EE (2006). Hill climbers and mutational heuristics in hyperheuristics. In: Poli R, Cotta C, Coello CAC, Pelikan M, Ishibuchi H, Sastry K, Whitley D, Zitzler E, Sebag M and Sendhoff B. *Parallel Problem Solving from Nature – PPSN IX: Selected Papers from the 9th International Conference*, Lecture Notes in Computer Science Series, Vol. 4193, Springer: Berlin, pp 202–211.

Raafat F (1991). Survey of literature on continuously deteriorating inventory models. *J Opl Res Soc* **42**: 27–37.

Rattadilok P, Gaw A, and Kwan RSK (2005). Distributed choice function hyper-heuristics for timetabling and scheduling. In: Burke EK and Trick M (eds). *The Practice and Theory of Automated Timetabling V: Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science Series, Vol. 3616, Springer: Berlin, pp 51–70.

Ross P (2005). Hyper-heuristics. In: Burke EK and Kendall G (eds). *Search Methodologies: Introductory Tutorials in Optim and Decision Support Tech*. Springer: Berlin, pp 529–556.

Ross P, Marin-Blazquez JG, Schulenburg S and Hart E (2003). Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics. *Proceedings of the Genetic and Evolutionary Comput Conference*, *GECCO 2003*, Chicago, Illinois, USA, pp 1295–1306.

Ross P, Schulenburg S, Marin-Blazquez JG and Hart E (2002). Hyper heuristics: Learning to combine simple heuristics in bin-packing problems. *Proceedings of the Genetic and Evolutionary Comput Conference* (GECCO2002), New York, US, 9–13 July, pp 942–948.

Terashima-Marin H, Ross P and Valenzuela-Rendon M (1999). Evolution of constraint satisfaction strategies in examination timetabling. *Proceedings of the Genetic and Evolutionary Comput Conference* (GECCO 1999), Orlando, Florida, USA, 13–17 July, pp 1011–1018.

Urban T (1998). An inventory-theoretic approach to product assortment and shelf-space allocation. *J Retailing* **74**: 15–35.

Yang M-H and Chen W-C (1999). A study on shelf space allocation and management. *Int J Prod Econ* **60**(61): 309–317.

*Received December 2005;*
*accepted May 2007 after three revisions*