The PrePack Optimization Problem

Maxim Hoskins[1,2], Renaud Masson[1], Gabrielle Gauthier Melançon[1], Jorge E. Mendoza[2], Christophe Meyer[3], and Louis-Martin Rousseau[1]

[1]CIRRELT, École Polytechnique de Montréal, Montreal, Canada
[2]Université Catholique de l'Ouest, LARIS (EA 7315), Angers, France
[3]Université du Québec à Montréal, Montreal, Canada

Abstract. The goal of packing optimization is to provide a foundation for decisions related to inventory allocation as merchandise is brought to warehouses and then dispatched. Major retail chains must fulfill requests from hundreds of stores by dispatching items stored in their warehouses. The demand for clothing items may vary to a considerable extent from one store to the next. To take this into account, the warehouse must pack "boxes" containing different mixes of clothing items. The number of distinct box types has a major impact on the operating costs. Thus, the PrePack problem consists in determining the number and contents of the box types, as well as the allocation of boxes to stores. This paper introduces the PrePack problem and proposes CP and MIP models and a metaheuristic approach to address it.

1 Introduction

Major retail chains must fulfill requests from hundreds (or even thousands) of stores by dispatching items stored in their warehouses. For instance, in the fashion industry, one must dispatch to each store an assortment of clothes of varied colors and sizes, in such a way that the demand of the store clients is satisfied. The items are usually shipped in boxes containing several different items.

A *box configuration* is a possible way to fill a box. The number of different configurations used has a major impact on the operating costs. If there are many configurations, it will be possible to satisfy the demand of the stores in a precise manner but with elevated operating costs. If the number of configurations is restricted, the operating costs will be low, but some items may be overstocked or understocked at the stores.

The *PrePack* problem consists in determining the number and contents of the box configurations, as well as the configurations to ship to each store, to minimize overstocking and understocking. In this paper we will consider the number of configurations to be fixed, but variants where this number is a decision variable also exist. Moreover, the supplier will usually favor overstocking since understocking decreases customer satisfaction. However, because overstocking is costly for the stores they usually impose a hard limit. Furthermore, it is not permissible to overstock an item in a store where there is no demand for it, because the item would never be sold. Finally, the box capacities are predefined, and all boxes must be full.

The PrePack problem described above is of significant importance and the subject of recent patent applications [1,2,3]. However, to our knowledge, there are no current scientific publications that present an algorithm for it. The most relevant combinatorial problem, on which [1,2,3] build, is the multi-choice knapsack problem described in [4]. We formally introduce the PrePack problem and present three solution approaches. The preliminary results suggest that despite its apparent simplicity, the prepacking problem is a challenging combinatorial optimization problem that deserves to be studied more deeply.

The remainder of the paper is organized as follows. Section 2 formalizes the problem using constraint programming; Section 3 formulates the problem as a mixed integer program; and Section 4 presents hybrid metaheuristics. Section 5 discusses the results obtained by each approach, and Section 6 concludes the paper.

2 Problem Definition

A more precise description of the problem is given through the following constraint programming model. The model, which considers a fixed number of boxes where each box represents a configuration of items, uses the following indices: $i \in I$ for each item, $s \in S$ for each store, and $b \in B$ for each box configuration. From now on the sets will be implied; for example, we will write $\forall(i,s)$ instead of $\forall i \in I, s \in S$. We denote by K the set of possible box capacities (if there is no restriction on the capacities, we simply set $K = \mathbb{Z}_+$).

$$\min \text{Obj} = \sum_{i,s} (\alpha \cdot \mathit{under}(i,s) + \beta \cdot \mathit{over}(i,s)) \tag{1}$$

$$\sum_b \mathit{fill}(b,i) \times \mathit{send}(b,s) = \text{dem}(i,s) + \mathit{over}(i,s) - \mathit{under}(i,s) \quad \forall(i,s) \tag{2}$$

$$\mathit{over}(i,s) \leq \text{overlimit}(i,s) \quad \forall(i,s) \tag{3}$$

$$\sum_i \mathit{fill}(b,i) = \mathit{capa}(b) \quad \forall(b) \tag{4}$$

$$\mathit{capa}(b) \in K \quad \forall(b) \tag{5}$$

$$\mathit{over}(i,s), \mathit{under}(i,s), \mathit{fill}(b,i), \mathit{send}(b,s), \mathit{capa}(b) \in \mathbb{Z}_+ \quad \forall(i,s,b) \tag{6}$$

The objective (1) is to minimize the cost of understocking (*under*) and overstocking (*over*), where α and β are the understock and overstock penalties. Constraint (2) is nonlinear and ensures that the demand (*dem*) of each item in each store is met. Here, fill defines the amount of each item to be packed in each box configuration and *send* indicates how many boxes of each configuration are to be sent to each store. Constraint (3) ensures that the overstocks are less than or equal to the predefined overstock allowances. Constraint (4) ensures that the boxes are full. Finally constraint (5) imposes the restrictions on the box capacities.

Although the above constraints fully define the problem, it may be helpful to add redundant constraints to reduce the solution time. In particular, we considered the following constraints:

$$\sum_b send(b,s) \leq \left\lceil \frac{\sum_i \Big(\text{dem}(i,s) + \text{overlimit}(i,s)\Big)}{\text{[min box capacity]}} \right\rceil \quad \forall(s) \tag{7}$$

$$over(i,s) \cdot under(i,s) = 0 \quad \forall(i,s) \tag{8}$$

$$capa(b) \geq capa(b-1) \quad \forall(b \geq 1) \tag{9}$$

$$\sum_i (under(i,s) + over(i,s)) \geq modulo\left(\sum_i \text{dem}(i,s), 2\right) \quad \forall(s) \tag{10}$$

Constraint (7) is interesting when the box capacities are greater than one since it limits nontrivially the number of boxes to be sent to each store. Although constraint (8) will be automatically satisfied by any optimal solution, imposing it can lead to a substantial reduction of the solution space. Note that the formulation contains several symmetries. Some of them can be removed by adding constraint (9), which ensures that the box configurations are considered in increasing order of their capacity. Finally, unlike the previous constraints, constraint (10) is valid only when the boxes are required to have an even capacity. With this constraint, if the total demand of a store is odd, then there must be some overstock or understock at that store.

3 Mixed Integer Problem Model

For this formulation we need the following variables: y_{bi} = amount of item i in box configuration b; x_{bs} = number of box configurations b shipped to store s; z_{is} = total amount of item i shipped to store s; $t_{bk} = 1$ if box configuration b corresponds to a box of capacity k and 0 otherwise; o_{is} = overstock of item i at store s; and u_{is} = understock of item i at store s. The overall nonlinear model is then:

$$\min Obj = \sum_{i,s} (\alpha u_{is} + \beta o_{is}) \tag{11}$$

$$z_{is} - o_{is} + u_{is} = \text{dem}(i,s) \quad \forall(i,s) \tag{12}$$

$$z_{is} = \sum_b x_{bs} y_{bi} \quad \forall(i,s) \tag{13}$$

$$\sum_i y_{bi} = \sum_k k \cdot t_{bk} \quad \forall(b) \tag{14}$$

$$\sum_k t_{bk} = 1 \quad \forall(b) \tag{15}$$

$$o_{is} \leq \text{overlimit}(i,s) \quad \forall(i,s) \tag{16}$$

$$t_{bk} \in \{0,1\}, x_{bs}, y_{bi}, z_{is} \in \mathbb{Z}_+, o_{is}, u_{is} \geq 0 \quad \forall(b,i,k,s) \tag{17}$$

Constraints (12) and (13) ensure that the store demands are satisfied, with possible understock and overstock. Constraints (14) and (15) ensure that each box is completely filled to one of the predefined capacities. Constraints (16) restrict the overstocking, while constraint (17) defines each variable's domain.

The above formulation can be linearized by standard techniques to enable the use of standard solvers such as CPLEX. We decompose the x variables by introducing binary variables v_{bsl} such that $x_{bs} = \sum_l 2^l \cdot v_{bsl} \ \forall(b,s)$. When multiplying x_{bs} by y_{bi}, we obtain the product $w_{bisl} = v_{bsl}y_{bi}$. We replace this product by w_{bisl} and add the following constraints:

$$w_{bisl} \leq \bar{\mathrm{Y}} v_{bsl} \quad \forall(b,i,s,l) \tag{18}$$

$$w_{bisl} \leq y_{bi} \quad \forall(b,i,s,l) \tag{19}$$

$$w_{bisl} \geq 0 \quad \forall(b,i,s,l) \tag{20}$$

$$w_{bisl} \geq y_{bi} - \bar{\mathrm{Y}}(1 - v_{bsl}) \quad \forall(b,i,s,l) \tag{21}$$

$$z_{is} = \sum_{b,l} 2^l \cdot w_{bisl} \quad \forall(i,s) \tag{22}$$

where $\overline{Y}$ is an upper bound on the variables y_{bi}. The linearized model is then obtained by replacing (13) by (22) and removing the x variables.

Of the constraints (7)–(10), the last one proved to be helpful when the capacities of the boxes are even, and it was introduced in the following form:

$$\sum_i (u_{is} + o_{is}) \geq modulo\left(\sum_i dem(i,s), 2\right) \qquad \forall(s) \tag{23}$$

In contrast, the symmetry-breaking constraints (9) did not help and thus were not included in the final model.

4 Hybrid Metaheuristic

We developed a two-phase hybrid metaheuristic. In the first phase, the approach uses a memetic algorithm (MA) to explore the solution space and builds a pool of *interesting* box configurations. In the second phase, the approach solves a box-to-store assignment problem, to i) choose a subset of configurations from the pool and ii) decide how many boxes of each configuration should be sent to each store.

In our MA, individuals are represented using a multi-array genotype. Each of the $|B|$ arrays in the genotype represents a box configuration, i.e., a vector of integers with $|I|$ positions. The initial population is built using three constructive heuristics: demand-driven insertion, cost-driven insertion, and random insertion. The first heuristic fills boxes with items selected based on the demand across stores. The second heuristic fills boxes based on an estimation of the impact of the resulting configuration on the objective function. The third heuristic has two steps: in the first step it randomly selects a box capacity, and in the second step it fills the selected box with a random number of items of each type.

The initial population is evolved using two evolutionary operators, namely, crossover and mutation. The crossover operator, *horizontal-vertical crossover*, recombines two parents to form four offspring. The underlying idea is to mix whole box configurations to generate a new individual and also to build new box configurations by recombining existing packing patterns. Figure 1 illustrates the operator for $|B| = 3$. The mutation operator simply sweeps each configuration in the mutating individual and swaps some of the item values. Finally, the local-search operator tries to improve an individual by changing the number of each type of item in each of the individual's configurations. During the execution of the MA, the fitness function of a solution is computed by heuristically solving the box-to-store assignment problem using a fast procedure. To control the evolutionary process, we borrowed the logic of the generational MA introduced in [5]. To support our implementation we used the Java Genetic Algorithm framework [6]. The second phase of our approach consists in solving the box-to-store

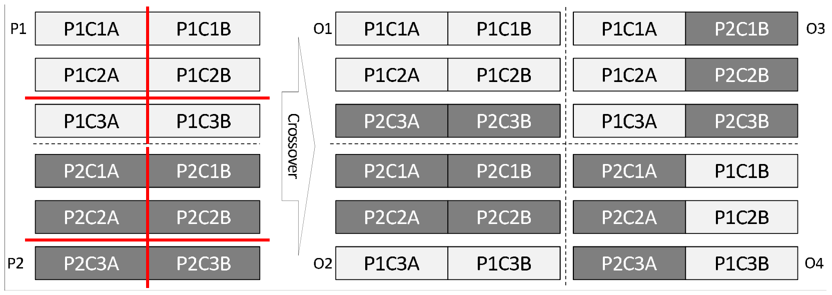

Fig. 1. Horizontal-vertical crossover example

assignment problem over a set of configurations C found during the MA. The composition of C may differ depending on the MA's parameters. To solve the assignment problem we use the following set covering model:

$$\min \sum_{i,s} \alpha u_{is} + \beta o_{is} \tag{24}$$

$$\text{s.t.} \sum_{c} a_{ic} x_{cs} = d_{is} + o_{is} - u_{is} \quad \forall (i,s) \tag{25}$$

$$\sum_{s} x_{cs} \leq \bar{X} y_c \quad \forall c \tag{26}$$

$$\sum_{c} y_c = |B| \tag{27}$$

where $c \in C$. The objective (24) is to minimize the understock u_{is} and overstock o_{is} with the parameters α and β representing the understock and overstock penalties. Constraint (25) controls the demand; a_{ic} is the number of item i present in configuration c, and x_{cs} is the number of box configurations c sent to store s. Variable y_c is 1 if configuration c is used and 0 otherwise. Constraint (26) ensures that no unused configuration is sent to a store. Constraint (27) ensures that the correct number of configurations is used to represent a solution.

To solve (24)–(27) we use either a commercial solver (CPLEX) or a large neighborhood search (LNS). Starting from a valid solution, the LNS partially destroys the solution with one operator and reconstructs it with another operator. The destruction operator selects random box configurations to remove from the solution. The reconstruction operator is based on a best-insertion algorithm. The algorithm adds to the solution the box configuration that provides the smallest increase in the value of the objective function. The objective function is determined after an assignment heuristic has assigned the box configurations to the stores. LNS repeats this process until the predetermined number of box configurations to use has been satisfied.

5 Results

We tested our approaches on variants of a real-world instance with 58 stores demanding 24 ($= 6 \times 4$) different items: T-shirts available in six different sizes and four different colors (black, blue, red, and green). Each item has a fixed overstock limit (0 or 1) for all stores but no understock limits. The available box capacities are 4, 6, 8, and 10. Finally, the overstock and understock penalties are $\beta = 1$ and $\alpha = 10$. From this instance we derived smaller instances obtained by considering only some of the colors and/or only the first ten stores. We set the maximum execution time to 15 minutes.

The results are presented in the following two tables. The second column indicates the number of box configurations used to build the solution. Columns 3 and 5 indicate the number of understocked and overstocked items across all the stores. Columns 4 and 6 indicate the time needed to prove optimality in the case of the exact models (unless the maximum time is reached, in which case we indicate the best solution found) or to find the best solution for the metaheuristic. Columns 7 and 8 indicate respectively the number of nodes in the branching tree and the resulting gap.

5.1 Exact CP and MIP Models

For the constraint programming model (CP), the results show that the model is able to quickly prove optimality when no understocking is required. This is because there exists a feasible solution for which (10) is tight. The mixed integer problem (MIP) approach performs efficiently on all the mono-color instances. However, for larger instances, there are two issues, First, it has difficulty finding good feasible solutions. For example, the LP relaxation value for the instance `BlackBluex10` coincides with the optimal value; what prevents CPLEX from solving the model is its inability to find the corresponding feasible solution. Second, it has difficulty improving the best bound; CPLEX may not be able to

Instance	No.	CP		MIP			
	Boxes	Under/over stock	CPU (s)	Under/over stock	CPU (s)	No. Nodes	Gap
Black x 58	4	(0; 58)	19	(0; 58)	0.09	27	0%
Blue x 58	4	(0; 0)	118	(0; 0)	0.83	580	0%
Red x 58	4	(16; 0)	900	(16; 0)	0.42	523	0%
Green x 58	4	(0; 0)	0.5	(0; 0)	0.04	4	0%
BlackBlue x 10	7	(0; 10)	300	(7; 13)	900	88900	87.95%
BlackBlue x 58	7	(76; 110)	900	(36;106)	900	55198	87.55%
AllColor x 10	14	(33; 3)	900	(47; 19)	900	3700	98.77%
AllColor x 58	14	(401; 93)	900	-	900	0	-

improve the lower bound found at the root node. For the instance `AllColorx58`, CPLEX is not even able to process the root node (the time limit was reached as CPLEX's heuristics were trying to find a feasible solution). The results are presented with CPLEX's default parameters; some other settings have been explored without any improvement.

5.2 Hybrid Metaheuristic

The metaheuristic was run with two separate configurations: solving (24)–(27) using CPLEX over the set C made up of all the box configurations found in the individuals of the MA's final populations; and solving (24)–(27) using LNS over the set C made up of all the box configurations explored during the MA's execution (that is, a much larger set of columns). The results show that on the

Instance	Nb of	CPLEX		LNS	
	boxes	under/over stock	CPU (s)	under/over stock	CPU (s)
Black x 58	4	(0; 58)	6	(0; 58)	7
Blue x 58	4	(1; 1)	7	(10; 10)	8
Red x 58	4	(58; 0)	7	(50; 0)	8
Green x 58	4	(0; 0)	7	(0; 0)	8
BlackBlue x 10	7	(4; 26)	7	(1; 11)	16
BlackBlue x 58	7	(35; 175)	43	(0; 174)	74
AllColor x 10	14	(18; 22)	49	(7; 19)	293
AllColor x 58	14	(168; 146)	273	(40; 148)	900

mono-color instances both metaheuristics find two of the four optimal solutions. On the larger instances, it is clear that the CPLEX-based model is less effective than the LNS model. This is because the latter approach has a larger solution space to explore.

6 Conclusion

We have introduced the PrePack optimization problem, a problem that does not seem to have been studied before. Our preliminary results show that this problem can be very hard, even for relatively small instances, as illustrated by the CP and MIP approaches. The hybrid metaheuristic was able to return a solution for all instances in a relatively short time. However, the results are not as good as those of the the CP and MIP approaches. This is because the quality of the results delivered by the second phase depends on the pool of generated configurations.

This problem certainly deserves further study. More valid inequalities and cuts will be necessary to improve the performance of the MIP approach. The performance of the CP model could be improved by adding effective surrogate constraints. The metaheuristic is promising because it is faster than the exact models, but the pool-generation phase needs to create more effective configurations to produce better solutions.

Acknowledgement. This paper is based on work that was initiated during the *Fifth Montreal Problem Solving Workshop* held at the *Université de Montréal* in August 2013. We would like to acknowledge financial support from the CRM and Mprime, and we thank the CRM for providing a stimulating working environment during the workshop. We are especially grateful to Eric Prescott Gagnon from JDA Software for proposing the problem and for being present throughout the week to help us understand the context.

References

1. Erie, C.W., Lee, J.S., Paske, R.T., Wilson, J.P.: Dynamic bulk packing and casing. International Business Machines Corporation, US20100049537 A1 (2010)
2. Vakhutinsky, A., Subramanian, S., Popkov, Y., Kushkuley, A.: Retail pre-pack optimizer. Oracle International Corporation, US20120284079 A1 (2012)
3. Pratt, R.W.: Computer-implemented systems and methods for pack optimization. SAS Institute, US20090271241 A1 (2009)
4. Chandra, A.K., Hirschberg, D.S., Wong, C.K.: Approximate algorithms for some generalized knapsack problems. Theoretical Computer Science 3(3), 293–304 (1976)
5. Mendoza, J.E., Medaglia, A.L., Velasco, N.: An evolutionary-based decision support system for vehicle routing: The case of a public utility. Decision Support Systems 46, 730–742 (2009)
6. Medaglia, A.L., Gutérrez, E.J.: An object-oriented framework for rapid development of genetic algorithms. Handbook of Research on Nature Inspired Computing for Economics and Management. Idea Publishing Group (2006)