Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Discrete Mathematics 39 (2012) 161–168

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

Variable neighborhood search for Multiple Level Warehouse Layout Problem [★]

Dragan Matić [a,1] Jozef Kratica [b,2] Vladimir Filipović [c,3] Djordje Dugošija [c,4]

[a] *Faculty of Science, University of Banjaluka, Banjaluka, Bosnia and Herzegovina*

[b] *Mathematical Institute, Serbian Academy of Sciences and Arts, Belgrade, Serbia*

[c] *Faculty of Mathematics, University of Belgrade, Belgrade, Serbia*

Abstract

In this paper we present a variable neighborhood search algorithm (VNS) for solving Multiple Level Warehouse Layout Problem (MLWLP). The algorithm deals with a specific representation of the solution, enabling the effective application of the shaking and local search procedures. System of neighborhoods changes the assignment ordering for an increasing number of items, while local search procedure tries to locally improve the solution by swapping the assignment ordering for pairs of items. Numerical experiments are performed on instances known in the literature. Computational results show that the proposed VNS achieves all optimal solutions for smaller instances, while for larger instances it finds rather better solutions than previously known method.

Keywords: Layout problem, Variable Neighborhood Search, Discrete Optimization

★ This work was supported by Serbian Ministry of Science under grants 174010 and 174033.

[1] Email: matic.dragan@gmail.com

[2] Email: jkratica@mi.sanu.ac.rs

[3] Email: vladaf@matf.bg.ac.rs

[4] Email: dugosija@matf.bg.ac.rs

doi:10.1016/j.endm.2012.10.022

1 Introduction

Multiple level warehouse layout problem (MLWLP) is to arrange the items into the cells placed on the multiple levels in a warehouse. The problem is NP hard and it is of the interest from both practical and academic points of view.

1.1 Problem description

Let us first briefly describe the assumptions for the MLWLP. The items in the warehouse need to be placed on the multiple levels. Each level is divided into the cells of the same dimension. There is only one I/O port on the ground and the only one elevator for transporting items from the ground to the levels. It is assumed that the elevator has enough capacity, which means that the vertical transportation operation is always available. The number of the cells on different levels may vary and for the cell k on the level l, the horizontal distance to the I/O port (denoted as D_{lk}) is known. Each item type j, $j = 1, ..., J$ has the following properties: monthly demand (Q_j), inventory volume(S_j), vertical unit transportation cost, i.e. the cost to move the unit of the item j between the ground and the level l (C^v_{jl}) and horizontal unit transportation cost, i.e. the cost to move one unit of the item 1m in horizontal distance (C^h_j). A cell may store more than one item type, but the total volume of the items placed in the single cell must not exceed the cell capacity. The solution of the MLWLP, i.e. the arrangement of the items, is feasible if each item type is assigned to the exactly one cell and no cell capacity is violated. For each item j, the transportation cost t_j is calculated by the formula $t_j = Q_j(D_{lk}C^v_{jl} + C^h_j)$ and the objective of the problem is to minimize the total transportation costs, i.e. $obj(MLWLP) = \min \sum_{j=1}^{J} t_j$.

1.2 Previous work

The first contributions for solving this problem dates in the eighties of the twentieth century, when Johnson ([2]) investigated the possibility of using the CRAFT heuristic for determining relative locations of facilities in a multi-floor building. The precise definition of the MLWLP, as well as the methods for solving the problem are introduced in [7]. The authors firstly defined all the constraints and the objective function for the problem, which enabled them to define an integer linear programming (ILP) model and to prove that the problem is NP hard. Also, they proposed a genetic algorithm (GA) for solving the problem. This GA approach uses the multiple storage areas in different levels of a warehouse and at each level and according to the inventory requirement, the same set of cells is used to store several item types. Two hybridizations of the proposed GA and path relinking strategy are presented in [8]. In the first hybridization, path relinking is used as an evolution operation in the GA, and the second hybridization applies path relinking when the GA was trapped in a local optimum. Another GA approach, as well as many experimental considerations are presented in [3]. The proposed GA uses an effective representation of the individuals and problem-specific objective function. The initial assignments and the good searching strategy are based on a principle that item types should be assigned to the "closer" cells. The paper also introduces a dozen of random generated problem instances and presents a plenty of experimental results. Other variants of the problem can also be found in the literature. In [9], the so called "adjacency constraints" are added to the starting problem defined in [7]. A new integer linear programming model is developed and several taboo techniques for solving the modified problem are presented. In [6], a multi-level layout problem is considered under the fuzzy environment. Based on the credibility measure, a chance-constrained programming model and a tabu search algorithm based on the fuzzy simulation are designed. A particle swarm optimization algorithm from [5] is used to manage multiple-level warehouse shelf configuration in order to minimize annual carrying costs.

2 VNS for the multiple warehouse layout problem

The VNS has been introduced by Mladenović and Hansen in [4]. The main strategy of the VNS approach is to investigate the quality of the solutions which belong to some neighborhood of the current best one. By a systematic change of the neighborhoods of the current solution, the algorithm is forced to leave suboptimal solutions and to change them over better ones. A detailed description of VNS is out of the paper's scope and can be found in [1].

2.1 Initialization and objective function

In order to make an appropriate representation of the solution, we have followed the following principles: (i) the items should be placed in the "closer" cells, i.e. cells for which the total transportation cost are of lower price; (ii) in most warehouses, there are some cells which are "closer" and "more suitable" for storage than others (probably they are placed closer to the I/O port, of somewhere in the warehouse where the elevator can quickly and comfortably access. So, the determining the layout depends on determining by which items these "closer" cells will be fulfilled.

The representation of the solution is realized as a permutation of the length J, where J is the number of items. This permutation proposes the ordering of the items which are arranged to the cells. Each item is arranged to the first closest, empty enough cell.

After the input of the data, for each item, the algorithm forms an array of cells, and for that specific item, the array is sorted according to the transportation cost from the I/O port to that cell. This may require additional CPU time, but we should notice that this action is performed only once, at the beginning of the algorithm. Calculating the objective function is as follows. The permutation proposes the ordering of the items which are arranged according that ordering. For each item, the closest empty enough cell is found and the current total transportation cost is increased by the corresponding total transportation cost for that item. Formally, if the solution f is represented by the permutation $\pi = (i_1, i_2, ..., i_J)$, than the objective function is calculated by the formula $obj(f) = \sum_{k=1}^{J} d(i_k, c_k)$, where for each item i_k, c_k is the closest empty enough cell and $d(i_k, c_k)$ is the transportation cost from I/O port to that cell for the specific item i_k.

It should be noticed that this representation can generate infeasible solutions. This can happen in cases when the arrangement based on the given permutation is not possible i.e. for some items there are no empty cells. In this case, the objective function is imparted an extremely large value. This enables the optimization process to change that permutation into another one, generating another, probably feasible solution. At the beginning, initial solution is created by a random permutation, i.e. the ordering of the item arrangement to the closest empty cell is initially random.

2.2 Neighborhoods and the shaking procedure

The shaking procedure creates a new solution x' which is based on the current solution x, $(x' \in N_k(x))$, in the following way: the algorithm completely randomly chooses and stirs some k elements of the current permutation. Thus, for a given k, the shaking procedure changes the ordering of some k items (each element of the permutation is related to one item). This approach arises from a practical consideration: If the elevator has a defined ordering of items which needs to arrange in the warehouse, a potential better solution can be found if that ordering is changed for some items. We should note that changing the schedule of the arrangement of the chosen k items can affect to more items, because the cells where "not changed items" previously has been placed, may now be full.

2.3 Local search

For the solution x' obtained by the shaking procedure, the local search is called. In each iteration of the local search, the algorithm is trying to improve the solution by swapping the positions of pairs of items in the permutation. The proposed local search procedure uses the first improvement strategy, which means that when an improvement is detected, the improvement is immediately applied and the local search continues. If, for each pair of items, swap produces the objective value greater or equal than the original one, the local search ends with no improvement.

Let us denote a new solution obtained by local search procedure as x''. Now, comparing the solution x'' and the incumbent, three cases can happen: (a) If $obj(x'') > obj(x)$, the search is repeated with the same x and the next neighborhood. (b)If $obj(x'') < obj(x)$, then the solution x gets the value x'', and the search is repeated with the same neighborhood. (c) If the objective values of the two solutions x and x'' are the same, then $x = x''$ is set with probability p_{move} and the algorithm continues the search with the same neighborhood. In other case, the search is repeated with the same x and the next neighborhood with probability $1 - p_{move}$.

3 Experimental results

This section contains computational results of the proposed VNS method. The VNS implementation is coded in C programming language. All the tests are carried out on single processor of the Intel Core 2 Quad Q9400 @2.66 GHz with 8 GB RAM. The test are performed on a subset of random generated instances from [3]. The whole set of instances is very large and contains five sets of small instances ($J \leq 40,\ L \leq 5$) and two sets of larger instances ($100 \leq J \leq 400$), generated with various values of the parameter α, which controls the percentage of the cells with same distance. For each problem instance, the VNS is run 20 times. Each run stops after 100 iterations.

Tables 1 and 2 provide results on 28 small instances with the control parameter $\alpha = 0.2$ and 12 large instances ($\alpha = 0.5$), respectively. The first three columns of both tables contain the number of the item types, levels and the parameter α, respectively. The rest of the Table 1 is organized as follows: the next two columns, named *opt* and t contain the optimal solution and the execution time needed to find it by the CPLEX linear solver; the next three columns contains the best GA solution from [3] (named $best_{sol}$) obtained in 20 runs, the average running time (t) used to reach the final GA solution for the

Table 1
Comparison of GA and VNS on the small instances

It	L	α	$CPLEX$		GA			VNS		
			opt	t[sec]	$best_{sol}$	$t[sec]$	$t_{tot}[sec]$	$best_{sol}$	$t[sec]$	$t_{tot}[sec]$
10	2	0.2	21062.300	<0.001	opt	0.001	0.391	opt	0.0016	0.0059
10	3	0.2	22324.209	<0.001	opt	0.001	0.412	opt	0.0019	0.0030
10	4	0.2	22389.975	0.02	opt	<0.001	0.4025	opt	0.0020	0.0030
10	5	0.2	22473.226	<0.001	opt	<0.001	0.413	opt	0.0034	0.0046
15	2	0.2	31144.125	<0.001	opt	0.001	0.5245	opt	0.0020	0.0069
15	3	0.2	28124.821	<0.001	opt	<0.001	0.534	opt	0.0023	0.0059
15	4	0.2	29878.762	0.01	opt	0.002	0.5115	opt	0.0023	0.0067
15	5	0.2	35417.046	<0.001	opt	0.002	0.4565	opt	0.0026	0.0050
20	2	0.2	35156.605	<0.001	opt	<0.001	0.6085	opt	0.0033	0.0102
20	3	0.2	35682.763	<0.001	opt	<0.001	0.569	opt	0.0037	0.0083
20	4	0.2	35470.801	<0.001	opt	0.007	0.5595	opt	0.0049	0.0121
20	5	0.2	49135.649	<0.001	opt	0.005	0.5735	opt	0.0044	0.0135
25	2	0.2	43962.444	0.01	opt	<0.001	0.7345	opt	0.0083	0.0240
25	3	0.2	41779.067	0.02	opt	<0.001	0.733	opt	0.0066	0.0172
25	4	0.2	40814.643	0.02	opt	0.035	0.7245	opt	0.0092	0.0262
25	5	0.2	57930.108	0.02	opt	0.022	0.6775	opt	0.0080	0.0241
30	2	0.2	58000.906	0.02	opt	<0.001	0.867	opt	0.0093	0.0324
30	3	0.2	51355.355	0.02	opt	<0.001	0.8515	opt	0.0118	0.0282
30	4	0.2	47189.872	0.03	opt	0.002	0.836	opt	0.0130	0.0396
30	5	0.2	49518.000	0.06	opt	0.011	0.8475	opt	0.0201	0.0422
35	2	0.2	81913.009	2.15	opt	0.198	1.143	opt	0.0538	0.1379
35	3	0.2	59878.908	<0.001	opt	0.001	0.9285	opt	0.0195	0.0477
35	4	0.2	57463.357	0.02	opt	0.012	0.9095	opt	0.0225	0.0699
35	5	0.2	60809.130	0.03	opt	0.141	1.042	opt	0.0401	0.0915
40	2	0.2	69045.672	0.05	69241.59	1.226	2.156	opt	0.1216	0.2796
40	3	0.2	67769.640	0.05	opt	0.045	1.069	opt	0.0416	0.0890
40	4	0.2	62302.340	0.06	opt	0.012	1.039	opt	0.0404	0.1311
40	5	0.2	64254.562	0.02	opt	0.002	1.004	opt	0.0213	0.0716

first time, and the average total running time for finishing the GA (t_{tot}). Last three columns contain the VNS results achieved after 100 iterations, presented in the same way as for the GA.

In order to further investigate the behavior of the proposed VNS, we executed the algorithm in running times comparable to the GA running times. So, in Table 2 after the three columns related to the GA (data presented in the same way as for the small instances), we have included the best and the average results achieved by the VNS after 20 runs, in the running times comparable the GA's. Last four columns in Table 2 contain the VNS results obtained after 100 iterations: best and average solutions (columns $best_{sol}$ and avg_{sol}), the average running time (t) used to reach the final VNS solution for the first time, and the average total running time for finishing the VNS (t_{tot}).

From Table 1 it is evident that VNS achieves all optimal solutions for small instances. The running times of the GA and VNS approaches are compara-

Table 2
Comparison of GA and VNS on the large instances

It	L	α	GA			$VNS(t_{tot} \approx t_{tot_{GA}})$		VNS (100 iter)			
			$best_{sol}$	$t[sec]$	$t_{tot}[sec]$	$best_{sol}$	avg_{sol}	$best_{sol}$	avg_{sol}	$t[sec]$	$t_{tot}[sec]$
100	2	0.5	886029.67	5.779	7.310	759727.695	762191.38	759064.698	759529.06	50.2947	69.6041
100	3	0.5	599825.17	4.231	6.531	475483.194	477306.086	475218.267	475796.268	26.7213	35.6193
100	4	0.5	545607.15	5.191	6.743	455571.03	457114.303	455076.906	455837.08	18.7241	25.9103
100	5	0.5	531832.02	5.883	6.992	485770.19	487025.973	485770.189	486199.738	12.2591	18.4651
150	2	0.5	2497848.77	9.472	11.764	2052680.65	2090527.346	1942212.688	1943601.314	356.279	508.8808
150	3	0.5	1736344.65	8.579	11.450	1418016.09	1435285.624	1370885.963	1372485.563	290.410	382.0970
150	4	0.5	1627720.06	9.120	11.328	1499406.01	1505348.219	1496670.682	1497771.676	99.8718	128.3030
150	5	0.5	958259.04	7.966	10.897	845843.39	849712.016	842966.48	844178.354	89.4548	109.4509
200	2	0.5	3407078.25	14.719	18.548	3045007.49	3156264.961	2653204.37	2655805.32	1531.61	2366.858
200	3	0.5	3044631.21	13.458	17.047	2742820.22	2793568.069	2530982.819	2533059.684	1029.95	1288.879
200	4	0.5	1736508.16	11.262	16.341	1438581.69	1499889.533	1301771.488	1304155.442	1016.13	1323.98
200	5	0.5	1649873.13	14.303	16.831	1430581.24	1453813.058	1371399.911	1373396.567	456.812	611.3388

ble, while for most instances, the CPLEX is faster than these two heuristics. Comparing the values from the columns named $best_{sol}$ from the Table 2, we can conclude that the VNS approach achieve better results. The existence of the local search procedure increases the computational time for the VNS and it is greater than of the GA approach, which does not implement any local search. At the other hand, slower executions of the VNS are justified by achieving better objective values. As we can see from the the seventh and the eighth columns of the Table 2, even for the shorter execution time, similar to the GA's, the VNS still achieves good results.

4 Conclusions

In this paper, a variable neighborhood search approach for solving the Multiple Warehouse Layout Problem is presented. A solution is represented as a permutation, which proposes the ordering of the item's arrangement into the closest empty cells. In the shaking procedure, the algorithm forms the system of neighborhoods by changing the ordering of an increasing number of items in permutation. This approach disseminates the search in a good direction and enables the effective application of the local search to the current solution candidate. In order to improve the solution in local search, the VNS algorithm swaps the positions of pairs of items inside the permutation.

Computational results show that the VNS approach is successful for solving MLWLP. The VNS achieves optimal solutions for small instances and provides better results for larger instances compared to the other method from the literature, even if the execution times are similar.

The directions for further researches could be investigating the possibilities of hybridization of the VNS approaches with other heuristics or exact methods as well as parallelization of the algorithm and running on multi-processor machines.

References

[1] Hansen, P., N. Mladenović, and J. A. Moreno-Pérez, *Variable neighbourhood search: algorithms and applications*, Ann. Oper. Res. **175** (2010), 367–407

[2] Johnson, R. V., *SPACECRAFT for multi-floor layout planning*, Manage. Sci., **28** (1982), 407–417.

[3] Matić D., V. Filipović, A. Savić, and Z. Stanimirović, *A Genetic Algorithm for Solving Multiple Warehouse Layout Problem*, Krag. J. Math. **35** (2011), 119–138.

[4] Mladenović N., and P. Hansen, *Variable neighbourhood search*, Comput. Oper. Res. **24** (1997) 1097–1100.

[5] Önüt, S., U.R. Tuzkaya, and B. Dogac, *A particle swarm optimization algorithm for the multiple-level warehouse layout design problem*, Comput. Ind. Eng. **54** (2008), 783–799.

[6] Yang, L., and Y. Feng, *Fuzzy multi-level warehouse layout problem: New model and algorithm*, J. Syst. Sci. Syst.Eng. **15** (2006), 493–503.

[7] Zhang G. Q., J. Xue, and K. K. Lai, *A class of genetic algorithms for multiple-level warehouse layout problems*, Int. J. of Prod. Res. **40** (2002), 731–744.

[8] Zhang G. Q., and K. K. Lai, *A Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem*, Eur. J. of Oper. Res. **169** (2006), 413–425.

[9] Zhang, G. Q., and K.K. Lai, *Tabu search approach for multi-level warehouse layout problem with adjacent constraints*, Eng. Optimiz. **42** (2010), 775–790.