2009 Ninth International Conference on Intelligent Systems Design and Applications

Optimizing allocation in floor storage systems for the shoe industry by Constraint Logic Programming

Antonella Meneghetti
DiEM – Dipartimento di Energetica e Macchine,
University of Udine
33100 Udine, Italy
meneghetti@uniud.it

***Abstract*— Floor storage systems are used in the shoe industry to store fashion products of seasonal collections with low quantity and high variety. Since space is precious and order picking must be sped up, stacking of shoe boxes should be optimized. The problem is modeled by assigning an integer code to each box basing on shoe characteristics (model, material, color, and size) and trying to force similar boxes into near locations to improve pickers' ability of fast order retrieval. The model is encoded in Constraint Logic Programming and solved comparing different strategies, also using Large Neighborhood Search.**

Keywords – Allocation; Floor storage systems; Shoe industry; Constraint Logic Programming; Large Neighborhood Search.

I. INTRODUCTION

Floor storage systems allow very flexible configurations within a warehouse [1], since no additional structure other than pallets is needed to stack stock keeping units (skus). Thus, space can be quickly made available for different skus and adapted to their characteristics in terms of size, inventory quantity and picking frequency, reconfiguring the stocking area to optimize operations. Here the reason why these simple systems are suitable for industries characterized by seasonal productive campaigns. Furthermore, floor storage systems can be used as a temporary solution to face volume increase avoiding other costs, while waiting the construction of more sophisticated structures.

In one of the most famous Italian shoe company, floor storage systems are adopted to manage reorders of collections by retail points during each season. Every shoe model (M) proposed within a seasonal collection can be offered in different materials (T) (leather, tissue, etc.) and colors (C). Production plants all over the world send cartoons containing up to 15 shoe boxes of same MTC and eventually different sizes (S) to satisfy reorders during any fashion season. When these cartoons are broken but only partially used to satisfy client orders during a working period, they produce a certain number of "free pairs". Free pairs enter the floor storage system, where they are stacked until used to satisfy another retail point demand. While more classical products can rely on a significant predictable quantity of reorders per MTC and therefore managed by a proper dedicated storage [2], the so-called "fashion products", which are proposed only in a given collection and usually with a great variety of MTC combinations, lead to very little quantities per MTC that become free pairs. Thus, the floor storage system devolved to fashion free pairs must manage a great variety of skus with very small quantity and hardly predictable arrivals/retrievals during each seasonal collection. Since storage space is precious, optimizing stacking of different shoe boxes becomes important in order to enhance fast order retrieval.

This problem can be formalized as a *Constraint Satisfaction Problem* (CSP), where values representing shoe boxes to be stored are assigned to available locations (variables), subject to a set of constraints [3]. A cost function can be associated with variable assignments, so that the minimum cost solution can be identified, leading to a *Constraint Optimization Problem* (COP). *Constraint Programming* (CP) is a programming methodology that allows to encode and solve CSPs and COPs. CP splits problem encoding into two parts: the modeling phase and the solving one. In the former, the problem is modeled using constraints, which are not limited to linear functions as in LP or MIP; non linear cost functions can also be used. In the latter, the programmer can rely on a constraint solver to find solutions. The constraint solver systematically explores the whole solution space alternating assignment steps, where a variable is selected and assigned to a value of its domain, to constraint propagation ones, where not assignable values are removed from domains. The programmer can improve the performance of this phase adding heuristics for variable choice, value choice selection, etc to the built-in solver. Among the various systems for CP, *Constraint Logic Programming* (CLP) is the more mature constraint programming methodology (the first definition can be traced back to [4]). CLP languages allow a declarative easy encoding, where the focus is on describing properties the desired solution should have, rather than on establishing a procedure to find it. Various families of efficient constraint solvers are then available. Moreover, there are either free and commercial CLP systems that are easy to be installed and used in all the platforms and allow code portability from one system to another. CLP has been successfully used in several industrial applications since nineties [5], while more recent research fields embrace alerting system location [6] and bioinformatics [7].

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.116

II. THE MODEL

The warehouse system is made by sequences of floor pallets, organized into aisles. One side of each pallet is associated with a given aisle, the other side belongs to the previous/following aisle. Given standard sizes of both pallets and shoe boxes, each pallet can contain up to max_{col} columns of shoe boxes on both sides, stacked up to max_{box} units ($max_{box} \leq 10$ to assure stability), for a maximum of $max_{col} \times max_{box}$ boxes per pallet side.

Pallets are set in U shape and numbered progressively. Thus, all available locations (the variables) in our storage system are uniquely identified by a tuple *(a, p, i, j)* of integers describing the number of the aisle, the number of the pallet within the given aisle ([1..max_{pal}]), the number of the column occupied in the pallet ([1..max_{col}]) and the slot ([1..max_{box}]) into the analyzed column, respectively. Furthermore, when long aisles should be preferred basing on warehouse floor size, storage quantities and available operators, then aisles could be split into their two fronts and assigned to two consecutive aisle indexes, if different behaviors or operators for any long front are desirable.

Each stock keeping unit (sku) is represented by a shoe box, which is characterized by an integer code ranging from 1000 to 99999. Thousand digits represent the model of shoes and given the high variety required by fashion products, 99 different models are considered in the current case. The model codes should be assigned consecutively to similar models. Every shoe model can be realized in different materials coded by the hundred digit, thus allowing 10 different combination (numbered 0-9) per model. Each model-material combination can be proposed to customers in different colors, represented by the decade digit, thus considering 10 alternatives. Finally, shoes are characterized by their size, described in our code by the unit digit, thus considering 10 different sizes per shoe model, taking into account the different measures required by men, women, juniors, and babies. Therefore, to each shoe box an integer code of 4 to 5 digits of the type MTCS is assigned (Model 1-99, material Type 0-9, Color 0-9, Size 0-9). If more models/parts/colors/sizes are needed to match with actual cases, the number of related digits can be properly increased, moving to greater integers. When a location is assigned to a shoe box, the related variable is set to its MTCS code, otherwise is set to zero.

A. Constraints

Constraints are imposed so that symmetries of solutions are broken: columns are filled in a bottom-up way resembling actual stacking of shoe boxes during storage operations and each pallet is filled from the first to the last column without empty stacks in the middle.

To force similar shoes to be stacked together, boxes assigned to a column must belong at least to the same shoe model, i.e. thousand digits must be the same for a given column (1). Furthermore, boxes into a pallet must be characterized by similar models, thus allowing a range of only $2 \cdot max_{mod}$ consecutive models to be assigned to a given pallet (2). Max_{mod} should be selected basing on the number of models to be storage in the planning horizon (i.e. models of a seasonal collection for fashion industries) and available columns. When very few space is available in the warehousing system, this constraint should be removed and great model variability within a pallet allowed.

$$int(x_{a,p,s,i}/1000) = int(x_{a,p,s,j}/1000) \quad \forall\,(a,p,s) \tag{1}$$

$$|int(x_{a,p,1,\_}/1000) - int(x_{a,p,i,\_}/1000)| \leq max_{mod} \quad \forall (a,p,i) \tag{2}$$

Finally, the number of non zero variables (i.e. the locations to be occupied) should be exactly equal to the number of new entering boxes and their values must be chosen among their integer codes.

When multiple aisles are managed and a class-based allocation [2] should be preserved in order to limit variety of models within an aisle and enhance pickers' capability of fast retrievals, constraints linking models belonging to the same class (e.g. woman, man, junior, or baby shoes) to a given aisle should be added.

B. The cost function

To speed up manual retrieval operations, workers should rely on a logical scheme of shoes distribution along aisles, so that similar shoes (i.e. characterized by consecutive integers) are stacked as close as possible.

The cost function C_{tot} to be minimized is made up by 5 different contributions, which resemble the need of proximity of similar shoe boxes, as shown below (3):

$$C_{tot} = \sum_i \left(C^i_{col} + C^i_{e\_col} + C^i_{prox}\right) + \sum_s \left(C^s_{pal} + C^s_{e\_pal}\right) \tag{3}$$

We would like to store shoes differing only by their size in the same column, so that workers can easily retrieve them basing on client reorders. To force boxes with similar characteristics (i.e. with near integer codes) to enter the same stack, a column cost C_{col} is calculated for every column i with a new entering box, as the sum of the difference between each in-box code and the codes of shoe boxes already stacked, plus the code difference between every combination (without repetition) of any two entering boxes assigned to that column. If R is the set of locations within a column already filled with shoe boxes during past replenishments and V is the set of empty locations, then:

$$C_{col} = \sum_{i \in V} \sum_{k \in R} |x_{a,p,s,i} - x_{a,p,s,k}| + \sum_{(i,j) \in V} |x_{a,p,s,i} - x_{a,p,s,j}| \quad \forall (a,p,s) \tag{4}$$

In order to minimize the number of different shoe codes within a column, the cost contribution is set to zero for a new box with a code already present into the stack and new boxes with the same code are counted only once for a given column.

We would like to store shoes differing only by their size in the same column, so that workers can easily retrieve them basing on client reorders. To force boxes with the same model, type and color (MTC) in the same stack, a cost to fill a new column C_{e_col} is introduced, in order to discourage splitting of boxes differing only by their unit digit. Since shoes with same MTC are expected to be stacked into the

same column if possible, we set C_{e_col} equal to the column cost $\widehat{C_{col}}$ in the worst desirable case, when new entering boxes are characterized by same MTC but all different sequential sizes. If codes are sorted in decreasing order, any box code differs from the successive ones for a quantity growing from 1 to its unit value. Basing on (4), $\widehat{C_{col}}$ can be evaluated as:

$$\widehat{C_{col}} = \sum_{n=1}^{max_{box}-1} n(n+1)/2 \qquad (5)$$

C_{e_col} should be greater than $\widehat{C_{col}}$ to discourage a new column occupation when boxes are characterized by same MTC. Thus, we set C_{e_col} equal to the nearest integer multiple of 5.

Eventually, boxes with near integer codes, but differing for more than their size, can present the same cost C_{col} and could be forced into the same column (e.g. boxes 1009 and 1011 differ for 2 units as boxes 1007 and 1009, but have different colors). Thus, a weight equal to C_{e_col} is added to (4) to distinguish these particular cases, so that filling another column is allowed, when enough space is available in the warehouse.

Within a given pallet, we would like to have the most similar shoes, so comparisons between all the boxes in that pallet are performed, by calculating a pallet cost C_{pal} very similar to the column one, but encompassing all the ($max_{box} \times max_{col}$) possible allocations, considered as a unique column of $max_{box} \times max_{col}$ elements. Therefore, the same rules for column cost calculation (4) are applied to the pallet component, without adding any weight.

As in the column case, we would like to encourage very similar shoes to occupy the same pallet in order to enable fast picking. In the worst of desirable cases, boxes in each column differ only by size (the unit digit) and different columns differ from each other only by their shoe colors (i.e. progressive decade digits are considered). In such a situation, the pallet cost component will be equal to $\widehat{C_{pal}}$, as shown in (6).

$$\widehat{C_{pal}} = \sum_{n=1}^{max_{col} \times max_{box}-1} n(n+1)/2 \qquad (6)$$

We introduce a little greater empty pallet cost C_{e_pal}, equal to the nearest integer multiple of 10, to avoid splitting of same size and color boxes among many pallets.

Finally, a proximity cost is introduced to force similar shoes to occupy adjacent pallets. This cost component is calculated by the average sum of difference between the model code of any new column $\bar{s}$ (see constraints in par. II.A) and model codes of non-empty stacks (in total $\bar{N}$) occupying the previous and the following pallet, as shown in the following (7).

$$C_{prox} = \frac{1}{\bar{N}} \left[\sum_{j=1}^{max_{col}} \left| \text{int}\left[\frac{x_{a,p,\bar{s},}}{1000}\right] - \text{int}\left[\frac{x_{a,p-1,j,}}{1000}\right] \right| + \sum_{j=1}^{max_{col}} \left| \text{int}\left[\frac{x_{a,p,\bar{s},}}{1000}\right] - \text{int}\left[\frac{x_{a,p+1,j,}}{1000}\right] \right| \right] \quad \forall(a,p,\bar{s}) \qquad (7)$$

Since cost components have different magnitude because of the different number of pair-wise comparison involved, weights are introduced in order to counterbalance them. In particular, C_{pal} is divided by a weight based on the ratio $C_{e\_pal}/(C_{e\_col} \times max_{col})$.

III. CLP Optimization

A program was created in SICStus Prolog (www.sics.se/sicstus) to solve the problem, encompassing the three typical steps of CLP approach: (1) define the domain of each variable; (2) declare problem constraints; (3) search for a good feasible solution or find an optimal one exploring the whole search tree by branch and bound techniques.

An example of constraint encoding is shown in Figure 1, where boxes stacked into the same column are forced to have the same shoe model (constraint (1)). The storage system is encoded as a list of lists (the columns), which are themselves lists of 4-5 digit (MTCS) integers (the box codes).

Given the first two CLP steps, both the searching strategies (optimization and backtracking) were applied, as described in par. III.A and par. III.B, respectively. Two heuristics are proposed to be used while exploring the search tree, in order to reach good solutions faster than built-in procedures provided by the CLP over finite domains (CLP(FD)) solver of SICStus Prolog. The variable choice heuristic and the value choice heuristic are described in par. III.B.

A. The CLP minimize approach

The CLP(FD) solver of SICStus Prolog [8] provides a branch and bound algorithm for optimization, with different options for variables and values choice selection.

As regards the choice of the next variable to be assigned, with the leftmost option, the leftmost variable is chosen, while with first fail the leftmost variable with the smallest domain is selected. Finally the most constrained (Ffc) option is tested: a variable with the smallest domain is selected, breaking ties by (a) selecting the variable that has the most constraints suspended on it and (b) selecting the leftmost one.

Concerning the way in which values are assigned to the selected variable, if the step option is selected, one value per time of the finite domain is tested and the domain is explored in increasing or decreasing order based on up and down option respectively. With the bisect option domains are split into two parts by their middle point, exploring the lower/upper part of domains first depending on up or down option respectively.

Finally, the above options were replaced by the search heuristics described in the following par. III.B, trying to speed up the search process.

```
constr_col([]).
constr_col([[_,_,_,COL]|R]) :-
  same_model(COL),
  constr_col(R).

same_model([]).
same_model([_]).
same_model([A,B|R]):-
  A #> 0 #/\ B #> 0 #=> A/1000 #= B/1000,
  same_model([B|R]).
```

Figure 1. Encoding of column constraint (1).

B. The search heuristics

The variable choice heuristic controls the order in which the next variable is selected for assignment. Empty locations on partially occupied columns are selected as the first variables to be assigned, then empty columns in partially occupied pallets are considered, while empty pallets are selected in the end. This selection should force new in-boxes similar to already stored skus to be stacked near them, whenever possible. To enhance the ability of fast finding good solutions, random permutation within the two groups of empty columns is performed.

In the value choice heuristic, instead, a hierarchical procedure is proposed to identify alternative values to be assigned to a given variable, when a branch fail occurs. The following hierarchy of choice is adopted:

1. The value assigned to the previous selected variable, if different from zero;
2. A feasible value with same MTC of the last assigned variable, if different from zero, in increasing order;
3. A feasible value with same MT of the last assigned variable if different from zero, in increasing order;
4. A feasible value with same M (model) of the last assigned variable if different from zero, in increasing order;
5. A feasible value with characteristics different from the last assigned one, if different from zero, in increasing order;
6. If the last assigned value was zero, then choose a feasible positive value in increasing order;
7. Eventually, set the variable to zero, thus leaving empty the related location.

The reference value for the first variable is set to the lowest entering box code. The position of step 7 within the above hierarchy of choice is made dynamic basing on the percentage of space available in the floor storage system. When the warehouse has less than 20% of locations already occupied, enough space is available to storage boxes preserving their MTC characteristics, i.e. stacking into the same column boxes differing only by their size. In this case step 7 is shifted soon after step 2 and the selected variable is set to zero immediately after failing step 2. As the number of available locations becomes lower and lower, stacking boxes with different characteristics in the same or near column becomes more probable and therefore step 7 is progressively moved after each following step every 20% increase of occupied space, becoming the last possible choice if more than 60% of locations are already occupied.

C. Searching solutions by CP and LNS

Adopting a feasible solution strategy by backtracking search instead of an optimal one, the labeling predicate of SICStus Prolog is powered by the variable choice heuristic for assignment order selection and the value choice heuristic for domain exploration, previously described in par. III.B. An iterative procedure moving towards lower and lower cost values was implemented, in order to find the best feasible solution available when the time-out condition is eventually reached. Its performances are compared to minimize procedure's ones, when the number of available locations in the floor storage systems (variables) and the number of in-boxes are progressively increased.

To further improve the capability of obtaining a near optimal/optimal solution with lower and lower computational times, a local search procedure is added after obtaining a good solution by the above mentioned heuristics. In particular a Large Neighborhood Search (LNS) is adopted [9]. A LNS algorithm is an iterative process that destroys at each iteration a part of the current solution using a chosen neighborhood definition procedure and reoptimizes it, hoping to find a better solution. The neighborhood procedure selects the subset of variables, the so-called "free variables" (FV), that should be reassigned, while maintaining the others unchanged wrt the current solution. The constraint structure of the model is preserved, in order to find only feasible solutions. In our case, top positions of the good solution are randomly made variable again and reassigned in order to lower the cost function. In particular, two kind of moves are allowed for a given solution: two boxes on the top of related columns can be switched or a box can be removed from the top of its column and stacked on the top of another column. Thus, the first empty location of each column and the top location occupied by an entering box are selected to become FV and be reassigned. Furthermore, the number of FV to be managed by a LNS run is kept low by randomly extracting among the selected top locations ($2max\_stack \times max\_pals$ in the worst case), starting with a narrow group of variables and increasing its size if no improved solutions can be found. FV are then reassigned by using the CLP leftmost step up strategy, with minimizing or iterative backtracking approach depending on problem size.

Since LNS is itself based on a random extraction of variables, random permutation within the 3 groups of variables in the variable choice heuristic (par. III.B) is removed. In this way, we capitalize on the sorting process provided by the CP heuristics to obtain a good starting solution and leave shifting of boxes to LNS ability of improving a given configuration faster.

IV. RESULTS

Experiments were run on a Windows Vista laptop Intel Core 2 Duo, 2.6 GHz, 3 GB.

Any input configuration can be described by 3 different parameters: the number of in-boxes to be stored, the size of the floor storage system (i.e. number of locations), and the percentage of locations already occupied. The last two parameters are related to the number of variables to be assigned and therefore to the size of the problem. The former is associated to the number of non-zero variables to be assigned.

A first group of experiments were performed involving 3 pallets and 5 boxes per stack, for a maximum of 75 available locations when the floor storage system is empty, i.e. at the beginning of reorders for a seasonal collection. This relative small instance allowed to compare all the minimize labeling options described in par. III.A. All the options require very similar run times (about 7 s), excepted for the Ffc variable choice option, which leads to worse run times with different M in-boxes. Generally, when entering boxes are more

similar (e.g. same M, MT or even MTC) run times increase (see column 2 in Table I) and the Ffc option gets the best performance, since more constraints are involved. The variable choice and value choice heuristics don't improve the minimize approach performance and this is because all the solutions must be generated in order to identify the optimal one.

Relative performances of labeling options remain even when a partially filled storage system is adopted (20%, 40%, 60% and 80% of already filled locations were considered), but run times become lower and lower.

The capability of the heuristics to find the best solution faster than the built-in procedures, however, results when not all the solutions should be generated, but it is required to find the assignment related to the minimum cost. In Table II, run times of the leftmost step up procedure are compared to the proposed heuristics ones, when the best solution is already known. Heuristics lead to dramatically lower run times (from a minimum of 52 times for consecutive M to 3890 times for boxes with same MTC).

When the size of the floor storage system is increased in terms of locations to be assigned and consequently the size of the problem (i.e. the number of variables) grows, run times of the minimize approach become unacceptable. With 10 pallet per aisle, several hours are required to reach the optimum, even when only 5 boxes per column are considered (i.e. 250 variables, see Table I). If 10 boxes are allowed to be stacked into the same column (the extreme situation in real applications), run times for some instances exceed one day of computation. Furthermore, increasing the number of entering boxes to be located (i.e. the number of non-zero variables to be assigned) dramatically rises run times. For 10 boxes with very different models and 75 available locations, run time is 4 hours (versus 7.2 seconds for 5 in boxes).

By the backtracking approach, floor storage system size and entering boxes can be increased obtaining good solutions in more reasonable times. Heuristics prove their force in managing similar entering boxes, which obtained the worst run times with the minimize approach (see Table I).

TABLE I. RUN TIMES OF LEFTMOST STEP UP MINIMIZE [MIN].

In Boxes	75 vars	250 vars	500 vars
5 different M	0.12	47.58	271.90
5 consecutive M	0.78	173.41	708.63
5 same MTCS	0.05	4.22	17.62
5 same MTC	5.19	455.74	> 1500.00
5 same MT	6.78	549.13	> 1500.00
5 same M	6.87	554.85	> 1500.00

TABLE II. RUN TIMES [S] WHEN THE MINIMUM COST IS IMPOSED AND 75 AVAILABLE LOCATIONS CONSIDERED.

In Boxes	Leftmost [s]	Heuristics [s]
5 same MTCS	2.761	0.016
5 different M	2.012	0.063
5 consecutive M	12.303	0.234
5 same MTC	120.619	0.031
5 same MT	114.751	0.062
5 same M	117.016	0.062

For 5 same MT entering boxes we obtained the optimal solution in 1 iteration in 0.079 s, 0.25 s and 0.749 s for 75, 250, 500 available locations in an empty system respectively, which are dramatically lower than the related times in row 5 of Table I. For 5 different M boxes, run time is 0.063 s.

To simulate actual situation in shoe industry, experiments were run taking into account from 10 up to 40 entering boxes and a floor storage system of 10 pallets in one aisle (multiple aisles are managed by a class-based allocation policy and therefore it is imagined to run heuristics for each aisle/class separately, especially if a zone-picking is adopted and workers have their specific aisles to serve [2]). A 60% available storage capacity is considered: for a 5 boxes per column configuration, 100 filled locations and 150 empty ones are taken into account. With 10 very different M in-boxes, the minimize approach requires about 1 day of computational time; while heuristic based CLP search is able to reach a solution 7% far from the optimal one in 1 hour and half (see Table III). Even if run time has been drastically reduced, it is still too much longer for real applications.

Table III highlights how heuristics are able to reach a solution 15% far from the optimal one in a very low time (100 s), but further improvements are quite time expensive. This is the reason why a local search approach was introduced. After obtaining a good solution in a relative small time (total time out at 100 s), the iterative LNS procedure (see par. III.C) is added with a global time-out of 100 s, in order to make the improvement phase faster. Results are shown in the last row of Table III: only 132 s are needed on average to reach the optimal solution.

The number of entering boxes was then increased to 20, 30 and 40 with 10 different models, thus introducing a certain degree of similarity (for 40 boxes, 2 same MTCS + 2 same MT boxes per model), as the actual "free pairs" generation process suggests. Number of runs for the CLP heuristic phase and maximum number of runs and time-out per run for the LNS phase had to be identified by trials and errors to find a proper balance. Since heuristics (H) are more time expensive, the mixed approach consists of 1 iterative heuristic based procedure and 2 iterative LNS procedures in turn, starting from the same initial good solution (LNS solutions are compared and the worst discharged). Results are shown in Table IV.

TABLE III. RUNTIMES FOR A 250 LOCATION SYSTEM, 60% AVAILABLE CAPACITY AND 10 VERY DIFFERENT M IN-BOXES.

Strategy	Cost	Δ% min	Runtime [min]
Minimize	2415	0%	1473.3
Heuristics	2775	15%	1.7
	2595	7%	92.8
CP + LNS	2415	0%	2.2

TABLE IV. RESULTS FOR MIXED HEURISTICS (H) + LNS APPROACH.

In boxes	Runs H	Time Tot H [s]	Max runs LNS	Tout LNS [s]	Best cost	Mean cost
20	7	27	150	2	5156	5174
30	5	96	150	4	8779	8951
40	3	20	150	5	14177	14869

V. CONCLUSIONS

Floor storage systems represent a high flexible low-cost solution for a temporary inventory or a seasonal business. When a great variety of products in very small quantities should be managed in the short term, the effort of combining space savings and fast picking operations leads to the need of rational allocation of items along aisles and within pallets.

In the shoe industry, where different fashion products are proposed collection after collection, a picker should rely on a logical stacking of shoe boxes, basing on their characteristics in terms of model, material, color and size, so that similar products are likely stored in near positions and their identification could be faster even in the absence of sophisticated recognition systems.

Mixing CP and LNS revealed a powerful methodology for solving such allocation problems in floor storage systems. Computational time for very good solutions are reasonable low to make the proposed methodology being applied in real warehousing of seasonal low quantity high variety products. A case study of one of the world-wide known Italian shoe company highlighted how allocations of fashion shoe boxes are generally performed twice per day in the floor storage system. Therefore, run times provided by CP+LNS solving methodology (roughly 5 min for 40 in-boxes) looks adequate to such a planning period, even when a high number of product classes and aisles should be considered to efficiently manage order picking. Given the complexity of the problem even with a small number of entering shoe boxes, such timely results could hardly be reached by traditional optimizing approaches.

Furthermore, the declarative nature of CLP allows the programmer to easily describe what properties are required to the desired solution. Requirements can be modified, added or deleted to adhere to a dynamic industrial environment without changing the basic model, but only declaring new constraints, making it adaptable and transferrable to different industrial realities. In the analyzed decision making contest, such CLP flexibility is precious to develop a warehousing tool effectively usable collection after collection. Shoe collections, in facts, differ from one another for seasonal and fashion characteristics, thus affecting stacking requirements. Moreover, different clients' behaviors can impact on picking strategy and therefore on properties storage solutions should have to speed up operations and offer quick response to clients, as fashion market requires. The proposed CLP based methodology presents the required capability of customizing solution properties still maintaining the basic conceptual model.

ACKNOWLEDGMENTS

I'm grateful to Agostino Dovier for his precious support in Constraint Logic Programming.

REFERENCES

[1] J. J. Bartholdi III and S.T. Hackman, *Warehouse & Distribution Science*, Georgia Institute of Technology, 2008, available at http://www.warehouse-science.com;

[2] G. P. Sharp, "Warehouse Management", *Handbook of Industrial Engineering*, G. Salvendy Ed., Wiley Interscience, 2001;

[3] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming,* Elsevier, NY, USA, 2006;

[4] J. Jaffar and J. L. Lassez, "Constraint Logic Programming", *Proceedings of POPL* 1987, 111-119;

[5] J. Jaffar and M. J. Maher, "Constraint Logic Programming: A Survey", *Journal of Logic Programming*, 19/20: 503-581, 1994

[6] F. Avanzini, D. Rocchesso, A. Belussi, A. Dal Palù, and A. Dovier, "Designing an Urban-Scale Auditory Alert System", *IEEE Computer* 37(9): 55-61, 2004;

[7] A. Dal Palù, A. Dovier, and F. Fogolari, "Constraint Logic Programming approach to protein structure prediction", *BMC Bioinformatics*, 5:186, 2004;

[8] Carlsson M., Ottosson G., and B. Carlson, "An Open-Ended Finite Domain Constraint Solver", *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997;

[9] P. Shaw,"Using constraint programming and local search methods to solve vehicle routing problems", *Proc. of CP '98*, 417–431, Springer-Verlag, 1998.