2010

Optimizing the Rearrangement Process in a Dedicated Warehouse

Hector J. Carlo
University of Puerto Rico - Mayaguez, hector.carlo@upr.edu

German E. Giraldo
University of Puerto Rico - Mayaguez

OPTIMIZING THE REARRANGEMENT PROCESS IN A DEDICATED WAREHOUSE

Hector J. Carlo

German E. Giraldo

Industrial Engineering Department, University of Puerto Rico – Mayagüez, Call Box 9000, Mayagüez, PR 00681

Abstract

Determining the optimal storage assignment for products in a dedicated warehouse has been addressed extensively in the Facility Logistics literature. However, the process of implementing a particular storage assignment given the current location of products has not received much attention in the existing literature. Typically, warehouses use downtime or overtime to remove products from their current location and move them to the suggested location. This work presents the *Rearrange-While-Working* (RWW) policy to optimize the process of rearranging a dedicated warehouse. The RWW policy seeks to relocate products in a warehouse from the initial arrangement to the optimal arrangement while serving a list of storages and retrievals. This study considers three scenarios: (1) when there is only one empty location in the warehouse and the material handling equipment (MHE) is idle (*i.e.* reshuffling policy); (2) when there is only one empty location in the warehouse under the RWW policy; (3) when there are multiple empty locations in the warehouse under the RWW policy. In the first case, the MHE can make any movement desired as it is idle. In the other cases, the movements correspond to a list of storages and retrievals that need to be served. In these cases it is assumed that products can only be moved when they are requested. After being used, they are returned to the warehouse. Several heuristics are presented for each scenario. The proposed heuristics are shown to perform satisfactorily in terms of solution quality and computational time.

1. Introduction and Literature Review

An important operational decision in warehouses is to determine the best storage location for each product in order to minimize the total material handling effort (or cost). This problem is known in the literature as the storage location assignment problem (SLAP). The SLAP can be classified according to the amount of information that is known about

the arrival and departure of the products stored in the warehouse: (1) item information (SLAP/II), (2) product information (SLAP/PI), or (3) no information (SLAP/NI) [1].

The SLAP/II problem assumes that the complete information about the arrival and departure times of individual items is known. A commonly used policy for the SLAP/II is the Duration-of-Stay (DOS) policy where items with the expected shortest visit are assigned to the most desirable locations [2].

In the SLAP/PI the information available is at the product level (instead of the item level, where items are instances of products). At the same time, products may be classified into product classes, typically according to physical characteristics or requirements. The SLAP/PI seeks to assign product classes to storage locations. After product classes are assigned to storage locations, the item location within its class is determined by simple rules (e.g. randomly). As described by Haussman *et al.* [3], the special case where the number of classes equals the number of products (n) is called *Dedicated (or Fixed Slot) Storage* as each product would have a specific set of storage locations for storage. If there is only one class, the storage policy is referred to as *Randomized or Floating Slot* as any item could go to any storage location. On the other hand, when the number of classes is between two and n-1, it is known Class-Based Storage. A commonly used policy for the SLAP/PI is to assign classes with small cube-per-order index (COI) [4] to the most desirable locations.

In the SLAP/NI, no information is available on the characteristics of the arriving items. Hence, only simple storage policies can be developed (*e.g.* Closest-Open-Location). For more details regarding existing studies addressing the SLAP problem the reader is referred to [1, 5].

The SLAP problem can be further classified as static or dynamic. In the static version of the problem the material flows are assumed constant over the planning horizon. On the other hand, the dynamic version continuously adjusts storage assignments based on material flows. Most of the existing literature focuses on the static version of the problem [1]. An interesting compromise was proposed by Christofides and Colloff [6] (*warehouse rearrangement*), Linn and Wysk [7-8] (*restoring policy*), and Muralidharan *et al.* [9] (*shuffling*). The basic idea in all of them is to relocate items during idle times in order to actualize the storage location assignment. Clearly, at some point the optimal SLAP configuration will change due to seasonality and life cycles of products. At this point, a new SLAP configuration is determined and idle times are used to update the warehouse configuration. Christofides and Colloff [6] proposed a two-stage algorithm to sequence item movements to minimize the material handling effort required to rearrange the products in a dedicated warehouse. Linn and Wysk [7-8] suggested a restoring policy for Automated Storage / Retrieval Systems (AS/RSs) using Class-Based Storage where idle times are used to move fast-moving items closer to the I/O. The authors do not provide details or computation results. Muralidharan *et al.* [9] formulated the problem of updating the warehouse configuration under Class-Based Storage for AS/RSs as a Precedence Constrained Selective Asymmetric Travelling Salesman Problem. Given the computational complexity of the problem, the authors proposed two heuristics: Shuffling with Nearest Neighbor Heuristic (SNN) and Shuffling with Insertion (SI). Based on

simulation results the authors conclude that using idle times to update the warehouse (AS/RS) configuration increases the AS/RS operating efficiency.

In this study we seek to update the configuration of a dedicated warehouse while serving orders. We do not rely on idle times to move products. Instead, when an item is retrieved, its storage location is changed in order to systematically update the warehouse arrangement. We assume that the initial (current) and final (*i.e.* optimal SLAP) storage location assignments are known. Hence, the problem is to move the items from an initial location to a final location by only moving those items that are required in the orders being served. For example, assume that an order includes an item that is stored in a pallet that will be retrieved by an AS/RS. The pallet is retrieved, and upon picking from the pallet, it is stored in a different location than the one held originally. The proposed *Rearrange-While-Working* (RWW) strategy assumes the problem is static (as we compute the final SLAP arrangement once and then implement it). Clearly, if one resolves the SLAP problem continuously the proposed strategy could solve the dynamic SLAP problem.

Although the work presented in this paper applies to any general warehouse with one material handler, we assume an AS/RS for convenience (for details on AS/RS the reader is referred to [10]). The remainder of this paper is organized as follows: Section 2 presents and solves the case where there is exactly one empty storage space in the AS/RS (warehouse) and the rearrangement will occur during an idle time (or overtime). Section 3 discusses the case where there is one empty location in the AS/RS and the rearrangement will occur while the AS/RS is serving orders. Section 4 presents the case where there are multiple empty locations and the rearrangement will occur while the AS/RS is serving orders. Lastly, Section 5 presents the conclusions.

2. One Empty Location - AS/RS Idle

In this Section we consider an AS/RS that serves a rack that has one empty location and while idle. The AS/RS will move the products in order to rearrange the rack from the current product arrangement to a final (SLAP optimal) arrangement. Notice this is a special case of the problem solved by Christofides and Colloff [6] where there is only one empty space.

Consider the product arrangement in the 3×3 racks shown in Figure 1. The left rack contains the initial (or current) product arrangement while the right rack contains the final arrangement. Let matrix A and B be associated with these racks such that $A(1,1) = 6$ and $B(1,1) = 1$. Notice that product 1, currently in position (1,2) needs to be moved to position (1,1) in the rack. The zero in $A(3,2)$ and $B(3,3)$ represent the empty location.

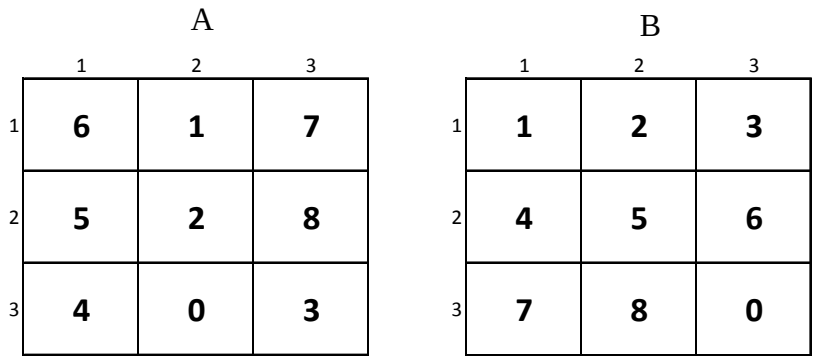

Figure 1: The Initial and Final Product Location in a Rack.

Any item can be moved to the empty location. Graphically, this movement would be equivalent to exchanging the location of the item with the empty location. Figure 2 presents the natural movement required to move items 6 and 8 to their final locations. The S/R machine moves from the I/O (assumed in the lower-left corner) to location (2,2), picks up item 8 and moves it to location (3,2), travel empty to location (1,1) to pick up item 6 and moves it to location (2,3). A similar process can be used to relocate the remaining items.

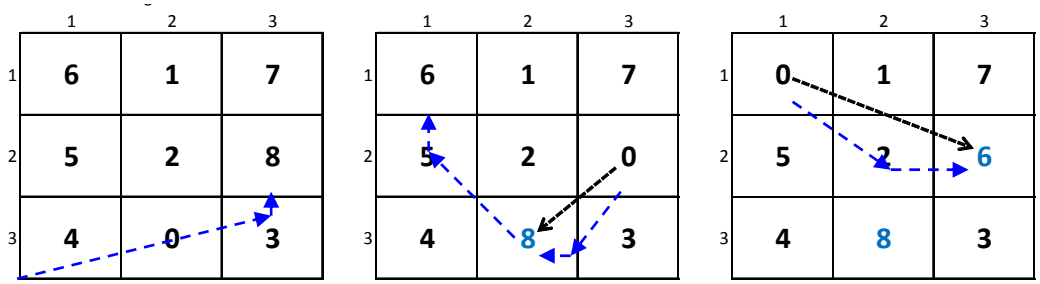

Figure 2: S/R Machine Path to Relocate Items 6 and 8.

2.1 Solving the Case with One Empty Location - AS/RS Idle

The problem of minimizing the material handling effort required to move from rack *A* to rack *B* during idle time (*i.e.* the S/R can move any item desired) can be found by using Dynamic Programming or by the algorithm in [6]. Unfortunately, as shown by Muralidharan *et al.* [9] this problem is *NP-Complete*. Hence, some heuristic approaches are considered to solve this problem.

The proposed heuristics only consider moving those items that are not in their final position. These items, if possible, will be moved to their final storage location without any intermediate steps. Every time an item is moved to its final storage location, the

problem is reduced until all items have reached their final storage location. If no items can be moved directly to their final location the heuristics decide how to proceed.

2.1.1 Heuristic 1 (H1): Evaluate all possible movements to the empty location

This heuristic will start by moving the item whose final destination is the initial location of the empty space. It continues until the location of the empty space coincides with the location of the empty space in the final arrangement. At this point the heuristic will branch to consider moving each item that is not in its final destination. In each branch the heuristic continues moving the item whose final destination is the current location of the empty space. If the location of the empty space once again coincides with its final location, another set of branches are generated. The solution will be the branch that yields the minimum cost.

2.1.2 Heuristic 2 (H2): Maximum Index

In this heuristic, instead of branching when the location of the empty location coincides with its final location, the item closest to the I/O in the horizontal axis is moved to the empty location. In case of a tie, choose the item closest to the I/O in the vertical axis. Evidently, this rule is arbitrary. However, it is expected to be faster than H1.

2.1.3 Heuristic 3 (H3): Closest Item

This heuristic is different from H2 in that the item to be selected (to move to the empty location if the empty location is in its final location) is the one closest to the empty space.

2.1.4 Heuristic 4 (H4): Closest to its Final Location

In this heuristic, the item selected (to move to the empty location if the empty location is in its final location) is the one that is closest to its final location (shortest distance from its destination).

2.2 Experimental Results and Analysis

To determine the best heuristic for the one empty location operating during idle time, a design of experiments was conducted. The factors were size of the rack (at levels 9, 100, 400) and the rack organization (*i.e*. percent of items not in their final location, at levels 0%, 50%, and 85%). Five replicates of each instance complete the experiment. It was assumed that the S/R machine takes 1 unit of time to move horizontally and vertically. The travel time for the S/R machine is assumed Chebyshev. The instances were randomly generated using VBA and coded in MATLAB 8.0. The experiments were run in a Dell Core 2 vPro 4 GB of memory and an AMD Turion 2.1 GHz processor. An ANOVA

considering including three co-variants (R^2 = 98.8%) suggest that H1 is statistically better (at a 99% level) than the other heuristics in terms of solution quality, followed by H3. However, in terms of solution time H1 is not able to solve large problems (400 locations). Hence, for practical purposes, H3 is the preferred heuristic.

3. One Empty Location and Using RWW

This Section explores the scenario where the rack has exactly one empty location (*n*-1 items) and uses the Rearrange-While-Working (RWW) policy. In other words, the moves performed by the AS/RS are limited to those in a list of items to be retrieved. Furthermore, the sequence in which the AS/RS can move the items is also dictated by the list. The sequence of the list is assumed known. Consider the initial and final arrangements in Figure 1. Now, assume that the AS/RS is expected to serve the following list of *k* orders {8,1,7,2,6,5, 3,1,7}. This means that the AS/RS must retrieve item 8, then item 1, and so on. We assume that the item will return to the rack as soon as it has been retrieved. Therefore, the AS/RS will retrieve item 8 (as it is the first in the list), we assume a picker will pick from the pallet and it will be returned to the rack. At this point, since there is only one open space, there are two options: return the item to its original position or move it to the empty space. After storing item 8, the process is repeated with the remainder items in the list. Notice that the approached followed could be described as a *block sequencing* approach.

3.1 Solving the Case with One Empty Location - RWW

Since the possible AS/RS movements are limited to items in the list, some items might never be requested. Also, since the position of the empty space depends on the specific order in which items are requested, the expected time to reach the final arrangement is very long. Clearly, the biggest challenge for this problem is to determine if a particular movement takes the configuration of the rack closer to the final configuration of the rack. Notice that in this scenario items are likely to be moved to a location other than its final location. Therefore, it is very likely that the rack configuration after serving one list of orders is neither the initial nor the final configuration. Hence, one needs to be able to quantify how close to the final rack configuration is a particular product arrangement. In other words, given two rack configurations one needs to determine which one is closer to the final configuration.

In this study we propose using H3 (from Section 2.1.3) to evaluate how close is a particular configuration to the final one. Clearly, by doing this we implicitly assume that the moves will be performed during idle time (as assumed by H3). If this is possible (*i.e.* use downtime to perform reshuffles), then the proposed Rearrange-While-Working policy is used in combination with the "shuffling" concept [6-9].

To explicitly enumerate the solution space one needs to consider 2^k combinations (for each of the *k* items in the order, consider moving to current location or moving to empty

space). Each of the 2^k combinations can then be evaluated using H3 to determine which is closer to the final configuration. (H3 is used instead of the optimal solution or H1 in order to be able to evaluate the 2^k combinations.) The best solution is the combination that minimizes the time to serve the k orders plus the time to take the resulting configuration to the final configuration using idle time. This solution is optimal if indeed the RWW policy will be combined with the optimal shuffling sequence during idle times. Henceforth, we use the term "optimal" to refer to the solution evaluated using H3, although we recognize it is not necessary optimal. The following sub-Sections present four heuristic methods to optimize the Rearrange-While-Working policy when there is only one empty location. In general, we propose to evaluate candidate solutions for Case 2 (one empty location under RWW) using the solution from Case 1 (1 empty location with ASRS idle).

3.1.1 Heuristic 5 (H5): Recognizing Repeated Items

This heuristic finds the first order to be repeated in the list and treats it as a sub-problem. All combinations of this sub-problem are evaluated using H3 and the best one is implemented. Then, the next order to be repeated after the sub-problem is treated as the subsequent sub-problem, until all decisions are made. In Figure 3 there are 13 items in the list of orders (to be served from left to right). The first item to be repeated is item 1. All possible combination of this sub-problem (that includes 7 items) is explored and evaluated. Then the next five items are identified for the second sub-problem, and so on.

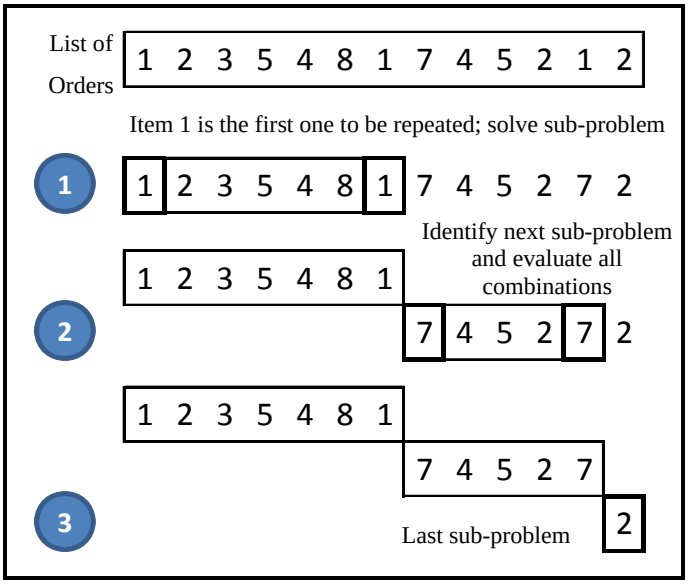

Figure 3: Concept of H5 - Recognizing Repeated Orders Heuristic.

3.1.2 Heuristic 6 (H6): Greedy

In this heuristic one evaluates both possibilities for the first order in the list and implements the better one. This process continues for the remaining items in the list of orders. Hence one only evaluates 2*k* options.

3.1.3 Other Heuristics

Case 2 can also be solved using Steepest Descent Heuristic (H7) and Genetic Algorithm (H8). In general, these heuristics can be applied by deciding if an item will return to its current location after it has been retrieved, or if it will be relocated to the empty space.

3.2 Experimental Results and Conclusions

To determine the best heuristic for the one empty location under the RWW policy, a design of experiments was conducted. The factors were size of the rack (at levels 9, 100, 400), rack organization (*i.e*. percent of items not in their final location, at levels 0%, 50%, and 85%), the size of the list of orders (at levels *k*=10, 25, 50), and item distribution in list (at levels uniform, where all items are equally likely to be in list, a case where some items have a 15% higher probability of being selected, and lastly a case where some items are 25% more likely to be selected). As before, five replicates of each instance complete the experiment and it was assumed that the S/R machine takes 1 unit of time to move horizontally and vertically according to Chebyshev distance. The experiments were generated in Visual Basic for Applications (VBA). The heuristics were coded in MATLAB 8.0 and the experiments were run in a Dell Core 2 vPro 4 GB of memory and an AMD Turion 2.1 GHz processor. An ANOVA (R^2 = 99.7%) suggest that H8 (GA) is statistically better (at a 99% level) than heuristics H6 (Gradient) and H7 (Steepest Descent). The solution time for the optimal solution and for Heuristic 5 (Recognizing Repeated Orders) exceeded 3 days for instances greater than 100 locations. Hence, we were not able to solve these instances. Clearly, these solution methodologies are not practical for industrial implementation. In the small instances, the best of 5 GA runs (H8) found the optimal solution in eleven of the twelve instances. For seven of the instances all GA runs found the optimal solution. For larger instances we were unable to identify the optimal solution. In terms of solution time, H8 (GA) solved all problems in less than one minute. In general, it is clear that H8 (GA) is the preferred solution methodology for the one empty location under the RWW policy. The second best heuristic, able to solve all instances of the problem, is the Steepest Descent heuristic (H7).

4. Multiple Empty Locations Using RWW

This Section explores the scenario where the rack has multiple empty locations and uses the rearrange-while-working policy. Theoretically, one should find the best empty

location for each item. However, given the complexity of the problem it is infeasible to consider all empty location. Instead, the proposed approach is to only consider returning the item to its original location or moving it to the open location closest to its final location. Clearly, this is analog to the case solved in Section 2 for which Genetic Algorithms (H8) proved to be a good solution methodology. Hence, we modify H8 to consider the closest empty location to the items final location and use it to solve the case with multiple empty locations. To compare our results, we also modify the second best heuristic for the one empty location under rearrange-while-working policy *(i.e.* Steepest Descent Heuristic, H7).

4.1 Experimental Results and Conclusions

To determine the best heuristic for the multiple empty locations under the RWW policy, a design of experiments was conducted. The factors were size of the rack (at levels 9, 100, 400), number of empty locations (5%, 20%, 40%), rack organization (at levels 0%, 50%, and 85%), the size of the list of orders (at levels 10, 25, 50), and item distribution in list (at levels uniform, 15% and 25%). Five replicates of each instance complete the experiment. It was assumed that the S/R machine takes 1 unit of time to move horizontally and vertically according to Chebyshev distance. The experiments were generated in VBA. The heuristics were coded in MATLAB 8.0 and the experiments were run in a Dell Core 2 vPro 4 GB of memory and an AMD Turion 2.1 GHz processor. An ANOVA suggest that H8 (GA) is statistically better (at a 99% level) than heuristic H7 (Steepest Descent) in terms of solution quality. In terms of runtimes, on average, H7 ran faster than H8.

5. Conclusions

This paper introduces the rearrange-while-working (RWW) policy, which seeks to organize a warehouse by relocating items as they are retrieved and re-stored in a warehouse according to a (production or order) list. The proposed RWW policy can be used in parallel with existing “shuffling” policies while the equipment is idle. Three cases were studied in this paper: (1) one empty location with idle equipment; (2) one empty location with RWW; and (3) multiple empty locations with RWW. A total of eight heuristics were tested for the three cases studied. In the one empty location with idle time the closes item heuristic (H3) was preferred. For the one empty location with RWW and the multiple locations with RWW the Genetic Algorithm approach (H8) was preferred. More details regarding the proposed RWW policy are available in Carlo and Giraldo [11].

The proposed RWW policy could be used to solve the static or dynamic SLAP problem. For the static SLAP problem, the optimal arrangement is computed and the RWW is used to implement it using a block scheduling approach. For the dynamic case,

after performing each movement the SLAP problem is solved and the proposed solution methodologies are used to determine the next movement.

References

[1] Gu, J., Goetschalckx, M., and McGinnis, L.F., "Research on Warehouse Operation: A Comprehensive Review." *European Journal of Operational Research*, 177, 1, 1-21 (2007).

[2] Goetschalckx, M., Ratliff, H.D., "Shared Storage Policies Based on the Duration Stay of Unit Loads," *Management Science*, 36, 9, 1120–1132 (1990).

[3] Hausman, W.H., Schwarz, L.B., and Graves, S.C., "Optimal Storage Assignment in Automatic Warehousing Systems," *Management Science*, 22, 6, 629-638 (1976).

[4] Heskett, J.L., "Cube-Per-Order Index: A Key to Warehouse Stock Location," *Transportation and Distribution Management*, 3, 27-31 (1963).

[5] Jeroen, P. and Van den Berg, J.P., "A literature survey on planning and control of warehousing systems," *IIE Transactions*; 31, 8, 751-762 (1999).

[6] Christofides, N. and Colloff, I., "The Rearrangement of Items in a Warehouse," Operations Research, 21, 2, 577-589 (1973).

[7] Linn, R.J., and Wysk, R.A., "An Expert System Framework for Automated Storage and Retrieval System Control, *Computers & Industrial Engineering*, 18, 1, 37-48 (1990a).

[8] Linn, R.J., and Wysk, R.A., "An Expert System Based Controller for an Automated Storage/Retrieval System, *International Journal of Production Research*, 28, 4, 735-756 (1990b).

[9] Muralidharan, B., Linn, R.J., and Pandit, R., "Shuffling heuristics for the storage location assignment in an AS/RS," *International Journal of Production Research*, 33, 6, 1661-1672 (1995).

[10] Roodbergen, K.J. and Vis, I.F.A., "A survey of literature on automated storage and retrieval systems," *European Journal of Operational Research*, 194, 2, 343–362 (2009).

[11] Carlo, H.J. and Giraldo, G.E., "Using Rearrange-While-Working Policy to Maintain Storage Location Assignments," working paper, University of Puerto Rico – Mayaguez.