Journal of Computations & Modelling, vol.3, no.4, 2013, 137-163
ISSN: 1792-7625 (print), 1792-8850 (online)
Scienpress Ltd, 2013

Army Rapid Fielding by Optimizing Order Picking Routes in Warehouses with Parallel Aisles – Implementation in a Real Case Study

Athanassios Nikolakopoulos[1]

Abstract

Army fielding is the process by which new equipment is distributed to soldiers either at dispersed units at homeland or at theatre of operations (J.D. Carter, SIEDS IEEE, 2007). Minimization of personnel, space and time resources are of utmost importance for improving the supply chain management system and the fielding of the necessary equipment at the theatre of operations. The equipment is stored in central warehouses where frequent and large orders of miscellaneous items are received each day. The items have to be picked from specific location within the warehoused then consolidated and finally dispatched. The most common type of warehouse is divided into parallel aisles between multistore storage racks. The order picking personnel walks or drives along the aisles to pick items from storage. They can change aisles at a number of cross aisles, which are usually located at the front and the back of the warehouse, but in some cases additional cross aisles are located at positions in between. In this paper, optimal

[1] National Technical University of Athens, School of Chemical Engineering, 9 Heroon Polytechneiou str., 15780 Zografou Campus Athens Greece.
E-mail: nikolako@mail.ntua.gr

Article Info: *Received* : July 1, 2013. *Revised* : August 30, 2013.
Published online : December 1, 2013.

routing of order pickers in parallel aisle warehouses is formulated as a Travelling Salesman Problem (TSP). A Threshold Accepting meta-heuristic algorithm is proposed next for the solution of the problem.

The proposed method is tested on a large set of randomly generated instances and is compared favourably to other methods that can be found in the literature. Moreover, the proposed method is implemented for the solution of a number of instances obtained during the operation of a real warehouse. The method produces better results compared to the method used in practice and it is shown that the warehouse operations can be significantly benefited.

Keywords: Military Logistics, Order picking; Threshold accepting; Metaheuristic; Routing; Warehouse Operations; Material Handling

1 Introduction

The seasonable replenishment of supplies and equipment during military operations is of outmost importance for accomplishing the desired outcome. At the heart of the military supply chain system lies the warehouse, where a multitude of material handling operations are performed daily. There, the scope for optimization is clear especially because time, space and personnel resources are limited in times where military operations take place. The central and most resource consuming activity of warehouse operations is the order-picking operation. This paper considers order-picking in warehouses with parallel aisles, which is the most common warehouse type used in practice. Order-picking is the process of collecting the items included in an order, from specified storage locations, and transporting them to the shipping area. In warehouses with parallel aisles, order-pickers drive along the aisles and pick requested items from storage locations. Possibilities for changing aisles are at the front and the back of the

warehouse and at a number of cross aisles in between. According to de Koster et. al. ([1]), these systems form the majority of picking systems in warehouses and they are named *low-level picker-to-parts* systems.

The activity of order picking is responsible for up to 55% of the total warehouse operating cost ([2]). The warehouse productivity can be improved significantly by reducing the order-picking travel time, which is an increasing function of the travel distance. Consequently, the travel distance is considered as a primary objective in optimization of the picking route. Except from precedence constraints imposed by the need for picking small and fragile items after heavy and big-volume items, the choice of an order picking tour is not affected by other picking activities.

The problem is identified as a special case of the Travelling Salesman Problem (TSP) ([3]), where given a number of locations and the cost of transiting from any location to any other location, the objective is to find the route that visits each location exactly once, at minimum cost. There is no polynomial-time algorithm for the solution of this NP-hard problem. There are however, dynamic programming algorithms for the case of optimizing order-picking routes in rectangular warehouses with two cross aisles ([4]) and three cross aisles ([5]), in running time which are linear with respect to the number of aisles and the number of pick locations. Extensions of these algorithms to more cross aisles are non-trivial.

In real warehouse practice, a heuristic solution to the problem is most often obtained by applying the Traversal or else S-shape heuristic strategy in which aisles are served in order and if an aisle contains at least one pick the picker crosses the entire length ending at the opposite side of the warehouse.

Hall ([6]) evaluated and compared heuristic strategies for routing a manual picker through warehouses with two cross aisles. Performance comparisons between optimal routing and heuristics for this type of warehouse are also given by De Koster and Van der Poort ([7]) and Petersen ([8]). These methods were

modified by Roodbergen and De Koster ([9]) and implemented for the solution of order-picking routing problems in warehouses with more than two cross aisles. Makris and Giakoumakis ([10]) presented a route improvement method using the Lin and Kernighan's ([11]) k-opt methodology. Daniels et al. ([12]) considered the case where units of the same item can be stored in multiple locations so that order picking requires choosing a subset of the locations that store an item to collect the required quantity. Thus, the total travelled distance is affected both by the assignment of inventory to an order and the sequence in which the locations have to be visited. For the solution of the aforementioned problem, Daniels et al. proposed a modified standard nearest neighbour ([13]) a modified shortest arc ([14]) heuristic and a Tabu search algorithm.

The present work implements a metaheuristic algorithm for obtaining good results for practical problems in realistic computational times. The Floyd's ([15]) shortest path algorithm is used for the calculation of the minimum distances between all pairs of picking locations and a modified Threshold Accepting method ([16]), with an intensified Local Search procedure is proposed for the solution of the problem of determining the best picking routes in warehouses with multiple cross aisles.

The optimal picking routes produced by the cutting-plane method of Dantzig et al. ([17]) are used as benchmarks for the performance analysis of the metaheuristic method proposed in this paper and other heuristic methods found in the literature. The proposed method is also tested by providing solutions to instances that emerged during the operation of a real warehouse. The proposed method compares preferably to the method followed in practice.

The rest of the paper is structured as follows: The next section defines the problem. Section 3 is devoted to the description of the proposed metaheuristic algorithm. Section 4 reports computational results on benchmark instances and on instances from a case study. The paper ends with conclusions in Section 5.

2 Warehouse Description

The layout of warehouses considered in this work, consists of a number of back to back shelves and a number of parallel aisles between them. We assume that the aisles can be traversed in both directions. Aisles communicate through cross aisles at the front end, the rear end and probably through one or more cross aisles in between. The aisles are narrow enough to permit simultaneous picking of items from both sides of a pick aisle. Picked orders have to be deposited at the depot, where the picker is supplied with an empty pallet or container for the collection of the items of the next order. Without loss of generality, the depot is located at the beginning of the first pick aisle.

An example of a warehouse layout with 6 aisles and 4 cross aisles is given in Figure 1(a) where black boxes indicate the locations from where the items of an order have to be picked. By associating each crossing of aisles and cross aisles with a node, the set

$$U=\{u_1,u_2,...,u_l\}$$

of the crossing nodes is constructed. Additionally, the depot and each point in the aisles next to the picking locations are also represented by nodes, which formulate the set

$$V=\{v_1,v_2,...,v_n\}$$

of nodes. Then a graph $G(W, A)$ is defined, where

$$W= U \cup V =\{w_1,w_2,...,w_m\}$$

is the set of all nodes and A the set of all undirected links in G, i.e. all arcs that connect adjacent nodes. The equivalent graph representation of the warehouse layout and order presented in Figure 1(a) is given in Figure 1(b), where the nodes in set V are represented by thicker circles.

Given that the order picker has to collect a number of items in specified quantities at known locations, the question is in what sequence the order picker should visit these locations in order to minimize the total travelling distance. As

already mentioned, the problem can be formulated as a variation of the TSP. However, the route which is produced by the algorithm does not need to visit all nodes in set *U*, in contrast to the nodes in the aisles next to the pick locations, which are all visited. Moreover, the order-picking route may visit any node in sets *U* and *V* more than once. Next section describes the proposed method for the solution of the specified problem.

3 Description of the Proposed Method

3.1 Formulation of the TSP Problem

The TSP is formulated by constructing a symmetric matrix containing the distances between all pairs of nodes in set *V*, i.e. pairs of nodes that represent picking locations. The distance between any pair of nodes of the graph is calculated as the shortest path connecting these nodes. To this end, the Floyd's algorithm ([15]) is utilized, which as showed by Dreyfus ([18]) is about 50% faster than the application of the Dijkstra's algorithm ([19]) when applied for every pair of nodes in *G*.

The distance unit is defined under the assumption that all storage locations have equal square-shaped base which is also equal to the square-area where two crossing aisles overlap. The length of each side of the square is used as the distance unit and will be called from now on as *lsl*. For any two adjacent nodes in graph G, assignment of the distance is straightforward. For example in Figure 1(b) the distance between the adjacent nodes v_4 and v_5 is 4 *lsl*s and the distance between the adjacent crossing nodes u_{16} and u_{23} is 3 *lsl*s. For all other pairs of nodes, i.e. non-adjacent nodes distances are assigned by calculating the respective shortest paths. The Floyd's algorithm is used for this task, which consist of the following three steps:

Initialization: The $m \times m$ distance square matrix

$$\mathbf{D} = \begin{bmatrix} d(1,1) & d(1,2) & \dots & d(1,m) \\ d(2,1) & d(2,2) & \dots & d(2,m) \\ \vdots & \vdots & \vdots & \vdots \\ d(m,1) & d(m,2) & \dots & d(m,m) \end{bmatrix},$$

where $m = |W|$ is initialized so that $d(i,i) = 0 \ \forall \ i = 1, \dots, m$, and $d(i,j) = \infty$ whenever arc $(w_i, w_j) \notin A$.

Set $k = 0$.

Step 1. $k = k + 1$

Step 2. For all $i \neq k$ and $d(i,k) \neq \infty$

For all $j \neq k$ and $d(k,j) \neq \infty$

$$d(i,j) = \min[d(i,j), d(i,k) + d(k,j)] \quad (1)$$

Step 3. If $k = m$, **D** is contains the shortest paths between all pairs of nodes in *W*. *Stop*.

Else go to *Step* 1.

After the calculation of matrix **D**, the $n \times n$ distance square matrix

$$\mathbf{P} = \begin{bmatrix} p(1,1) & p(1,2) & \dots & p(1,n) \\ p(2,1) & p(2,2) & \dots & p(2,n) \\ \vdots & \vdots & \vdots & \vdots \\ p(n,1) & p(n,2) & \dots & p(n,n) \end{bmatrix},$$

where $n = |V|$ is easily obtained by keeping only the rows and columns of matrix **D** that correspond to nodes in $|V|$. A TSP is now clearly formulated: Assuming a complete undirected graph of $|V|$ vertices with distances given by matrix **P,** find the Hamilton cycle of minimum length. This problem is solved using the meta-heuristic algorithm that is described in the next subsection.

However, the sequence of nodes which is the result of the TSP does not suffice to describe completely the optimal route for the order picker. It may be possible that two subsequent nodes in the optimal solution are not actually

adjacent in graph *G*. For these cases, a record containing the intermediate nodes should be available. This is attained by the bookkeeping mechanism suggested by Hu ([20]), according to which an additional $m \times m$ matrix

$$\mathbf{H} = \begin{bmatrix} h(1,1) & h(1,2) & \ldots & h(1,m) \\ h(2,1) & h(2,2) & \ldots & h(2,m) \\ \vdots & \vdots & \vdots & \vdots \\ h(m,1) & h(m,2) & \ldots & h(m,m) \end{bmatrix}$$

is constructed. The element $h(i,j)$ is the node just before w_j on the shortest path from w_i to w_j, in graph *G*. Calculation of matrix **H** can be included in Floyd's algorithm, by initializing the matrix (setting $h(i,j) = w_i$, $i, j = 1, \ldots, m$) and by adding the following step right after equation (1):

If $d(i,k) + d(k,i) < d(i,j)$, then $h(i,j) = h(k,j)$.

If two subsequent nodes, say u_i and u_j in the optimal solution of the TSP are not actually adjacent in graph *G*, the solution is augmented by including the realizable optimal path between the nodes. In particular, if w_i and w_j are the respective nodes in graph *G*, the realizable shortest path can be obtained immediately from matrix **H** as follows:

$$w_i, w_r, \ldots, w_c, w_b, w_a, w_j$$

where $w_\alpha = h(i,j)$, $w_b = h(i,a)$, $w_c = h(i,b)$, …, $w_i = h(i,r)$.

3.2 Description of the algorithm

This subsection describes the meta-heuristic algorithm which was used to solve the TSP that was formulated in subsection 3.1.

3.2.1 The Threshold Accepting (TA) Method Combined with Gradually Reinforced Local Search

Assuming that $\boldsymbol{X}^*$ is the set of all feasible solutions (sequences of nodes) of the problem, TA starts with an element $\mathbf{x}_0 \in \boldsymbol{X}^*$, which may be randomly chosen. Then, the method proceeds in an iterative manner. In each iteration the algorithm decides if the current solution $\mathbf{x}_c$ will be replaced by a new one $\mathbf{x}_{new}$. The new candidate is chosen (by use of Local Search moves) as a small perturbation of the current solution or-speaking in mathematical terms-in a given neighborhood of the current solution $\mathbf{x}_c$. The value of the objective function is calculated for the new candidate and the results are compared:

$$\Delta f = f(\mathbf{x}_{new}) - f(\mathbf{x}_c).$$

In the TA method, the new element $\mathbf{x}_{new}$ is accepted as the current solution if and only if $\Delta f \leq T$ for the current threshold value T. The proposed algorithm uses an adaptable string of prospective thresholds named **TS**. The values of thresholds in **TS** are sorted in an order of increasing values. For all the test cases that will be presented in this work, **TS** is initially formulated as

$$\{0,1,2,3,\ldots,T_{max}\},$$

i.e. it contains zero and all the positive integer numbers up to the maximum threshold T_{max} . Although the maximum threshold value can be selected by trial and error, experimentation with many test cases showed that a good choice is to select T_{max} as half of the maximum distance between any two nodes i.e.

$$T_{\max} = \frac{\max_{i,j}\left[p(i,j)\right]}{2},\ i = 1,\ldots,n,\quad j = 1,\ldots,n \tag{3}$$

The last threshold value in the string T_{max} is considered as the current threshold T. The decision of updating the value of the current threshold T is taken within an iterative procedure, which starts by initializing the parameters *Ls* and *Im*. The parameter *Ls* indicates the number of iterations devoted to Local Search and is initially set to Ls_o, which depends on the magnitude of the problem (in all test

cases we used $Ls_o = n$). The binary parameter *Im* indicates the improvement state, that is whether there has been an improvement on the objective function value and is initially set to 0. When the objective function value is improved ($\Delta f < 0$), the value of *Im* becomes 1 and the current threshold *T* (the last element of **TS)** is set to 0, meaning that only improvements in the objective function are accepted. The parameter *Im* remains equal to 1 for the next $Ls = Ls * a_{ls}$ iterations, where $a_{ls} \in (1, 1.5]$. This way, the algorithm is forced to conduct local search when an improvement in the value of the objective function occurs. Furthermore, as the number of improvements increases, local search is becoming more insistent, since *Ls* is continuously increasing. If $0 \leq \Delta f \leq T$, then the last and largest value in **TS** is replaced by Δf and the values in the string are sorted in an ascending order. If $\Delta f > T$ the algorithm proceeds to the next move and all the parameters remain to their current values. The algorithm terminates when all the elements of **TS** become equal to 0. Since this usually requires a large number of iterations, practically we terminate the algorithm at an earlier stage, which is defined by a maximum number of iterations. The flow diagram of the algorithm is shown in Figure 2.

The above method initializes a group of potential thresholds and changes the current threshold, based on the progress of the algorithm. Thus the functional features of the algorithm are problem-independent, which is a great advantage compared to other threshold accepting algorithms ([16], [21] and [22]).

3.2.2 Representation of the Solution and the Insertion Method

We consider as current solution a vector $\mathbf{x} \in \boldsymbol{X}^*$, where $\boldsymbol{X}^*$ is the set of all possible permutations of the nodes. In a specific permutation **x**, the position of each node denotes the order in which it will be inserted in the schedule. Thus, for the rest of the paper, the name *Insertion Order* will be used for the sequence **x**, while the corresponding schedule will be denoted by **S**. This way the searching operations of the algorithm are imposed on the unscheduled string **x, x** in turn is

translated into a schedule **S** and the solution **x** is surveyed right after the value of the respective cost has been calculated.

Let us assume an *Insertion Order* **x** = (v_1,…,v_n), where v_k is a node in **x** and n is the total number of nodes. The nodes of **x** will be inserted one by one into the current partial path, until they formulate the final schedule **S** = (s_1, …, s_n). According to the insertion scheme, the incomplete schedule **S´** = (s_1,…, s_k) where the first k nodes (v_1, …, v_k) of the *Insertion Order* vector **x** have been positioned, is augmented by placing the node $v_l = v_{k+1}$ between the nodes s_r and s_{r+1} in **S** that correspond to the minimum value of

$$R_i = p(s_{i,}v_l) + p(v_l, s_{i+1}) - p(s_{i,}s_{i+1}) \tag{4}$$

i.e.:

$$r = \arg\min_i \left[p(s_i, v_l) + p(v_l, s_{i+1}) - p(s_i, s_{i+1}) \right], \qquad i = 1, ..., k-1 \tag{5}$$

where $p(i, j)$ is the distance or processing time from node i to node j and arg min stands for the argument of the minimum, i.e. the value of i for which the value of formula (5) attains its minimum value. The above procedure is repeated until $k = n$, where the schedule is complete and feasible. Figure 3 presents the flow diagram of the Insertion Method described above. To follow the iterative procedure of the TA method, the algorithm needs to start with a feasible initial solution. The trivial sequence (v_1, v_2, …, v_n) is utilized for all implementations.

3.2.3 Local Search Moves

The proposed algorithm is based on an *Insertion Order* **x**. As it was discussed earlier, the actual schedule **S** results from **x**, by the use of node insertion method. Local Search moves do not intervene on the schedule **S**, but on the *Insertion Order* **x**. The new *Insertion Order* **x´** is located in the neighborhood of **x**. However the new offspring schedule **S´** is not always a neighbor of **S**. This occurs because the insertion method leads potentially to considerably altered schedules.

We utilized six different types of Local Search moves. Given a primary *Insertion Order* $\mathbf{x} = (v_1, \ldots, v_n)$, the six moves are described as follows:

1) Choose randomly two nodes from **x**, say v_k and v_l ($v_k, v_l \in \{v_1,\ldots,v_n\}$and $v_k \neq v_l$) and exchange their positions in the *Insertion Order* string. If v_k precedes v_l then $\mathbf{x} = (v_1,\ldots, v_{k-1},v_k,v_{k+1},\ldots, v_{l-1},v_l,v_{l+1},\ldots,v_n)$ becomes $\mathbf{x}' = (v_1,\ldots, v_{k-1},v_l,v_{k+1},\ldots, v_{l-1}, v_k, v_{l+1},\ldots,v_n)$.

2) Choose randomly two nodes from **x**, say v_k and v_l ($v_k, v_l \in \{v_1,\ldots,v_n\}$and $v_k \neq v_l$) and invert the sequence of nodes between them including v_k and v_l. Using this move $\mathbf{x} = (v_1, \ldots, v_{k-1},v_k,v_{k+1},v_{k+2},\ldots,v_{l-2},v_{l-1},v_l, v_{l+1},\ldots,v_n)$ becomes $\mathbf{x}' = (v_1,\ldots, v_{k-1},v_l,v_{l-1},v_{l-2},\ldots,v_{k+2},v_{k+1},v_k, v_{l+1},\ldots,v_n)$.

3) Choose randomly two nodes from **x**, say v_k and v_l ($v_k, v_l \in \{v_1,\ldots,v_n\}$and $v_k \neq v_l$) and move the sequence of nodes $(v_k,v_{k+1},\ldots, v_{l-1},v_l)$ after another randomly chosen node, say $v_m \in \{v_1,\ldots,v_n\}/\{ v_k,v_{k+1},\ldots, v_{l-1},v_l\}$. For example, if v_k precedes v_l and v_m precedes v_k then $\mathbf{x} = (v_1,\ldots,v_m,v_{m+1},\ldots,v_{k-1},v_k,v_{k+1},\ldots,v_{l-1},v_l,v_{l+1},\ldots,v_n)$ becomes $\mathbf{x}' = (v_1,\ldots,v_m,v_k,v_{k+1},\ldots,v_{l-1},v_l, v_{m+1},\ldots,v_{k-1},v_{l+1},\ldots v_n)$.

4) Choose only one node, say $v_l \in \{v_1,\ldots,v_n\}$ and transfer it to the first position in the *Insertion Order* string. So $\mathbf{x} = (v_1,\ldots,v_{l-1},v_l,v_{l+1}, \ldots,v_n)$ becomes $\mathbf{x}' = (v_l,v_1,\ldots v_{l-1},v_{l+1},\ldots v_n)$.

5) Divide the *Insertion Order* **x** into four, not necessarily of the same size, sequential groups of nodes, say $\mathbf{x} = (g_i, g_{i+1}, g_h, g_{h+1})$. Altering the order of the groups, the new sequence that is produced is: $\mathbf{x}' = (g_i, g_{h+1}, g_h, g_{i+1})$. This operator can be thought of as a variation of operator (3) which also divides *Insertion Order* **x** into four segments, but the new sequence that is produced is $\mathbf{x}' = (g_i, g_h, g_{i+1}, g_{h+1})$.

6) Divide the *Insertion Order* **x** into six, not necessarily of the same size, sequential groups of nodes, say $\mathbf{x} = (g_i, g_{i+1}, g_j, g_{j+1}, g_h, g_{h+1})$. Altering the order of the groups, the new sequence that is produced is $\mathbf{x}' = (g_i, g_{j+1}, g_h, g_{i+1}, g_j, g_{h+1})$ or $\mathbf{x}' = (g_i, g_h, g_{j+1}, g_{i+1}, g_j, g_{h+1})$ with equal probability.

Remark 1: Moves 5 and 6 are similar to the ones used by Cirasella et al. [23] and Gambardella and Dorigo [24] and originate from the *k-exchange* moves when *k* is equal to 2 and 3 respectively.

Remark 2: At each iteration of the algorithm, only one of the six moves is selected to be applied to the current string $\mathbf{x}_c$ in order to produce the new string $\mathbf{x}_{new}$. The probability of selection is the same for all six moves.

4 Computational Results

4.1 Comparative Results for Randomly Generated Instances

The MatLab programming language was used for implementing the algorithm and a 3.2 MHz Pentium 4 processor was used for all computational experiments.

For our simulation experiments we assume warehouses with 18 aisles. We consider 2, 3, 4 and 5 cross aisles, and in the resulting layouts the aisles contain 50, 48, 48 and 44 stock locations respectively on either side of every aisle. These values are representative of small warehouses or zones ([1]) that comprise large order picking areas and are assigned to specific order pickers. Typical numbers of items requested in a single order for most *low-level picker-to-parts* systems are 20, 50 and 100 items/order. Combinations of the aforementioned values sum up to 12 configurations. For each configuration, we generate 10 random orders thus producing 120 instances in total. We assume a *random storage assignment* where storage locations are selected randomly among all eligible empty locations with equal probability [8], which results to a uniform and independent distribution of the locations.

The optimal solutions of the corresponding TSPs were obtained using the cutting plane algorithm introduced by Dantzig ([17]). It should be noted however, that this exact algorithm has unpredictable large computational times, which is an

undesirable property for practical implementations. Some problems with up to 50 nodes may require more than 15 hours in a typical PC configuration running Matlab depending on the structure of the distance matrix of the nodes. The computations are even more time consuming for problems with 50 to 100 nodes.

All 120 instances were also solved using five heuristics and the proposed method. The five heuristics: S-shape, Largest gap, Aisle-by-Aisle, Combined and Combined+, are described in detail by Roodbergen and De Koster ([9]) as extensions of the heuristic routing methods for warehouses with two cross aisles developed by Roodbergen and De Koster ([25]) and the combination of heuristics and dynamic programming algorithm developed by Vaughan and Petersen ([26]). Results for instances with 2 and 3 cross aisles are contained in Table 1, while Table 2 contains results for instances with 4 and 5 cross aisles. The first column of Tables 1 and 2 contains the name of the instances. Columns 2 – 5 contain the number of aisles, cross aisles, stock locations on either side of an aisle and the number of pick locations for each instance. Columns 6 – 12 contain the resulting total traveled distance in *lsl*s by all heuristic methods, the Dantzig's cutting plane method and the proposed metaheuristic method. Computational times are omitted because all algorithms (except the cutting plane method) produce their results within milliseconds. For all instances the proposed method outperforms all other heuristic methods. For 107 out of 120 instances the proposed method reaches the optimal value of the total traveled distance. For the remaining 13 instance the optimality gap is practically negligible as it varies from 0.14% (instance *HundredOrd36*) to 0.7% (instance *HundredOrd34*).

Average results of Tables 1 and 2 are reported in Table 3. The first column contains the name of the groups of instances of the same configuration (number of aisles, cross aisles, locations per aisle and picking locations) where x = 0, …, 9. Figure 4 gives a visual representation of the results contained in the last row of Table 3 and illustrates the efficiency of the proposed algorithm. It should be particularly emphasized that the average of the solutions obtained by the proposed

algorithm is only 0.03% above the average over the optimal solutions, while the best next heuristic, Combined + shows a 13,7% positive gap.

4.2 Case Study

The proposed method was also tested on data from real warehouse management problems arising in a warehouse of a major electrical retailer in Greece. The warehouse consists of 30 aisles, 2 cross aisles (at the front and rear end of the warehouse) and 45 storage locations on either side of every aisle. Figure 5 shows the layout of the warehouse. The total storage area is 4200 m^2 and it is divided into two areas of 2100 m^2: AREA 1 and AREA 2. This partition of the storage area is due to safety reasons against fire, according to the relative legislation (P.D. 71/88 art. 11.3), which defines that the maximum section area should not exceed 2500 m^2. Each one of these two areas has 15 aisles: A, B, C, …, Q. This partition poses limits on the circulation of the order pickers. If a picker has to pass from one area to the other, this can only be done at the front end of aisle Q (storage locations 45) and not at the rear end because areas 1 and 2 are isolated. The warehouse uses VNA (Very Narrow Aisle) vehicles, thus the order pickers can pick items from both sides of a pick aisle simultaneously. The depot is located at the beginning of aisle A (storage location 45) of AREA 1.

The practice followed so far for picking the items of an order is even simpler than the S-shape method ([9]): All storage locations have a serial number. When an order is placed the items of the order are picked in order of the serial number of the location they are stored. This leads to excessively large total travelled distances.

The problem was solved by the proposed method by setting the distances between the nodes crossing at the end of aisles 15 and 16 (AREA 1, aisle Q and AREA 2, aisle Q), to a fairly large positive number.

The problem can be solved to optimality by using the dynamic programming method of Ratliff and Rosenthal ([4]) and setting the length of arc configurations (i), (ii) and (iv) in [4] (p. 515) for crossover between aisles 15 and 16 (aisles: AREA 1, aisle Q and AREA 2, aisle Q), equal to a fairly large positive number.

The proposed method is tested on five orders of 67 to 102 items/order and the results are compared to those from the method used in practice and the optimal results produced by the method of Ratliff and Rosenthal with the modification introduced earlier. The results are reported in Table 4. The proposed method always finds the optimal solution and produces routes with 16.7 % to 22.8 % shorter total distances compared to the method used in practice.

5 Conclusions

The present paper considers the problem of optimizing order picking routes in *low-level picker-to-parts* systems of warehouses that contain parallel aisles, while changing of aisles is allowed through two or more cross aisles. First, the problem is identified and formulated as a TSP problem for any warehouse with the features mentioned before and any order distribution within the storage area. This is achieved by implementing a dynamic programming algorithm for the calculation of the shortest path between any pair of nodes in a connected graph. Then, a metaheuristic algorithm based on the framework of the Threshold Accepting method is proposed for the solution of the formulated problem. An intensive neighborhood (Local) search procedure is incorporated in the body of the algorithm and a sorted list of Thresholds is maintained and updated throughout the entire iterative procedure. The former facilitates the exploration of the most promising areas of the search space, while the latter reduces the number of user defined parameters of the algorithm to just the length of the threshold list, thus making tuning of the algorithm much easier. Computational results of the

proposed method have been compared favorably to a number of algorithms found in the literature, which are specially devised for the solution of the problem at hand. Finally the proposed method has been tested on the solution of a number of instances obtained during the operation of a real warehouse. The method produces better results compared to the method used in practice and improves significantly the warehouse operations.

Future research could encompass dynamic storage assignment, simultaneous order picking and replenishment route optimization, and also inbound receipt and outbound delivery of goods in a complete framework for the maximum utilization of labour and equipment via the practice of task interleaving.

References

[1] De Koster R, Le-Duc T, Roodbergen K J. Design and control of warehouse order picking: A literature review, *Eur J of Oper Res*, **182**, (2007), 481-501.

[2] Tompkins JA, White JA, Bozer YA, Frazelle EH, Tanchoco JMA and Trevino J., *Facilities Planning*, New York: John Wiley & Sons, Inc. 2nd edn. 1996.

[3] Christofides N., *Graph Theory: An Algorithmic Approach*, London, Academic Press, 1975.

[4] Ratliff HD and Rosenthal AS, Orderpicking in a rectangular warehouse: A solvable case of the traveling salesman problem, *Oper Res*, **31**, (1983), 507-521.

[5] Roodbergen KJ, De Koster R. Routing order-pickers in a warehouse with a middle aisle, *Eur J of Oper Res*, **133**, (2001), 32–43.

[6] Hall RWH, Distance approximations for routing manual pickers in a warehouse, *IIE Trans*, **25**, (1993), 76 - 87.

[7] De Koster R, Van der Poort E., Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions, *IIE Trans*., **30**, (1998), 469 - 480.

[8] Petersen CG, An evaluation of order picking routing policies, *Inte J Oper Prod Man*, **17**, (1997), 1098 - 1111.

[9] Roodbergen KJ and De Koster R., Routing methods for warehouses with multiple cross aisles, *Int J Prod Res*, **39**(9), (2001), 865–1883.

[10] Makris PA and Giakoumakis IG. k-Interchange heuristic as an optimization procedure for material handling applications, *Appl Math Model*, **27**(5), (2003), 345–358.

[11] Lin S and Kernighan BW, An effective heuristic algorithm for the traveling salesman problem, Oper Res 1973;21(2): 498–516.

[12] Daniels R, Rummel J and Schantz R., A model for warehouse order picking, *Eur J Oper Res*, **105**, (1998), 1-17.

[13] Rosenkrantz DJ, Steams RE, Lewis EM., An analysis of several heuristics for the Traveling Salesman Problem, *SIAM J Comput*, **6**, (1977), 563-581.

[14] Golden BL, Bodin LD, Doyle T, Stewart W, Approximate traveling salesman algorithms, *Oper Res*, **28**, (1980), 694-711.

[15] Floyd RW, Algorithm 97: Shortest Path, *Commun ACM*, **5**(6), (1962), 345.

[16] Dueck G and Scheuer T., Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing, *J Comput Phys*, **90**, (1990), 161-175.

[17] Dantzig G, Fulkerson R and Johnson S. Solution of a Large-Scale Traveling-Salesman Problem, *J Oper Res Soc Am*, **2**, (1954), 393-410.

[18] Dreyfus SE, An appraisal of some shortest path algorithms, *Oper Res*, **17**, (1959), 395.

[19] Dijkstra EW, A note on two problems in connection with graphs, *Numer Math*, **1**, (1959), 269.

[20] Hu TC., *Integer Programming and Network Flows*, Reading, Massachusetts, Addison-Wesley, 1969.

[21] Nissen V and Paul H., A modification of threshold accepting and its application to the quadratic assignment problem, OR Spectrum, **17**, (1995), 205-210.

[22] Winker P and Fang KT., Application of Threshold Accepting to the Evaluation of the Discrepancy of a Set of Points, *SIAM J Numer Anal*, **34**(5), (1997), 2028-2042.

[23] Cirasella J, Johnson DS, McGeoch LA and Zhang W., The asymmetric traveling salesman problem: Algorithms, instance generators, and tests, *In Proc. 3rd ALENEX*, **2153** of LNCS, Springer-Verlag, (2001), 32-59.

[24] Gambardella LC and Dorigo M., An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, *INFORMS J Comput*, **12**, (2000), 237-255.

[25] Roodbergen KJ and De Koster R., *Routing orderpickers in a warehouse with multiple cross aisles*, in R. J. Graves, L. F. McGinnis, D. J. Medeiros, R. E. Ward and M. R. Wilhelm (eds.), Progress in Material Handling Research: 1998 (Charlotte, NC: Material Handling Institute), (1998), 451 - 467.

[26] Vaughan TS and Petersen CG., The effect of warehouse cross aisles on order picking efficiency, *Int J Prod Res*, **37**, (1999), 881- 897.

List of Figures

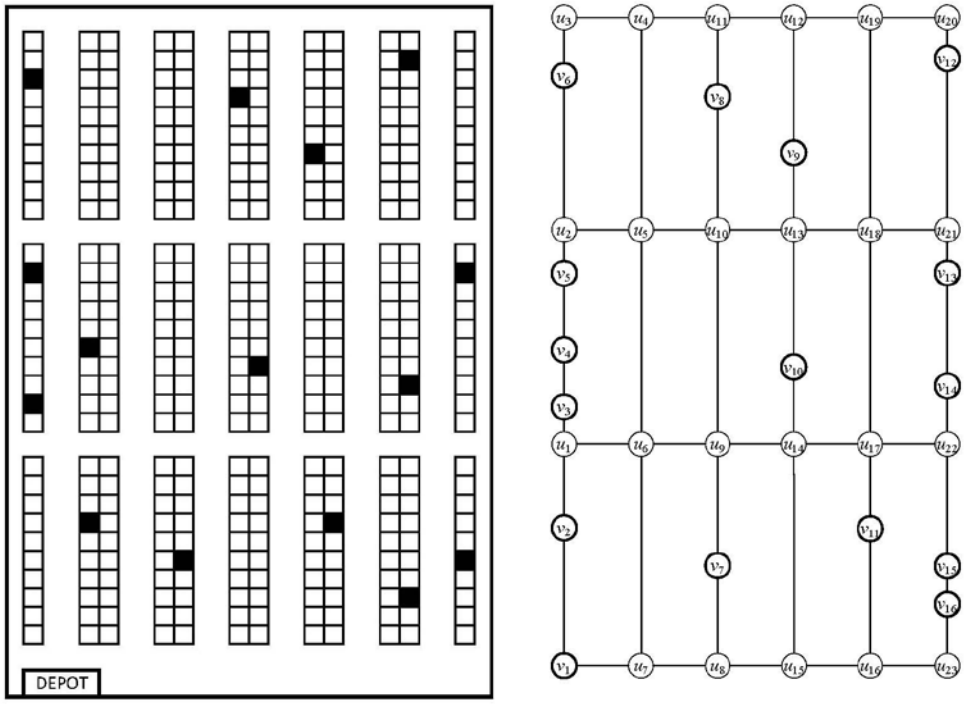

Figure 1: (a) A warehouse layout with 6 aisles and 4 cross aisles and a particular order location problem.
(b)The graph that corresponds to the problem appearing on Figure 1 (a).

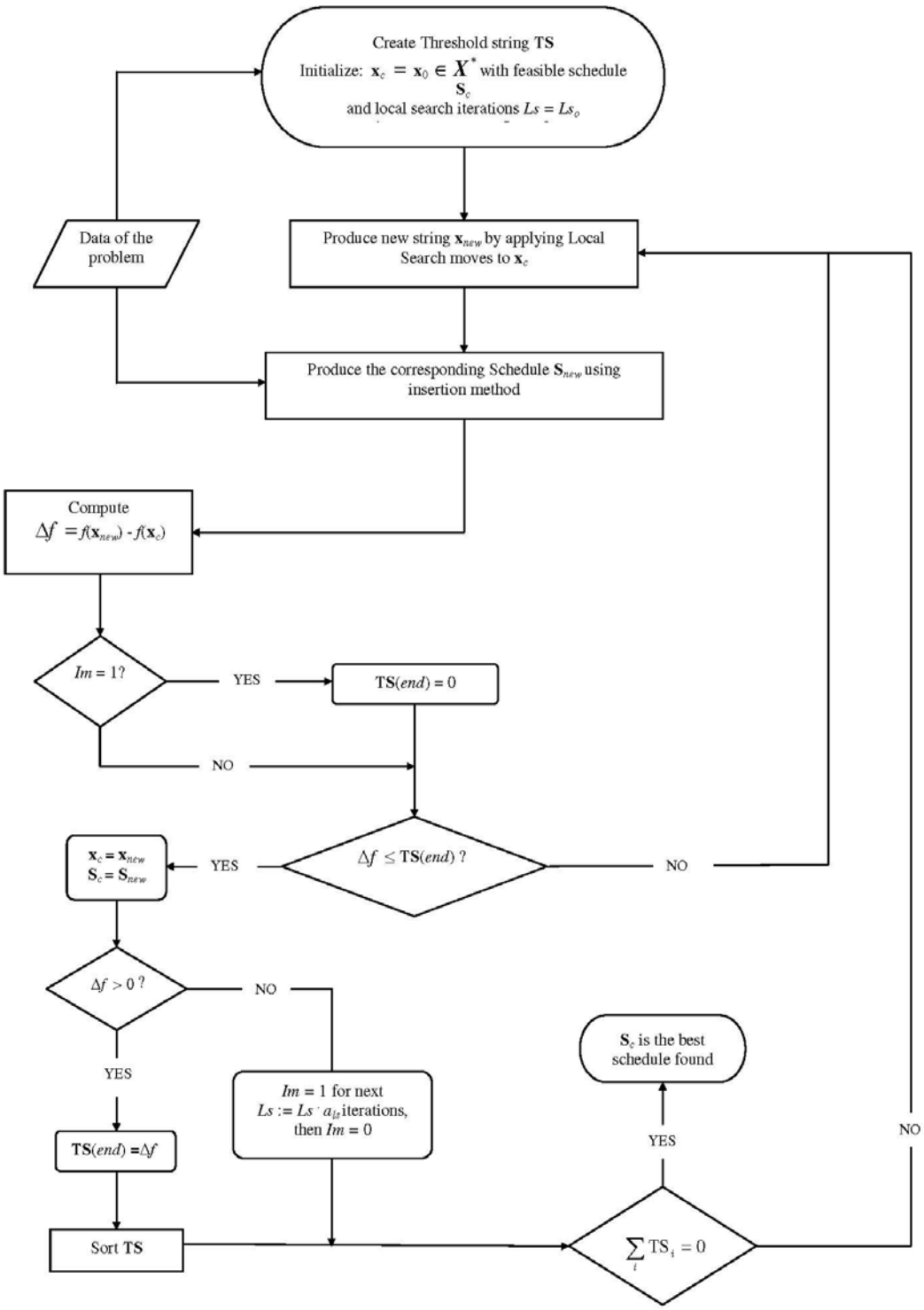

Figure 2: The flow diagram of the proposed algorithm

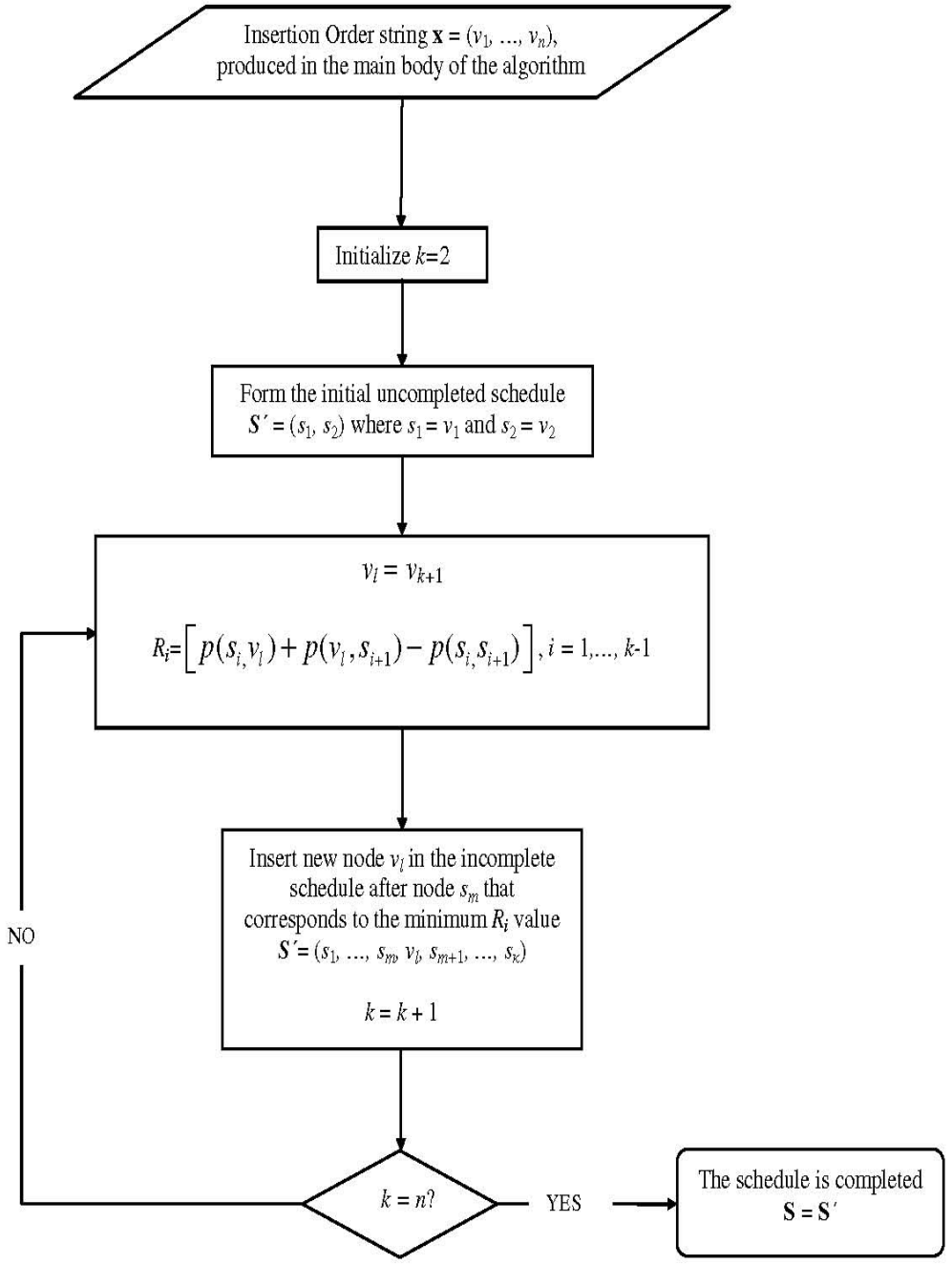

Figure 3: The flow diagram of the Insertion Method

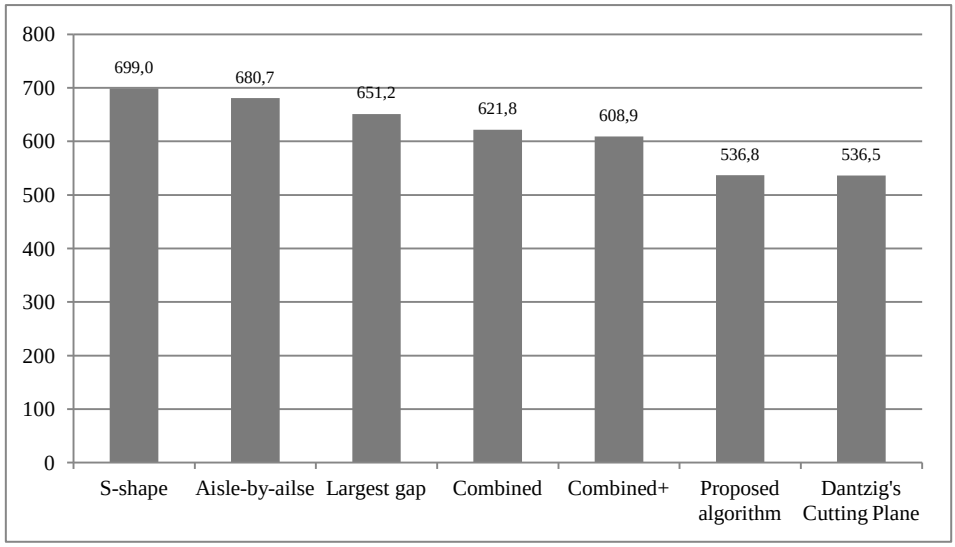

Figure 4: Average travelling distances (in *lsl*s) produced by all tested methods

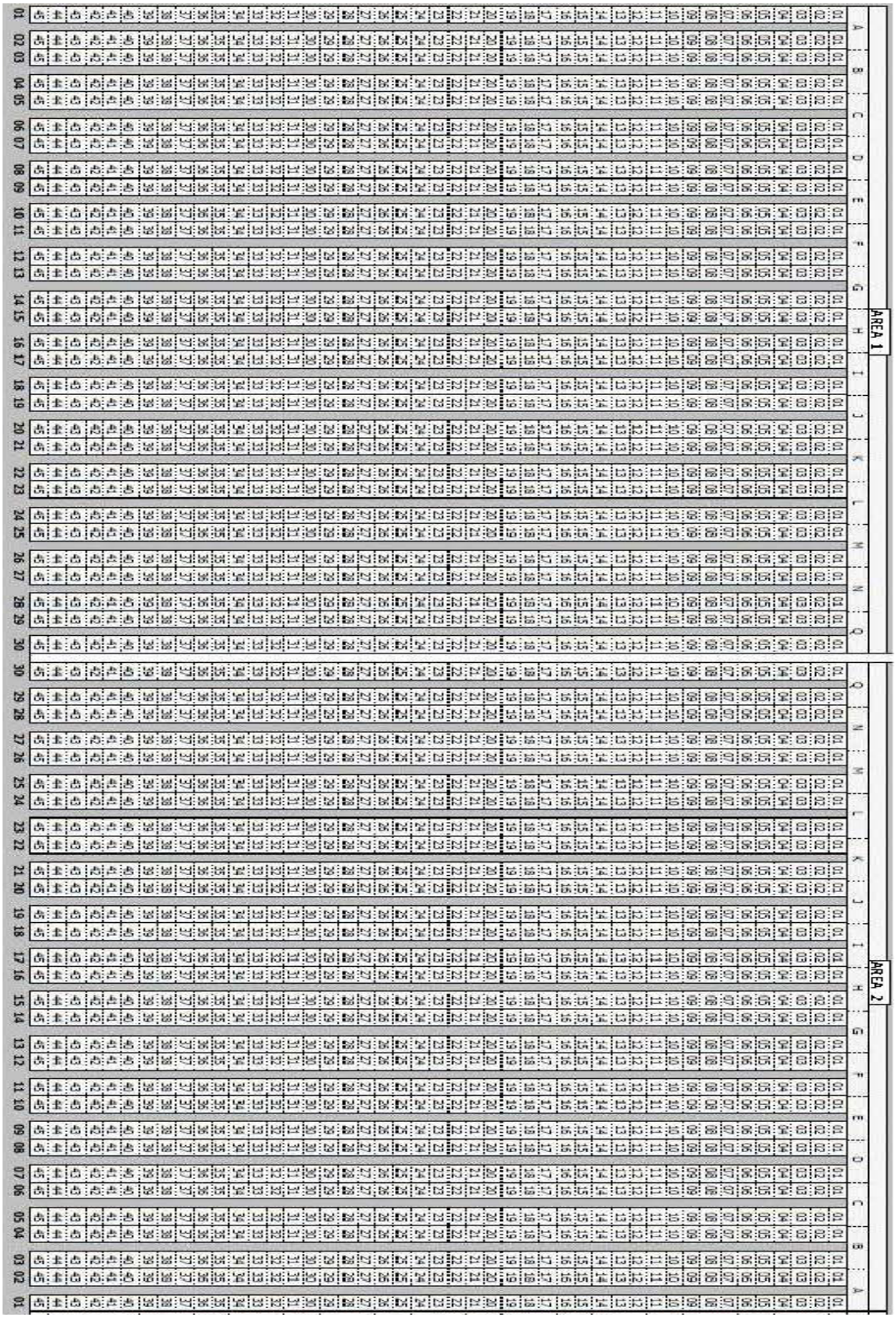

Figure 5: Layout of the warehouse of a major Greek electrical retailer

List of Tables

Instances	Warehouse and order features				Method						
	No. of aisles	No. of cross aisles	No. of locations per aisle	No. of picking locations	S shape	Largest gap	Aisle by aisle	Combined	Combined+	Dantzig's Cutting Plane	Proposed algorithm
TwentyOrd10	18	2	50	20	659	479	563	569	563	466	466
TwentyOrd11	18	2	50	20	768	532	572	618	572	512	512
TwentyOrd12	18	2	50	20	764	605	646	646	646	570	570
TwentyOrd13	18	2	50	20	621	461	477	477	477	427	427
TwentyOrd14	18	2	50	20	668	445	532	542	532	409	409
TwentyOrd15	18	2	50	20	759	500	549	549	549	458	458
TwentyOrd16	18	2	50	20	568	514	532	532	532	492	492
TwentyOrd17	18	2	50	20	601	445	557	557	557	429	429
TwentyOrd18	18	2	50	20	647	524	557	557	557	471	471
TwentyOrd19	18	2	50	20	629	453	507	523	507	435	435
TwentyOrd20	18	3	48	20	450	311	437	352	352	285	285
TwentyOrd21	18	3	48	20	469	370	401	399	373	294	294
TwentyOrd22	18	3	48	20	496	433	454	426	377	335	335
TwentyOrd23	18	3	48	20	400	359	373	330	330	280	280
TwentyOrd24	18	3	48	20	452	361	470	398	368	315	315
TwentyOrd25	18	3	48	20	425	421	432	387	370	356	356
TwentyOrd26	18	3	48	20	439	399	355	403	365	333	333
TwentyOrd27	18	3	48	20	438	337	416	310	298	297	297
TwentyOrd28	18	3	48	20	431	344	388	347	309	276	276
TwentyOrd29	18	3	48	20	382	377	451	358	331	323	323
FiftyOrd10	18	2	50	50	959	936	897	897	897	794	794
FiftyOrd11	18	2	50	50	937	840	801	801	801	769	769
FiftyOrd12	18	2	50	50	965	880	841	841	841	763	763
FiftyOrd13	18	2	50	50	947	839	845	877	845	734	734
FiftyOrd14	18	2	50	50	961	796	895	903	895	753	753
FiftyOrd15	18	2	50	50	965	902	847	847	847	803	803
FiftyOrd16	18	2	50	50	968	941	886	886	886	873	873
FiftyOrd17	18	2	50	50	968	839	872	872	872	742	742
FiftyOrd18	18	2	50	50	859	791	771	771	771	652	652
FiftyOrd19	18	2	50	50	965	757	853	853	853	660	660
FiftyOrd20	18	3	48	50	827	632	793	653	612	537	537
FiftyOrd21	18	3	48	50	731	641	645	651	635	548	548
FiftyOrd22	18	3	48	50	824	674	692	676	647	557	557
FiftyOrd23	18	3	48	50	726	591	745	626	626	521	521
FiftyOrd24	18	3	48	50	743	584	779	641	633	530	530
FiftyOrd25	18	3	48	50	692	673	724	642	642	605	605
FiftyOrd26	18	3	48	50	749	650	711	663	649	589	589
FiftyOrd27	18	3	48	50	781	625	723	663	657	569	569
FiftyOrd28	18	3	48	50	688	643	693	622	608	543	543
FiftyOrd29	18	3	48	50	740	660	801	644	640	574	574
HundredOrd10	18	2	50	100	968	1177	968	968	968	944	944
HundredOrd11	18	2	50	100	968	1144	968	968	968	906	906
HundredOrd12	18	2	50	100	968	1229	968	968	968	938	938
HundredOrd13	18	2	50	100	968	1175	968	968	968	961	961
HundredOrd14	18	2	50	100	968	1195	968	968	968	968	968
HundredOrd15	18	2	50	100	968	1215	968	968	968	951	951
HundredOrd16	18	2	50	100	968	1252	962	962	962	951	951
HundredOrd17	18	2	50	100	968	1162	942	942	942	925	925
HundredOrd18	18	2	50	100	968	1184	968	968	968	968	968
HundredOrd19	18	2	50	100	968	1051	942	942	942	872	872
HundredOrd20	18	3	48	100	932	929	930	896	890	790	790
HundredOrd21	18	3	48	100	932	962	932	832	832	805	805
HundredOrd22	18	3	48	100	923	922	905	873	842	737	737
HundredOrd23	18	3	48	100	919	935	901	855	853	770	770
HundredOrd24	18	3	48	100	932	963	932	896	892	794	794
HundredOrd25	18	3	48	100	929	955	901	869	869	826	826
HundredOrd26	18	3	48	100	919	984	911	863	863	842	842
HundredOrd27	18	3	48	100	884	961	876	828	828	779	779
HundredOrd28	18	3	48	100	932	927	880	832	832	777	777
HundredOrd29	18	3	48	100	884	882	895	790	788	739	739

Table 1. Computational results in *lsls* for instances in warehouses with two and three cross aisles

Instances	Warehouse and order features				Method						
	No. of aisles	No. of cross aisles	No. of locations per aisle	No. of picking locations	S shape	Largest gap	Aisle by ailse	Combined	Combined+	Dantzig's Cutting Plane	Proposed algorithm
TwentyOrd30	18	4	48	20	406	345	420	358	334	296	296
TwentyOrd31	18	4	48	20	396	404	356	344	307	267	267
TwentyOrd32	18	4	48	20	453	390	403	405	335	276	276
TwentyOrd33	18	4	48	20	351	310	349	321	298	249	249
TwentyOrd34	18	4	48	20	410	363	456	346	318	288	288
TwentyOrd35	18	4	48	20	371	343	337	313	303	254	254
TwentyOrd36	18	4	48	20	395	356	317	325	311	273	273
TwentyOrd37	18	4	48	20	385	338	410	327	305	281	281
TwentyOrd38	18	4	48	20	366	325	374	322	288	247	247
TwentyOrd39	18	4	48	20	316	306	392	266	259	253	253
TwentyOrd40	18	5	44	20	314	341	419	278	267	246	246
TwentyOrd41	18	5	44	20	344	343	324	310	266	216	216
TwentyOrd42	18	5	44	20	358	362	372	298	274	232	232
TwentyOrd43	18	5	44	20	306	310	313	270	243	218	218
TwentyOrd44	18	5	44	20	340	322	428	296	271	252	252
TwentyOrd45	18	5	44	20	323	331	307	307	267	218	218
TwentyOrd46	18	5	44	20	336	317	270	302	265	230	230
TwentyOrd47	18	5	44	20	338	290	401	298	269	246	246
TwentyOrd48	18	5	44	20	304	298	342	268	222	212	212
TwentyOrd49	18	5	44	20	299	280	360	273	234	223	223
FiftyOrd30	18	4	48	50	766	645	780	626	626	526	526
FiftyOrd31	18	4	48	50	664	609	615	606	606	498	498
FiftyOrd32	18	4	48	50	708	604	665	606	606	497	497
FiftyOrd33	18	4	48	50	655	532	714	571	571	441	441
FiftyOrd34	18	4	48	50	744	626	685	642	642	506	506
FiftyOrd35	18	4	48	50	721	570	620	595	595	468	468
FiftyOrd36	18	4	48	50	680	615	698	578	578	472	472
FiftyOrd37	18	4	48	50	664	541	657	550	550	439	439
FiftyOrd38	18	4	48	50	696	564	756	580	580	450	450
FiftyOrd39	18	4	48	50	749	543	765	615	615	465	465
FiftyOrd40	18	5	44	50	611	529	723	531	472	391	391
FiftyOrd41	18	5	44	50	582	556	575	512	500	419	419
FiftyOrd42	18	5	44	50	602	525	599	516	500	389	389
FiftyOrd43	18	5	44	50	550	505	651	480	461	384	384
FiftyOrd44	18	5	44	50	544	487	698	452	440	395	395
FiftyOrd45	18	5	44	50	605	545	617	487	468	401	401
FiftyOrd46	18	5	44	50	558	532	565	472	453	381	381
FiftyOrd47	18	5	44	50	551	531	643	459	442	380	380
FiftyOrd48	18	5	44	50	522	508	604	458	448	383	383
FiftyOrd49	18	5	44	50	528	508	693	448	435	387	388
HundredOrd30	18	4	48	100	972	836	910	834	817	704	704
HundredOrd31	18	4	48	100	932	886	894	816	810	718	723
HundredOrd32	18	4	48	100	910	868	883	842	827	707	709
HundredOrd33	18	4	48	100	833	779	855	735	727	637	640
HundredOrd34	18	4	48	100	990	843	915	860	853	702	707
HundredOrd35	18	4	48	100	952	859	873	862	836	707	711
HundredOrd36	18	4	48	100	951	826	860	865	851	716	717
HundredOrd37	18	4	48	100	886	865	862	818	810	692	692
HundredOrd38	18	4	48	100	911	791	870	795	795	662	665
HundredOrd39	18	4	48	100	855	802	864	765	738	634	634
HundredOrd40	18	5	44	100	830	716	826	706	693	570	575
HundredOrd41	18	5	44	100	793	750	815	711	705	601	604
HundredOrd42	18	5	44	100	808	702	816	724	710	554	555
HundredOrd43	18	5	44	100	783	738	790	681	681	550	551
HundredOrd44	18	5	44	100	848	751	844	740	707	596	599
HundredOrd45	18	5	44	100	799	733	796	721	721	586	588
HundredOrd46	18	5	44	100	818	731	785	684	667	576	576
HundredOrd47	18	5	44	100	791	734	792	697	677	577	577
HundredOrd48	18	5	44	100	765	708	791	651	651	565	566
HundredOrd49	18	5	44	100	716	710	805	636	630	552	552

Table 2. Computational results in *lsls* for instances in warehouses with four and five cross aisles

Group of Instances	Warehouse and order features				Method						
	No. of aisles	No. of cross aisles	No. of locations per aisle	No. of picking locations	S shape	Largest gap	Aisle by ailse	Combined	Combined+	Dantzig's Cutting Plane	Proposed algorithm
TwentyOrd1x	18	2	50	20	668,4	495,8	549,2	557,0	549,2	466,9	466,9
TwentyOrd2x	18	3	48	20	438,2	371,2	417,7	371,0	347,3	309,4	309,4
FiftyOrd1x	18	2	50	50	949,4	852,1	850,8	854,8	850,8	754,3	754,3
FiftyOrd2x	18	3	48	50	750,1	637,3	730,6	648,1	634,9	557,3	557,3
HundredOrd1x	18	2	50	100	968,0	1178,4	962,2	962,2	962,2	938,4	938,4
HundredOrd2x	18	3	48	100	918,6	942,0	906,3	853,4	848,9	785,9	785,9
TwentyOrd3x	18	4	48	20	384,9	348,0	381,4	332,7	305,8	268,4	268,4
TwentyOrd4x	18	5	44	20	326,2	319,4	353,6	290,0	257,8	229,3	229,3
FiftyOrd3x	18	4	48	50	704,7	584,9	695,5	596,9	596,9	476,2	476,2
FiftyOrd4x	18	5	44	50	565,3	522,6	636,8	481,5	461,9	391,0	391,1
HundredOrd3x	18	4	48	100	919,2	835,5	878,6	819,2	806,4	687,9	690,2
HundredOrd4x	18	5	44	100	795,1	727,3	806,0	695,1	684,2	572,7	574,3
AVERAGE					699	651,2	680,7	621,8	608,9	536,5	536,8

Table 3. Average computational results of Tables 1 and 2.

Instances	Warehouse and order features				Method			% improvement
	No. of aisles	No. of cross aisles	No. of locations per aisle	No. of picking locations	The method used in practice	Modified Ratliff and Raosenthal algorithm	Proposed algorithm	
1301	30	2	45	89	953	743	743	22,0%
2277	30	2	45	98	1032	860	860	16,7%
3503	30	2	45	90	1320	1074	1074	18,6%
4081	30	2	45	102	1417	1094	1094	22,8%
5552	30	2	45	67	997	825	825	17,3%
AVERAGE					1144	919,2	919,2	19,6%

Table 4. Computational results for five real order instances of a Greek electrical retailer warehouse