International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Shelf Space Allocation Problem Using LNS Solver

Dina Hamad Alghurair[1], Hedaya Ghanim Alshammar[2]

[1, 2]Higher Institute of Telecommunication and Navigation, HITN, PAAET, Ardia Block4, Street 602, Kuwait

Abstract: ***Shelf Space Allocation Problem is the problem of finding a good or an optimal solution for arranging objects on shelves in a store/warehouse to maximize profit. The SSAP yet remains a topic under research and has recently only been applied on a commercial level by large warehouses and the generalization of the problem to meet different requirements is still under investigation by the researchers. The problem considers different shelves with different priorities based on customer experience studies or based on certain criteria defined by the warehouse manager, while products have different dimensions and different profit margin. The current study is an item independent, thus it applies to supermarkets, parts warehouses, or any other space allocation problems with similar distribution criteria. Our study will focus on a simplified version of space allocation problem where we will ignore depth and stacking of same product on shelves, and not consider any clustering or aggregations beyond the fact that all items of same type are allocated within the same shelf next to each other. Our implementation will utilize an LNS solver, and present a planogram in motion, showing the arrangement of products on shelves in a video like sequence to watch the solution improvement and establish a basis for future improvements and generalization of the problem***

Keywords: Shelf Space Allocation Problem, SSAP, LNS, Large Neighborhood Search, PLNS, RLNS, ELNS, Planograms

1. Introduction

Shelf Space Allocation Problem is the problem of solving the allocation of items on shelves, whether this problem is directed towards warehouses, supermarkets, or any storage and allocation service on the market.

The problem on its own has been introduced in the literature by several researchers and is still under investigation and have space for improvements, this stems from the fact that a good allocation can increase the profit of an organization, not to mention the it can optimally utilize the space at hand given a predefined criteria set.

This problem can be described in a complicated context, with many demands and aggregations, and several rules, it can also be simplified and the solution is generalized for a more complicated rules.

Our solution will solve this problem and present planograms in motion to show the change of the objective value and the improvements while still satisfying the criteria of fitting all the objects within the minimum and maximum limits.

The solution at hand can be generalized for more complicated scenarios where similar products can be stacked on top of each other, and have a certain depth count in each allocation.

The problem is a large-scale optimization NP Hard problem, the solution here will present different approaches in the implementation for the same algorithm using LNS Largest Neighborhood Heuristics Search that shows how some approaches are faster than others, and all yield an improved arrangement from scratch given the products / shelves / priorities data set.

2. Literature Review

The shelf space allocation problem have been studied in the literature before in different ways and with different factors, several approaches have been introduced with different algorithms, mostly focusing on large retail stores [1] ,[2],[3],[4]. The topic is considered a commercial topic as companies are increasingly trying to improve their services and stay ahead in the competitive market.

Customer experience analysis has been applied in large retail companies such as Walmart to stay ahead in their competition, utilizing best allocations to maximize their profits.

Many factors affect total profit of a store, as mentioned in [1],[5],[6],[4], for instance the products that are most seen are the ones that are most bought by customers, also the products should always be available and the stock should always be replenished.

In [7] Anderson and his colleagues, investigated the customer preferences for specific brands for the optimal item allocation of shelves.

In [8] Dreze et al. suggested that the numbers of facings on a shelf after a certain limit has no importance and will no longer affect increasing the demand, according to customer experiences and investigations, so even if there is still more space available having more items of the same brand no longer affects the customer purchase process.

In [9] Yang proposed a simplified model to solve the problem depending on the profitability of the products. Most of the research of large warehouse focused on more complicated variables than those available of needed by small warehouses, such as products grouping and complicated customer behaviors, advertising and other factors, while the concept for small warehouses or none customer oriented warehouses doesn't include these factors.

Our work here mostly follows Yang simplified model [9] to establish a basis for a more generalized research on the topic at hand.

3. Large Neighborhood Search To Solve SSAP

Heuristics based on *large neighbourhood search* have been recently tested and showed great results in solving NP hard problems and various large-scale optimization problems such as transportation and scheduling problems, including vehicle routing problems, and the Travelling Sales Man Problem.

Large neighbourhood search methods explore a complex neighbourhood by use of heuristics. Using large neighbourhoods makes it possible to find better candidate solutions with each iteration thus traversing a more promising search path reducing the solution space that the problem is looking in; tell we find an optimal solution or stop by using specific criteria like time and return the resulting solution.

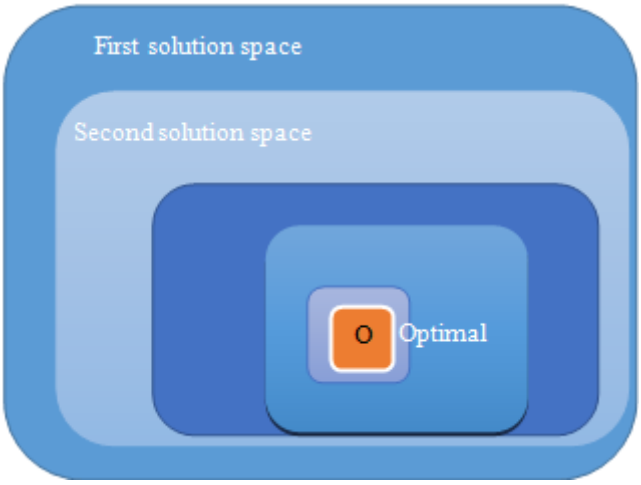

Figure 1: LNS Solution space down to the optimal solution

In the LNS the neighbourhood is defined implicitly by a *destroy* and a *repair* method (try-error-destroy-repair-try…) . A destroy method destructs part of the current solution whereas a repair method rebuilds the destroyed Solution.

The destroy method typically contains an element of stochasticity such that different parts of the solution are destroyed in every invocation of the method. The neighbourhood $N(x)$ of a solution x is then defined as the set of solutions that can be reached by first applying the destroy method and then the repair method.

LNS heuristic typically alternates between an infeasible solution and a feasible solution: the destroy operation creates an infeasible solution, which is brought back into feasible form by the repair heuristic. Alternately the destroy and repair operations can be viewed as fix/optimize operations: the *fix* method (corresponding to the destroy method) fixes part of the solution at its current value while the rest remains free, the *optimize* method (corresponding to the repair method) attempts to improve the current solution while respecting the fixed values. Such an interpretation of the heuristic may be more natural if the repair method is implemented using MIP Mixed Integer Problem or constraint programming solvers.

In our SSAP for instance we fix the current the product allocation on one shelf while the remaining values are randomly generated in search for a better solution, same applied for other variables.

Local search techniques are very effective to solve hard optimization problems. Most of them are, by nature, incomplete. In the context of constraint programming for optimization problems, the basic idea is to iteratively relax a part of the problem, then to use constraint programming to evaluate and bound the new solution by using fix and optimize principle mentioned earlier.

LNS is a two-phase algorithm, which partially relaxes a given solution and repairs it. Given a solution as input, the relaxation phase builds a partial solution (or neighbourhood) by choosing a set of variables to reset to their initial domain; The remaining ones are assigned to their value in the solution. Even though there are various ways to repair the partial solution, the focus is on the technique in which Constraint Programming is used to bound the objective variable and to assign a value to variables not yet instantiated. These two phases are repeated until the search stops (optimality proven, time limit reached or any other criteria defined by the developer, in our case we choose a time criteria of 3 times number of analysed products).

All LNS methods are problem dependent, so several approaches have been proposed by researchers to generalize the LNS and make it problem independent, thus, the following methods have been proposed by researchers: Propagation and explanation based methods.

A simplified general algorithm for the LNS method can be viewed as:

Table 1: LNS Search method (Found in [10])

Algorithm :Large Neighbourhood Search *Requirement: an initial solution S* *procedure LNS* *while Optimal solution not found and a stop criterion is not encountered do* *relax(S)* *S'= findSolution() . The current partial solution is then repaired in order to improve* *the current solution* *if S' != NULL then . An improving solution has been found* *S = S'* *end if* *end while* *end procedure*

3.1 PLNS

One drawback of LNS is that the relaxation process is quite often problem dependent. Some works have been dedicated to the selection of variables to relax through general concept not related to the class of the problem treated. However, in conjunction with CP (Constraint Programming), one generic approach, namely *Propagation-Guided LNS*, has been shown to be very competitive with dedicated ones. It must, in a way,

automatically detect the problem structure in order to be efficient.

In [10] the authors define two neighbourhood selection methods relying on information coming from constraint propagation. The first method defines the set of variables to freeze by incrementally building a partial assignment, starting from an empty scope. The underlying idea is that of freezing related variables. The volume of domain reduced, it helps to link variables together inside or outside partial solutions.

The algorithm uses the current size of the domains of the decision variables to control the size of the neighbourhood, and then when a variable is frozen, propagation occurs, by tracing the volume of the domain reduction. We can detect which variables are linked to the frozen variable; this information will be utilized to choose the next variable to freeze, so the algorithm becomes:

Table 2: PLNS guided search method (algorithm source [10])

Propagation Guided LNS method.
While fragment size is greater than desired size
If variable list is empty then
Choose unbound variable randomly
Else
Choose variable in variable list
End if
Freeze variable and propagate
Update variable list
End while

3.2 RLNS

The LNSFactory provides pre-defined configurations. Here is the way to declare LNS to solve a problem:

LNS Factory.rlns (solver, ivars, 30, 20140909L, new Fail Counter(solver, 100)); solver. Find Optimal Solution (ResolutionPolicy. MINIMIZE, objective); It declares a *Random LNS* which, computes a partial solution based on ivars. If no solution are found within 100 fails (Fail Counter (solver, 100)), a restart is forced. Then, every 30 calls to this neighbourhood, the number of fixed variables is randomly picked. 20140909L is the seed for the java.util.Random. it uses a Random selection scheme of the variables thus it is the slowest method out of the three mentioned here. While the PLNS gives an outstanding performance with optimal or near optimal results.

3.3 ELNS

An explanation is a subset of constraints and decisions which justifies a solver event such as a domain modification or a conflict they are used in various paradigms for improving search. It can be seen as an explicit trace of the propagation mechanism making it possible to identify a set of constraints and decisions (variable assignments, cuts) responsible for the current state of the domain of a Variable.

One is based on an explanation of the inability to repair a solution; the other is based on an explanation of the non-optimal nature of the current solution. An explanation records some sufficient information to justify an inference made by the solver (domain reduction, contradiction, etc.). It is made of a subset of the original propagators of the problem and a subset of decisions applied during search. Explanations represent the logical chain of inferences made by the solver during propagation in an efficient and usable manner.[11]

In a way, they provide some kind of a trace of the behaviour of the solver as any operation needs to be explained. Explanations have been successfully used for improving constraint programming search process. Both complete and incomplete techniques have been proposed.

Those techniques follow a similar pattern: learning from failures by recording each domain modification with its associated explanation (provided by the solver) and taking advantage of the information gathered to be able to react upon failure by directly pointing to relevant decisions to be undone. Complete techniques follow a most-recent based pattern while incomplete technique design heuristics to be used to focus on decisions more prone to allow a fast recovery upon failure.

Explanations are computed in a bottom-up way, from the conflict to the first event generated, keeping only relevant events to compute the explanation of the conflict.

4. Implemented Problem Formulation

The implemented problem has been simplified to include the following variables and constraints
n : number of products to allocate
m : number of shelves
ai : length of facing of product i
Tj : length of shelf j
Li : minimum number of facings required for product i
Ui : maximum number of facings required for product i
xij : number of facings of product i on part k of shelf j
yij = 1 if facings of product i are assigned to shelf j, 0 otherwise
Pi : profit per unit of product i
Cj : priority coefficient of shelf j
Pij = PiCj : profitability of product i if placed on shelf j
i = 1.. n, j = 1 .. m
Objective Function: Maximization of total profit given by :

$$Z = \sum_{i=1}^{n} \sum_{j=1}^{m} Pij.xij \qquad (1)$$

(1) Which means that the summations of all profits per unit product multiplied by the number of facings for a product for all products in the system allocated on all shelves in the system, the maximization of this value is the maximization of the total profit.

Subject to the following constraints:

$$\sum_{i=1}^{n} ai.xij \leq Tj \quad (for\ all)\ j = 1..m \qquad (2)$$

(2) Which says that the summation of the length of all facings of product *i* allocated on a shelf j is less than the length of the shelf for all shelves in the system.

$$\sum_{j=1}^{m} yij = 1 \qquad (for\ all)\, i = 1..n \qquad (3)$$

(3) This constraint states that a product can lie only on one shelf, thus this Boolean variable is either 1 or 0, it will be 1 if facing I lies on shelf j otherwise it will be 0, thus summation of all these variables will be 1 will make the product shelf positioning condition hold, so for this variable there will be nXm different variables, the summation of all variables in the same column is 1.

$$Li \leq \sum_{j=1}^{m} xij \leq Ui \qquad (for\ all)\, i = 1..n \qquad (4)$$

(4) The lower bound and upper bound of the number of facings of a given product, is defined by input, all product facing variables are limited by minimum number of facings condition and maximum number of facings condition defined by the input data.

$$\sum_{i=1}^{n} Li.ai \leq \sum_{j=1}^{m} Tj \qquad (5)$$

(5) is a bound for the entire system which means that the input sample actually meets the minimum space requirements, that means, that before going into attempting to solve the problem we need to verify that there is enough space in the shelves to hold the summation of the minimum bounds of the number of facings multiplied by the length of facings in a shelf for all products and all shelves. For the purpose of implementation and considering that a product is only located on one shelf, the objective can be rewritten as the following

$$Z = \sum_{i=1}^{n}\sum_{j=1}^{m} yij.Pij.xi \qquad (6)$$

where yij will denote the location of product xi.

The Y here will become a 2D array of 0s and 1s, which will become variable array along with xi, thus the problem in (6) is not completely linear, however in our implementation, the Choco Solver, can still handle this type of product of the two variables.

The LNS solver will iterate the Binary array as well as scanning the range of the X array between the lower bound and the upper bound to improve the final solution, depending on which approach is applied (RLNS, ELNS, PLNS).

Other solvers cant work the product of two variables (which makes the problem not truly linear), however; in our case, the none linearity is considered as simple as: Product of a Binary Variable with a Bounded variable (and generalizing it for all the variables at hand). Delinearization of the problem using constraint-programming methods can be done as the following[13]:

Considering:

$$z = A.x\ where\ x\ is\ binary, A\ is\ a\ bounded\ variable \qquad (7)$$

$$min\{0, \underline{A}\} \leq z \leq \bar{A} \qquad (8)$$

$$\underline{A}x \leq z \leq \bar{A}x \qquad (9)$$

$$A - (1-x)\bar{A} \leq z \leq A - (1-x)\underline{A} \qquad (10)$$

$$z \leq A + (1-x)\bar{A} \qquad (11)$$

The delinearization can be described using 7 constraints as the following.

$$z \leq A_{Upper_{Bound}} \qquad (12)$$

$$-z \leq 0 \qquad (13)$$

$$z \leq A_{Upper_{Bound}}.x \qquad (14)$$

$$-z + A_{Lower_{Bound}}.x \leq 0 \qquad (15)$$

$$z - A - A_{Lower_{Bound}}.x \leq A_{Lower_{Bound}} \qquad (16)$$

$$-z + A + A_{Upper_{Bound}}.x \leq A_{Upper_{Bound}} \qquad (17)$$

$$z - A + A_{Upper_{Bound}}.x \leq A_{Upper_{Bound}} \qquad (18)$$

Delinearization is only needed for solvers that cant handle the product of two variables.

In the implementation, extra variables have been created for the problem to allow the previous formulation to be adapted to the Choco solver and to create a successful solution.

5. Solver Choice

Implementation is mainly distributed into three main sections:

1) Modeling the problem, solving, and solution iteration
2) Planogram drawing and motion, double buffered drawing mechanism
3) GUI interaction and information processing using multithreading operations.

6. Planogram Drawing Concepts and Method

The planogram is drawn as mentioned before using double-buffered images, and consists of several components and several layers for drawing where each component knows how to draw itself within the given boundaries and information, the sequence of drawing the planograms happens as the following:

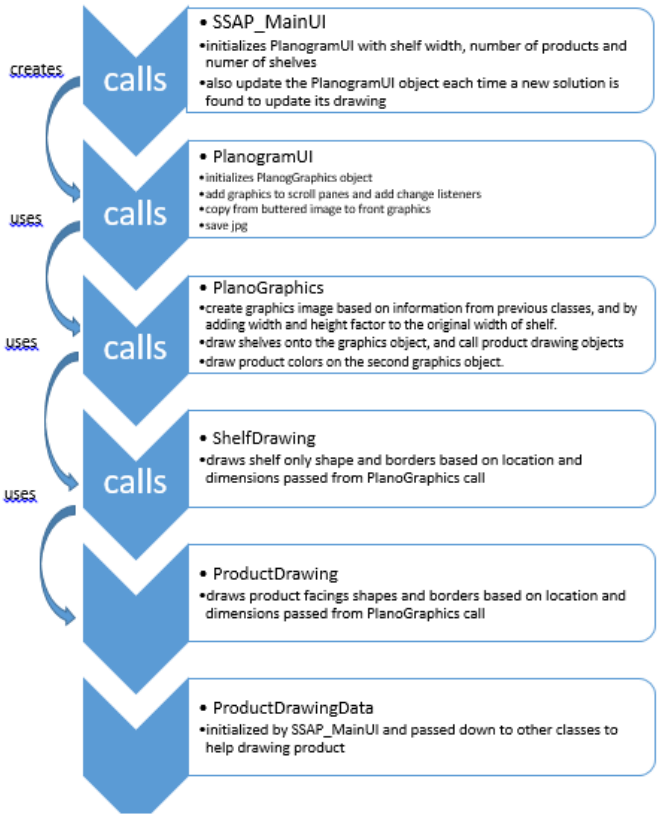

Figure 2: planogram drawing sequence

Figure 3: Planogram drawing
Figure 4: drawing planograms steps and method

As the planogram may contain hundreds of visual elements all to be drawn separately, the wise choice of drawing for smooth visualization is the use of double buffering images, where we draw on the background image then copy it one time to the front screen image when change occurs.

A. Solver working steps

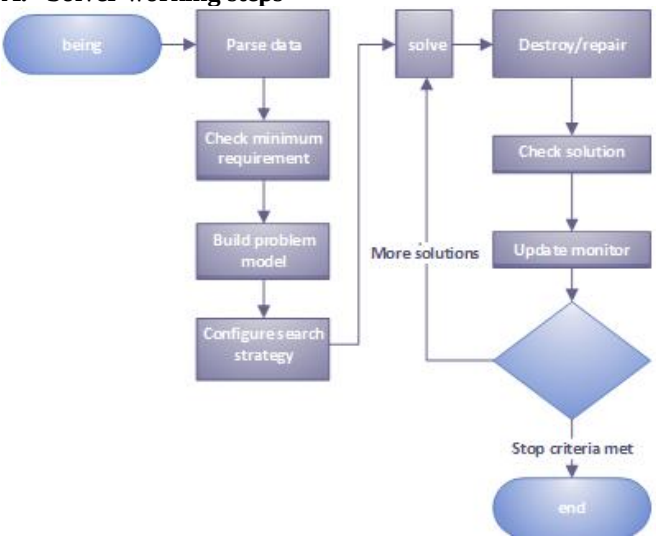

Figure 5: Solver working steps

Solver mainly works by defining variables and constraints over these variables, these variables are located within connected equations same as mentioned in a previous section about the mathematical model.

The solver running process consists of the following:

1) Calculate minimum space required to make sure that the problem can be solved, which means that there is enough space in the shelves that can allocate the minimum number of faces of all products, thus search should stop before it begins if this criteria is not met.
2) Creating the solver
3) Building the model:
   a) Create and initialize variables which includes:
   - Number of product facings variables (n), which is a bounded variable with the minimum and maximum number of facings obtained from the data set.
   - Product-shelf pair variables (n): which is bounded by 0 and the maximum number of shelves in data set.
   - Product-occupancy variables (n): which is a scaled version of the product facings variables, scaled by the facing length value (a scaled view)
   - Product-profit variables (n): which is a scaled view of the product facings variable with the scaling factor as being the profit value extracted from the data set (scaled view)
   - Product-profit-shelf-coeff variables (n) will be used for constraints purposes which is a none linear variable resulting from the product of product_profit and the shelf coefficient and finally used in the objective function, where the objective becomes the summation of the variables of this type.
   - Shelf-occupancy variables (n): how much a shelf is occupied, bounded by 0 and the length of the shelf.
   - Shelf_coffecient (n) : the coefficient of the shelf a product is lying onto.
   - Shelves Boolean variable (n X m) (which indicate whether a product I is located on shelf j), will also be used with constrains to impose the product_shelf rules, and that a product should only be located on one shelf.
   - Boolean variables (will contain a list of Boolean variables in the system), fattening the previous Boolean variables into a list.
   - Objective variable (total profit): which is the summation of the none linear variable created earlier of the product profit and the shelf coefficient.

   b) Constraints:
   - Constraints are modeled based on the formulas mentioned earlier, over the variables earlier.
   - shelf_prod_coeff_cons constraint: makes sure that shelfs_coeff[i] =coeffs[product_shelf[i]), where shelfs_coeff[i] and product_shekf[i] are variables, while coeffs is an int array.
   - prod_with_coeff_profit_cons constraint insures that product_profit_with_coeff[i] variable is the product of product_profit[i] and shelfs_coeff[i] variables.
   - c1 constraint insures that product_shelf[i] = j if shelves[i][j] is 1
   - c2 constraint insures that product_shelf[i] != j if shelves[i][j] is 0

- previous two constraints are connected via if then else constraint
- prod_on_one_shelf_cons constraint insures that summation of shelves[i] variables is 1, meaning that product lies on one shelf only.
- prod_shelf_occupany_cons constraint insures that prod_shelf_occupany[i] is the result of the product of product_occupancy[i] and shelves[i][j] variables
- shelf_cons insures that the summation of prod_shelf_occupany variables array equals to shelf_occupancy[j], thus the occupancy of single products on the same shelf are equal to the shelf occupancy.
- The final constraint is that the summation of product_profit_with_coeff variables equals to the objective variable.
- Constraints are carefully chosen and the slightest error or logical error will result in wrong results or in failure of finding a result.

c) Configuring search strategy

- Defines the search strategy, where we define the main strategy as the randomization of Boolean variables which indicate that the shelf-product ij pair presence, this value will be randomized to find solutions.
- Lexico UB attempts to randomize the occupancy numbers from larger numbers of the upper bound and then changing them to improve the solution which usually yields a good solution from the first attempt and faster search results.
- Lexico LB attempts to use the lower bounds and increase the occupancy numbers up, which usually yields a slow search
- We also add the stop criteria here (which is manual stop in our implementation if optimal is not found), we also define time limits for the solver to stop after some time of running equals to 3 times the number of products in seconds.

d) Listen to solver changes

- To provide log data to see what is happening.

e) Solve.

- The objective function is to Maximize the profit.
- Uses one of three methods:
  - PLNS : Propagation based (best method and fastest)
  - RLNS: Random based
  - ELNS: Explanation based.

7. Test Cases And Results

In order to test the solver, input data files have been created to simulate possible scenarios of shelves and products.

The application will have the option to load a customized external file other than those added to the resources and are part of the application.

It is important to use similar files as those attached in the same formatting and spacing and line intend, as strengthening the data file parser or data input is not our main concern in this project, we only need to present a way to test the different data files over our implementation)

Bellow you can see a simple text data file and how it is formatted, where the first line indicates how many shelves we have and how many products we need to place, with all info of minimum/maximum spacings allowed, profit, shelf part coefficients and product/shelf ids.

Table 3: sample input datafile formatting

```
8 8 180
   item1   class1     1  5  4  90 2.2
   item2   class1     2  10 4  90 3.1
   item3   class1     3  8  4  90 2.43
   item4   class2     4  40 3  90 3.66
   item5   class2     5  15 3  90 3.44
   item6   class3     6  5  6  90 2.2
   item7   class3     7  15 3  90 4.3
   item8   class4     8  20 4  90 2.6
   1  1  1.8  3
   2  1  1.8  2
   3  1  1.56 2
   4  2  1.56 2
   5  2  1.35 2
   6  2  1.35 1
   7  3  1.3  1
   8  3  1.3  2
```

The above data is combined into one text file of the following format (first line contains three numbers, indicating, number of product, number of shelves, length of shelf, respectively)

Then we read numbers of lines matching the first number, then number of lines matching second number, all data are separated by tabs or spaces (including first string of each line which starts with a tab or space), the parsing is straight forward for this file using Scanners.

Bellow we will see the results for 7 samples ranging from 4 shelves X 8 products up to 260 products to be placed over 90 shelves, with minimum shelf length of 180cm and maximum of 240cm.

Bellow we will display some graphics of the solutions given for the above 8x8 problem, where we have a main GUI to display all input/output data, another window to display the optimal plot, and a third window to display the planograms which show actual placement of products on shelves in a graphical manner.

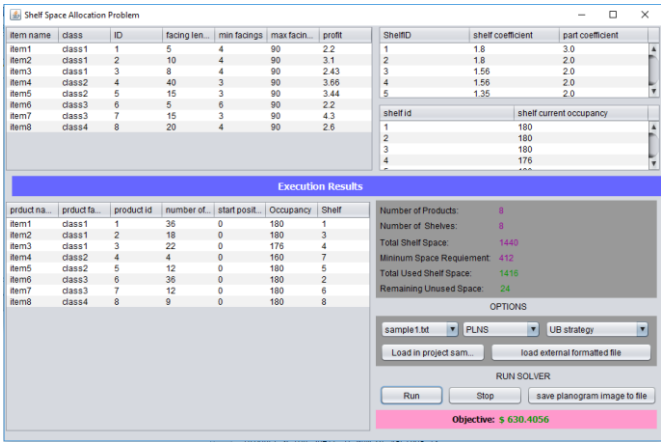

Figure 6: program interface

Program interfaces allows seeing products information in a table, shelf information in another, and the space allocation process and updated occupancy in a third table, while profit and other data is showed in a separate tab.

The interface allows choosing one of the 7 samples, or load a customized one, saving planogram images, stopping the solver before it completes and searching strategy.

Figure 7: Optimal Solution Convergence Plot for sample1

The plot above shows the improvement of the optimal solution or the total profit over solution number.

We continue to show the convergence plot for the 7 samples over a period of 6 times Number of Product using UB and PNLS search strategy. Below are the diagrams for 6 other samples

Figure 8: Sample2 optimal convergence plot

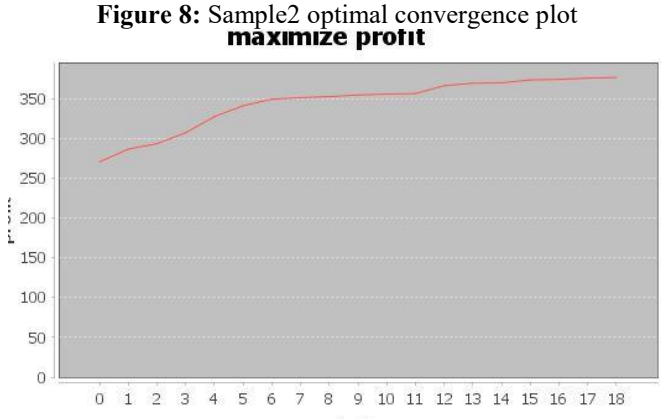

Figure 9: Sample3 objective function plot

Figure 10: Sample4 optimal convergence plot

Figure 11: Sample5 optimal convergence plot

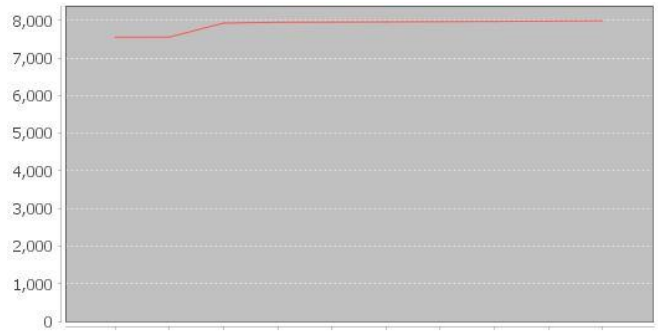

Figure 12: Sample6 optimal convergence plot

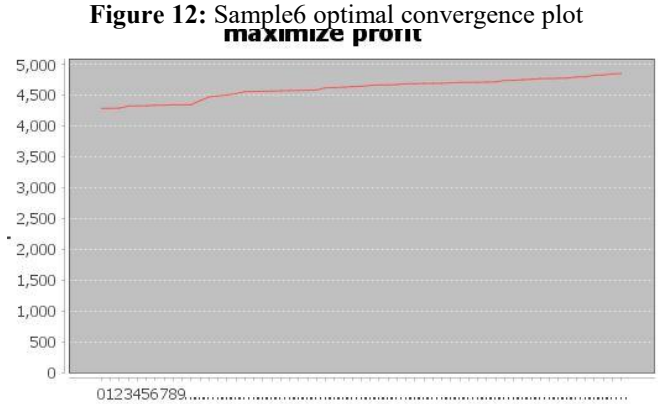

Figure 13: Sample7 optimal convergence plot

Following are the 7 planograms for the 7 sample files we tested.

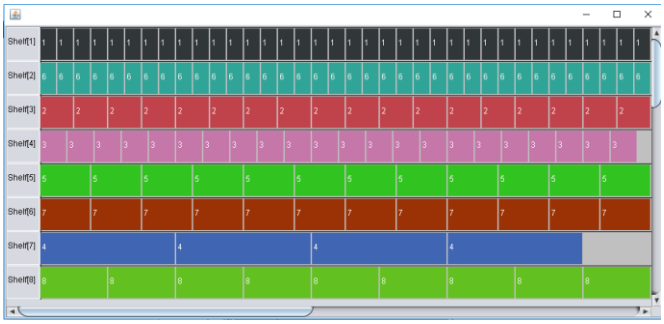

Figure 14: Sample1 shelf/product allocation planogram (8x8)

The planogram displays in a video like double buffered screen on a scrollable background and a canvas that can be saved by user for printing or other usages.

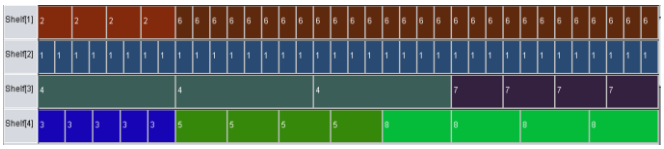

Figure 15: planogram for sample2.txt (8 products x 4 shelves)

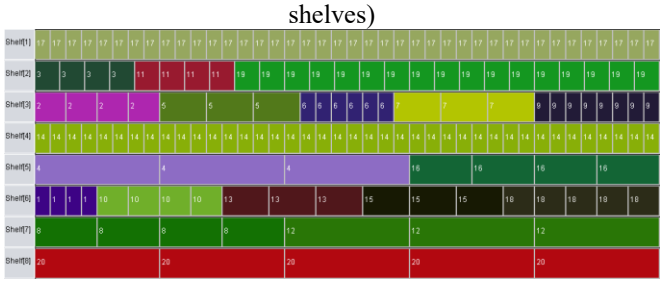

Figure 16: sample3 (20 Products X 8Shelves)

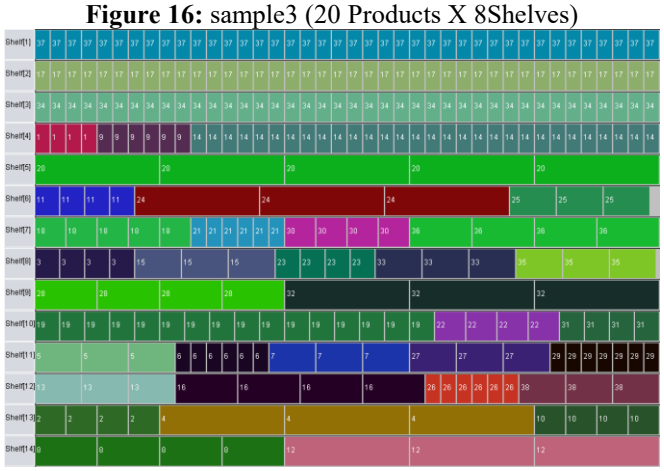

Figure 17: sample4 (38P x 14S)

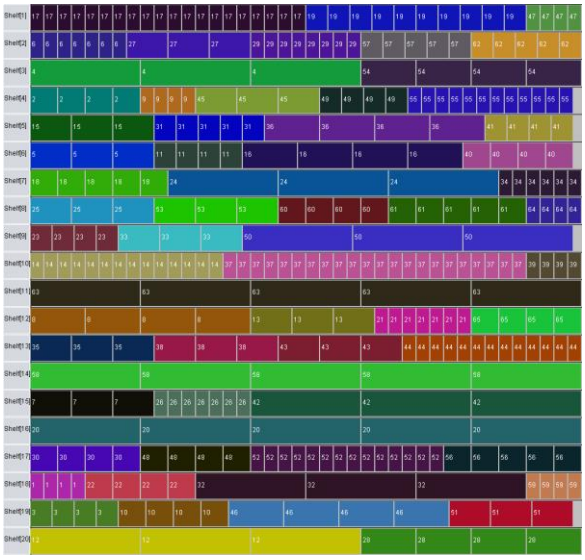

Figure 18: sample5 (65P x 20S) after 650seconds

Figure 19: sample6 (139px50s) after 834seconds

For the previous samples (1,2,3,4) a stop criteria of 3 times the number of products in seconds has been imposed, which is practical for such sample size. However for larger samples we make it 6 times to find better solutions.

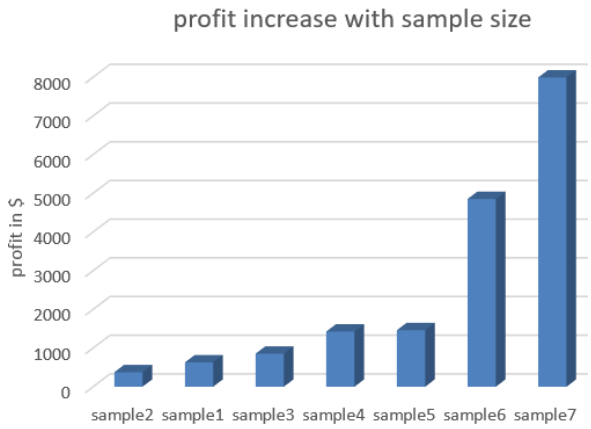

Figure 20: profit increase with the sample size increase considering stop criteria of 6xN. Products for all samples

On the other hand, we have tested some samples using the Upper bound search strategy and lower bound, and found that for our example: Upper bound is more effective for our application. Bellow we can see how LB investigates large number of solutions but still results with far lower profit result than UB using the same time window (444solutions vs 59 solutions) ($4002 vs $4853.8), as it explores the iterations randomly. LB starts by considering the minimum number of faces as the basis to search for valid solutions and starts incrementing the faces count gradually to fill the space and not exceed the shelf space. UB however starts by picking larger number of faces immediately.

Figure 21: sample6 using LB and PLNS over 834s

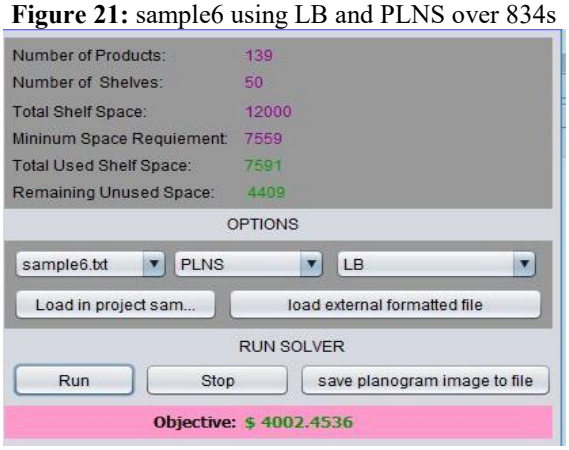

Figure 22:sample6 using LB and PLNS over 834s

```
- Incomplete search - Limit reached.
        Solver[Shelf Space Allocation Problem]
        Solutions: 444
        Maximize objective = 40024536,
        Building time : 0.767s
        Resolution time : 834.000s
        Nodes: 196,804 (236.0 n/s)
        Backtracks: 215,987
        Fails: 100,337
        Restarts: 1,004
        Variables: 49,537
        Constraints: 56,485
```

Figure 23:sample6 using LB and PLNS over 834s

Bellow are the UB results comparison for the same sample data file.

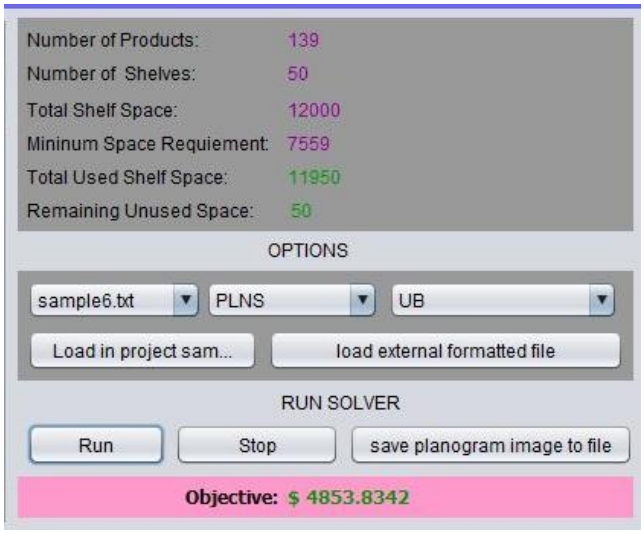

Figure 24:sample6 using UB and PLNS over 834s

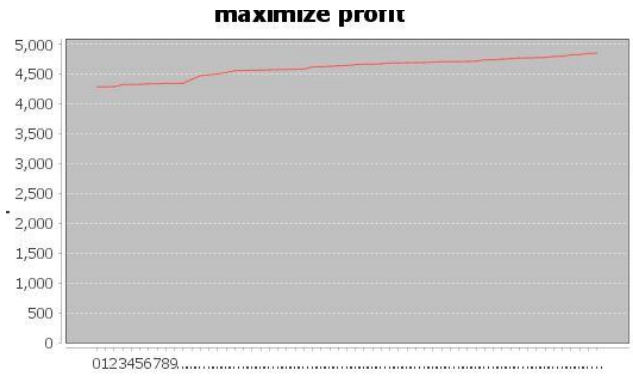

Figure 25:sample6 using UB and PLNS over 834s

```
- Incomplete search - Limit reached.
        Solver[Shelf Space Allocation Problem]
        Solutions: 59
        Maximize objective = 48538342,
        Building time : 0.692s
        Resolution time : 834.124s
        Nodes: 187,434 (224.7 n/s)
        Backtracks: 217,523
        Fails: 108,001
        Restarts: 1,081
        Variables: 49,537
        Constraints: 56,485
```

Figure 26:sample6 using UB and PLNS over 834s

8. Result Analysis Conclusion

We have presented a study for Shelf Space Allocation Problem, suggested a simple model to solve the problem in general, and proved that it can be solved efficiently using Largest Neighborhood Search Based methods.

The implementation presented a detailed GUI with a real time planogram frames to watch how the shelves are being sorted out and searched for better or optimal solution within the criteria's given.

The solver was able to iterate through hundreds of different solutions and improve the profit with each new solution tell the best solution was found based on time criteria dependent on the sample size.

The allocation utilized the maximum space allowed and places products where they yield the best profit and this can help stores and warehouses to use their spaces the best way possible to improve their profits or reduce expenses.

Upper Bound search strategy also finds a better solution much faster than Lower Bound strategy, and Propagation based solutions are also faster than Random based solutions.

9. Future Improvements

Our plan is to introduce family concepts into the algorithm and see how we can maintain proper aggregations according to predefined clustering rules.

We will also work on extending the problem to introduce topped up items and depth items as well to allow a more generalized algorithm that can be more flexible with the real world retail and warehouse item allocation.

10. Acknowledgment

This paper was created based upon a personal research on Shelf Space Allocation Problem algorithms, in *Higher Institute for Communication and Training, PAAET*

References

[1] P. Hansen and H. Heinsbroek. Product selection and space allocation in supermarkets. European journal of operational research, 3(6):474-484, 1979.
[2] A. Lim, B. Rodrigues, and X. Zhang. Metaheuristics with local search techniques for retail shelf-space optimization. Management science, 50(1):117-131, 2005.
[3] M. Yang and W. Chen. A study on shelf space allocation and management. International journal of production economics, 60-61:309{317, 1999.
[4] F. Zufryden. A dynamic programming approach for production selection and supermarket shelf-space allocation. Journal of the operational research society,
[5] F. Buttle. Retail space allocation. International Journal of Physics Distribution and Material Management, 14(4):3-23, 1984.
[6] J. Cairns. Allocate space for maximum pro_ts. Journal of Retailing, 39(2):41{55, 1963. 37(4):413-422, 1986.
[7] E. Anderson and H. Amato. A mathematical model for simultaneously determining the optimal brand collection and display area allocation. Operations
[8] X. Dreze, S. Hoch, and M. Purk. Shelf management and space elasticity. Journal of Retailing, 70(4):301-326, 1994. Research, 22(1):13{21, 1974.
[9] M. Yang. An e_cient algorithm to allocate shelf space. European journal of operational research, 131:107-118, 2001.
[10] Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighbourhood search. In Mark Wallace, editor, Principles and Practice of Constraint Programming - CP 2004, volume 3258 of Lecture Notes in Computer Science, pages 468{481. Springer, 2004. (also found in google books https://books.google.com/books?isbn=3540302018 *)*
[11] Explanation-Based Large Neighborhood Search, Charles Prud'homme, Xavier Lorca Narendra Jussien, 2013
[12] Principles and Practice of Constraint Programming - CP 2004: 10th International Conference, 2004
[13] Coelho, L. (2013, January 7). Linearization of the product of two variables. Retrieved February 25, 2017, from http://www.leandro-coelho.com/linearization-product-variables/
[14] Pisinger, D., & Røpke, S. (2010). Large Neighborhood Search. In M. Gendreau (Ed.), Handbook of Metaheuristics (2 ed., pp. 399-420). Springer. From http://orbit.dtu.dk/files/5293785/Pisinger.pdf

Author Profile

Dina Hamad Alghurair and Hedaya Ghanim Alshammar Both are Specialist Trainers in Higher Institute for Communication and Training, PAAET, Kuwait, having Masters of Science in Computer Engineering and Information Systems, Gulf University, Bahrain, both working together in education and research.