A Genetic Algorithm for the Order Batching Problem in Low-Level Picker-to-Part Warehouse Systems

Temel Öncan

***Abstract*—In this work we introduce a Genetic Algorithm (GA) for the Order Batching Problem considering traversal and return routing policies. The proposed GA has been tested on randomly generated instances and compared with the well-known savings algorithm. According to our extensive computational experiments we can say that the proposed GA yields promising solutions in acceptable computation times.**

***Index Terms*—Order Batching Problem; Genetic Algorithm; Warehouse Management**

I. Introduction

WAREHOUSE systems have several functions including receiving, storage, order picking and shipping. Among these functions, order picking is known to be the most labor intensive and costly one [8]. Order picking is the process of retrieving products from their storage locations in order to satisfy customer requests. Order picking costs is estimated to be as much as of 65 % of the total warehouse operating expenses [26]. In this study we consider the order batching problem (OBP) which is shown to be $\mathcal{NP}$-hard by Gademann and Van de Velde [11]. Given both a list of customer orders and order picking routing policy, the OBP deals with constructing batches of customer orders such that the total travel length of pickers is minimized.

Broadly speaking, order-picking systems can be grouped in two categories according to the material handling equipments used: picker-to-parts systems and parts-to-picker systems. In picker-to-parts systems, order pickers travel along the warehouse and retrieve the items requested. On the other hand, in parts-to-picker systems the requested items are handled and transported by automatic storage and retrieval systems (AS/RSs) to order pickers [7], [28]. Particularly, there exists two types of picker-to-parts systems: low-level and high-level picking systems. In low-level picking systems, the picker travels along the aisles in order to pick the requested items from the storage bins or racks. In high-level systems, the pickers drive a truck or crane to reach the pick locations. In this work, we address low-level picker-to-parts picking systems employing human pickers. De Koster, Le-Duc and Roodbergen [7] have claimed that 80 % of all order-picking systems in Western Europe are of this type.

In order picking systems, the service level basically consists of order delivery time, order integrity and accuracy [7]. Order delivery time is closely related with the travel time of the picker. As pointed out by Tompkins, White, Bozer and Tanchoco [26] almost half of the order picker time is wasted while travelling. Despite several activities other than travelling, require a considerable amount of the picker's time ([14], [21], [23]), the time devoted to the travel activity is seen as the most time consuming activity [7]. Furthermore, the travel time has a substantial role in customer satisfaction since the shorter the travel time is; the sooner the requested items are ready for shipping. Hence, among several objective functions that can be taken into consideration such as the minimization of order throughput, maximization of item accessibility, maximization of labor use; the minimization of pickers' total travel distance, which will be also addressed in this paper, is the most widely considered one [7].

In the literature, several order picking routing policies have been introduced: traversal [13], return, midpoint, largest gap [14], composite and optimal [22] routing policies. Petersen [21] claims that the routing policies range from simple to more complex in that order. That is to say, traversal, return and midpoint strategies are simpler than the largest gap, composite and optimal routing policies. According to the experiments by Petersen [21], the optimal routing strategy is the winner at the expense of its disadvantages such as discernible pattern and the routes with backtracks. However, the author asserts also that the traversal, return, midpoint, largest gap and composite routing policies are easy to use. Note that, complex routing policies may yield congestion problems when several pickers share long, narrow and two-way aisles. Simple routing policies may arise to be useful especially for complex order picking systems with many pickers.

In this work, we concentrate only on the OBP considering traversal and return routing policies. In the traversal routing policy also known as the S-shape algorithm, the picker starts from the I/O point, visits every aisle where an item is required to be picked up and returns the I/O point. The picker enters an aisle from one end and leaves from the opposite end. In case the number of aisles that must be visited is odd then the picker enters and returns in the last aisle when it retrieves the last item in that aisle. Note that, only in that case the picker does not necessarily traverses the last aisle completely. In the return routing policy, a picker starts from the I/O point and proceeds along the front aisle. The picker enters each aisle where an item has to be picked up and travels along this aisle as far as the deepest location where an item must be retrieved, then returns and leaves the aisle from the same end.

For the sake of clearness, we present with Fig. 1 an

Manuscript received December 8, 2012; revised January 7, 2013. This research is supported by the Turkish Scientific and Technological Research Council Research Grants Nos. 107M462 and 109M139, and Galatasaray University Scientific Research Project Grant No 12.402.009.

T. Öncan is with the Department of Industrial Engineering, Galatasaray Üniversitesi, Ortaköy, İstanbul, 34357, TÜRKİYE e-mail: ytoncan@gsu.edu.tr

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

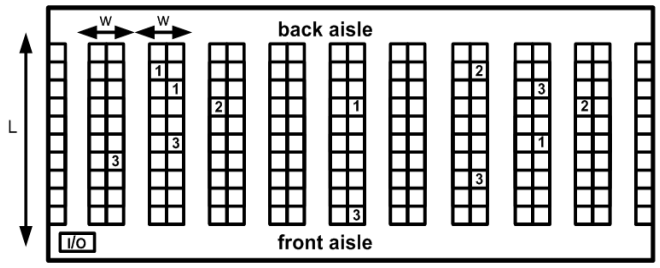

Fig. 1. Layout of a rectangular warehouse

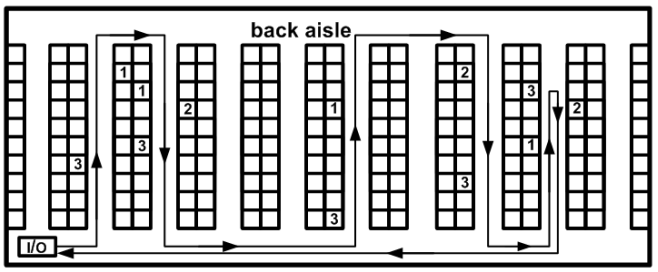

Fig. 2. Traversal Routing Policy

illustration of the warehouse layout that we focus in this work. In Fig. 1, we consider three orders, i.e. order 1, order 2 and order 3 which include 4, 3 and 5 items, respectively. Note that, the location of these items are indicated with order numbers. The shape of the warehouse is assumed to be rectangular with parallel storage. The warehouse totally incorporates 10 parallel aisles. The aisles are number starting from the left most aisle. The I/O point is situated in front of the leftmost aisle. The picking area has the capacity to store 200 items. Each order must be assigned into a batch and each order consists of at least one item. The locations of items are known a priori and we assume that sufficient number of homogenous pickers are available. The amount of items which belong to the orders assigned to a batch should not exceed the picker's capacity. The quantity to be picked up of each item is assumed to be one unit. For the OBP test problems, we assume that the horizontal distance within stocking aisles is negligible and the picker does not need additional time for entering and leaving the aisles. Fig. 2 and Fig. 3 depict the routes of the picker serving all of three orders considering traversal and return policies, respectively.

To the best of our knowledge, there are very few studies addressing the exact solution of the OBP. Gademann, Van den Berg and Van der Hoff [10] have designed a branch and bound algorithm for the order batching with the objective of minimizing the maximum travel time of the pickers. The OBP has been formulated as a set partitioning problem by Gademann and Van de Velde [11] where the authors have devised a branch and price algorithm and have reported the optimum solution of problems up to 32 customer orders. For a revised version of OBP considering the traversal routing policy, Bozer and Kile [2] have proposed a Mixed Integer Programming (MIP) formulation and they could solve small size instances (up to 25 customers) to optimality. The revised version of the OBP addressed by Bozer and Kile [2] is quite different than the original OBP considering the traversal routing policy. The authors address only the traversal routing policy when the number of traversals is even. Their formulation does not compromise the case when the number of traversals is odd and the picker returns back in the last aisle whenever he retrieves the last requested item. Recently, Henn and Wäsher [15] have claimed that after generating all possible feasible batches they could solve OBP instances with up to 40 customer orders by running the set partitioning partitioning formulation by Gademann and Van de Velde [11].

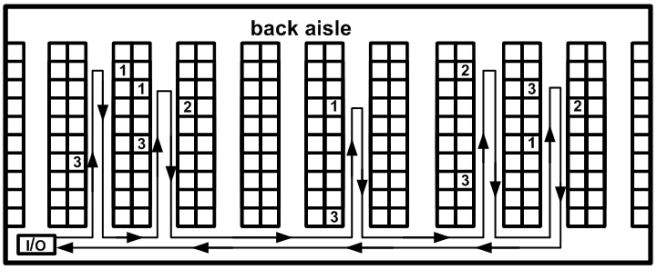

Fig. 3. Return Routing Policy

Several heuristic algorithms have also been developed for the OBP. Among them we can mention the first fit-envelope based batching heuristic [24], the priority rule based algorithms [12], the seed algorithms ([9], [16], [17]) and the savings algorithms [5]. Hwang and Kim [20] proposed order batching algorithm based on cluster analysis. Data mining approaches have been developed by Chen and Wu [4] and Chen, Huang, Chen and Wu [3]. De Koster, Le-Duc and Roodbergen [6] have computationally tested several construction heuristic procedures and have reported that among them the seed algorithm and the savings algorithm yield the most promising performance.

As meta-heuristic algorithms designed for the OBP, we can mention the Genetic Algorithm (GA) proposed by Hsu, Chen and Chen [19], Tabu Search (TS) algorithm by Henn and Washer [15] and the variable neighborhood search algorithm by Albareda-Sambola, Alonso-Ayuso, Molina and Simon de Blas [1]. Tsai, Liou and Huang [27] have simultaneously addressed the OBP and the routing problem which considers both travel distance and order due time. The authors have proposed a GA for this combined problem. For a survey on the OBP we refer to the recent review paper by De Koster, Le-Duc and Roodbergen [7].

The basic motivation of this study is to introduce an efficient GA for the OBP. We compare the performance of the proposed GA with the savings algorithm (named as C&W(ii) in [6]) which is known to be one of the best performing construction heuristics for the OBP [6]. According to our computational experiments we can state that the proposed GA yields comparable performance to the TS algorithm devised by Henn and Washer [15].

The rest of the paper is organized as follows. In Section 2 we present an outline of the proposed GA. This is followed by the computational results in Section 3. Finally, concluding remarks are given in Section 4.

II. Genetic Algorithm

The GA is a stochastic search procedure which simulates the natural selection process [18]. The GA considers a pop-

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

ulation of solutions, which are also called as chromosomes. Each solution is represented with an encoding scheme which serve to translate a solution into a string of genes that form a chromosome.

Each iteration of GA consists of several operators. Among them, reproduction, crossover and mutation are the most common ones. The reproduction operator selects the best-fit chromosomes to the next generation. Crossover allows solutions to exchange information from two randomly chosen parents in order to produce one or more offspring which include some combination of genes from the parents. The mutation operator randomly modify some of the genes within a chromosome. In our implementation, we employ the immigration operator in stead of the mutation operator. The immigration operator creates a small number of new solutions by means of the procedure used to construct the initial population.

A. Encoding and fitness value of a solution

A suitable encoding of the solution is crucial for the performance of the GA since with a good representation it would be possible to clearly state crossover and immigration operators. Keeping in mind this fact, we prefer to represent each solution with a vector of integer numbers. This representation is depicted with Fig. 4 for an OBP instance with 10 orders and 3 batches. Observe that, the first batch consists of orders $\{5, 7, 9\}$; the second batch includes orders $\{1, 3, 4, 6\}$ and the third batch incorporates order $\{2, 8, 10\}$.

Given the set of orders, $i = 1, \ldots, n$ let Q indicate the capacity of each picker and m_i denote the number of items in order i. Note that we assume homogeneous capacity for all pickers. Let x_{ij} be equal to 1 if and only if order i is assigned to batch j. Then the capacity constraints which enforce that the items of all orders assigned to batch j satisfy the capacity of the picker is as follows.

$$\sum_{i=1}^{n} m_i x_{ij} \leq Q \qquad \text{for } j = 1, \ldots, p \tag{1}$$

Here, the number of batches is computed as follows $p = \left\lceil \left(\sum_{i=1}^{n} m_i \right) / Q \right\rceil$ where $\lceil y \rceil$ is the smallest integer not less than y.

For each solution $\mathbf{x}^t \in \Psi$, where Ψ is the set of all possible assignments of n order to p batches, we compute fitness and unfitness functions. The fitness function of a solution is equal to its objective function value c^t. Namely, the total tour length of all pickers. The unfitness u^t of a solution $\mathbf{x}^t$ measures the infeasibility of the capacity constraints (1). That is to say, the unfitness function is as follows: $u^t = \sum_{j=1}^{n} \max \left\{ 0, \sum_{i=1}^{n} m_i x_{ij}^t - Q \right\}$. Note that when, $u^t = 0$ then the $\mathbf{x}^t$ solution satisfies the capacity constraints (1).

B. Initial Population

The GA is initialized with a randomly generated chromosome set which is called as the initial population. To generate initial population we make use of the seed algorithm with random order selection rule [12]. In the initialization step, p out of n orders are randomly selected as the seed orders. For each of the selected p orders, we assign a picker. Then in the next step, we first construct a random sequence of the remaining $n - p$ orders which are waiting to be assigned to a picker. Then, starting from the top of the constructed list, we assign each order to the most suitable picker as long as the picker has enough capacity to carry that order. This step continues until all orders are assigned to a picker.

C. Improvement Heuristic

When a new solution is created after crossover operator, immigration operator or the initial population construction phase, we try to improve the new solution using an improvement heuristic which consists of $MOVE$, $SWAP(1,1)$, $SWAP(1,2)$ and $SWAP(2,1)$ operations. The improvement heuristic perform these operations in that order and the best move for each operation is performed at the end of its run. The $MOVE$ operation tries to remove an order from its batch and re-assign to another batch. The $SWAP(1,1)$ operation considers all possible pairwise interchanges of orders. $SWAP(1,2)$ and $SWAP(2,1)$ operations attempt to remove one order from a batch and two orders from another batch; then exchange their assignments. The improvement heuristic consists of two phases. In the first phase, the heuristic tries to construct a feasible solution and in the second phase it makes an effort to improve the current solution value without harming the feasibility. In case the current solution is feasible the first phase is skipped.

In the first phase, we apply $MOVE$ operation which tries to remove an order assigned to a batch which exceeds the picker capacity Q and to move it to another batch which has enough capacity. Considering all possible moves the one which yields the largest enhancement in the feasibility, namely the largest improvement in the unfitness function u^t, is performed. The $MOVE$ operation is run until no further enhancement in the unfitness function is observed. Next, $SWAP(1,1)$ operation is performed in order to exchange two orders such that one of them is assigned to an over-capacitated batch and the other one is assigned to an under-capacitated batch. Then, among all possible order exchanges, the one which yields the largest enhancement in the feasibility, i.e. unfitness function u^t, is done and the $SWAP(1,1)$ operation is run until no more improvement is obtained. Finally, $SWAP(1,2)$ and $SWAP(2,1)$ operations are run until they do not yield enhancement in the feasibility of the solution. The first phase terminates after applying $MOVE$, $SWAP(1,1)$, $SWAP(1,2)$ and $SWAP(2,1)$ operations without having any improvement.

The second phase of the improvement heuristic address the enhancement in the solution value without violating the feasibility. Given the current assignment of n orders to p batches, $MOVE$, $SWAP(1,1)$, $SWAP(1,2)$ and $SWAP(2,1)$ operations are implemented in their turn and we compute the change in the objective value that results at the end of these operations. Then the move resulting in the largest improvement and without harming the feasibility of the solution is realized. This procedure is repeated until no further improvement in the solution value is possible.

D. Crossover and Immigration

In order to implement crossover operation, two solutions are selected from population and recombined producing

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

offspring. The crossover operations exchanges information between two selected solutions. We have employed the parameterized uniform crossover operation [25] which yields off-spring by mixing the information of two parents. The parameterized uniform crossover mixes the information of two parents according to a fixed mixing ratio. In case the mixing ratio is 0.5 then half of information comes from the first parent and the other half is inherited from the second parent. In other words, in case the mixing ratio is set to 0.5, then the parameterized uniform crossover considers each gene in the parent strings for exchange with a probability of 0.5. In our GA implementation we set the fixed mixing ratio to 0.6. That is to say, the genes of a child are inherited from the first parent with probability 0.6 and from the second parent with probability 0.4. The parameterized uniform crossover is illustrated with Fig. 4 and Fig. 5.

The immigration operator serves to yield genetic diversity from one generation to the next one and hence it prevents us falling into local optimums. Note that, our immigration operator first employs the seed algorithm with random order selection rule [12]. Then we try to enhance this solution by running the improvement heuristic.

E. Generation of a new population

The population size plays a critical role in the performance of the GA. According to our preliminary computational experiments, We have observed that when the size of the population is small, then the convergence of the GA becomes faster. Moreover, with the increasing population size one may encounter convergence problems. Considering the trade-off between the population size and the convergence of the GA, we have decided to employ a population size of $20+ n/2$ solutions.

During the evolution process, we form a mating pool from the current population by replicating each chromosome twice. Then we randomly select random pairs of the parents without replacement and we generate two offspring from each pair by using the parameterized uniform crossover operator. As a result of this operation the population is doubled. First, we sort the parents and offspring in increasing order of their unfitness values. Then, the feasible solutions in both lists are further ordered according to their fitness values.

After applying reproduction, crossover and immigration operators, we are ready to generate a new population. In our implementation, 20% of the new population is chosen from the best solution in the previous population, namely the list of parents, and 60% of the new population is selected from the best offspring obtained with the crossover operator. Finally, the remaining 20% is generated via immigration. The population has evolved through a number of generations until a stopping criterion is satisfied.

Furthermore, during the generation of new population we do not allow duplicate solutions. Whenever a new solution is created, after improvement phase, it is compared to the solutions already in the solution. In case it is the same to any of the existing solution then it is not included in the next generation. In addition, we do not allow multiple combinations of orders batches. As for instance, when orders $\{1, 7, 2\}$; $\{4, 6\}$ and $\{3, 5\}$ are assigned to batches 1, 2 and 3 respectively. Then we would not allow solutions which are represented with the assignment of these orders to batches 1, 3 and 2; 2, 1 and 3; 2, 3 and 1; 3, 1 and 2; 3, 2 and 1.

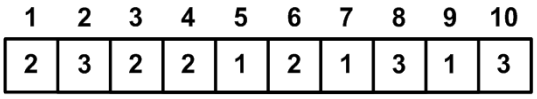

Fig. 4. Solution representation

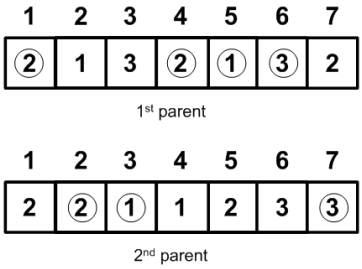

Fig. 5. Before uniform crossover operator

F. Stopping Criterion

After a careful experimentation and fine-tuning process, we decided to stop the algorithm when the number of generations reaches := $40+ \lceil n/3 \rceil$. This setting is found to be a suitable choice for most of the instances. Additionally, we stop the algorithm, when $\lceil n/4 \rceil$ consecutive generations have failed to improve the best known solution.

III. Computational Experiments

In this section we present the details of our computational experiments. The algorithms are coded in C++ and tested on a Dell Server PE2900 with two 3.16 GHz Quad Core Processors and 32 GB RAM with Microsoft Windows Server 2003 operating system. In the literature, there is no standard test library for the OBP. Hence we have produced our test instances to carry out our computational experiments.

In the test instances that we have randomly produced, the number of orders varies from 10 to 100 with an increment of 10. For each number of orders we have generated 10 instances which totally makes 100 OBP test problems. The number of items for each order is randomly chosen between 2 and 10. The items are randomly assigned to locations. According to the capacity of the order picker we have three classes of randomly generated test instances. These are

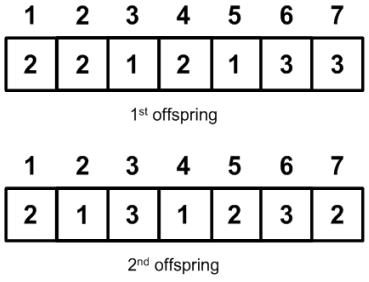

Fig. 6. After uniform crossover operator

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the first class, the second class and the third class of test instances, which assume picker capacities $Q = 24$, $Q = 36$, and $Q = 48$, respectively.

In Table I, Table II and Table III (Table IV, Table V and Table VI), we report the computational results obtained with OBP considering the traversal (return) routing policy and picker capacities $Q = 24$, $Q = 36$ and $Q = 48$, respectively. The first columns in all tables denote the instance names and sizes. The last row of all table include the overall column averages. The number of orders *n* is followed by the capacity of the picker *Q*. For example the row *20_36* stands for the computational experiments obtained with 10 OBP test instances with 20 orders and picker capacity $Q = 36$. The next two columns include the experimental results obtained with the savings heuristic (named as C&W(ii) in [6]) and the last two columns report the results obtained with the GA algorithm. We have chosen the savings heuristic because of its promising performance as pointed out by De Koster, Van Der Poort and Wolters [6]. The CPU times reported are in seconds and UB stands for the upper bound value. We assess the performance of the proposed GA in terms of solution accuracy within a CPU time limit on randomly generated test problems.

The average percent improvements obtained with the GA algorithm for the OBP considering traversal policy over the savings algorithm are 3.94 %, 3.55 % and 3.65 % for picker capacity $Q = 24$, $Q = 36$ and $Q = 48$, respectively. These values are 4.08 %, 3.89 % and 4.16 % for the OBP considering return policy.

The formulae used to calculate the average percent improvements is

$$100 \times \frac{(Z_{UB}^{S} - Z_{UB}^{GA})}{Z_{UB}^{S}} \tag{2}$$

where Z_{UB}^{S} and Z_{UB}^{GA} are respectively the upper bounds obtained with the savings algorithm and the GA.

As can be observed, the performance of MILP formulations improves for larger values of Q. Furthermore, considering the overall average percent improvements the MILP formulation for the OBP considering traversal and return policies, which are 3.71 % and 4.04 % respectively. Although we can not say that it is a fair comparison, since both the test instance and computational platform are different, the TS algorithm by Henn and Washer [15] for the OBP considering the traversal policy, yields an average improvement over the C&W(ii) in [6] amounts to 4.05 %. Hence, we can say that the performance of the proposed GA is comparable to their TS algorithm.

Moreover, when we consider the CPU times required by the GA and the savings algorithm; the winner is the saving algorithm. However, we believe the attempts to obtain more accurate solutions of the OBP at the expense of additional CPU time are worthwhile.

TABLE I
COMPUTATIONAL RESULTS WITH THE SAVINGS ALGORITHM AND THE GA FOR THE OBP CONSIDERING TRAVERSAL POLICY WITH $Q = 24$.

TRAVERSAL	SAVINGS		GA	
	UB	CPU	UB	CPU
10_24	391.44	0.0	367.04	1.1
20_24	691.76	0.0	666.86	2.0
30_24	968.4	0.0	930.63	3.5
40_24	1265.84	0.1	1232.93	5.4
50_24	1528.16	0.2	1450.22	8.6
60_24	1814.08	0.8	1745.14	12.5
70_24	2134.08	1.4	2078.59	28.9
80_24	2395.04	2.9	2313.61	43.4
90_24	2668.64	5.0	2559.23	76.1
100_24	2954.48	8.5	2833.35	128.8
Average	**1681.19**	**1.9**	**1617.76**	**31.0**

TABLE II
COMPUTATIONAL RESULTS WITH THE SAVINGS ALGORITHM AND THE GA FOR THE OBP CONSIDERING TRAVERSAL POLICY WITH $Q = 36$.

TRAVERSAL	SAVINGS		GA	
	UB	CPU	UB	CPU
10_36	291.6	0.0	278.48	1.3
20_36	511.36	0.0	488.86	2.2
30_36	687.36	0.0	656.43	4.2
40_36	922	0.1	895.26	6.2
50_36	1107.28	0.2	1075.17	10.5
60_36	1307.04	0.6	1257.37	15.7
70_36	1534.4	1.2	1496.04	24.1
80_36	1691.04	2.2	1640.31	48.9
90_36	1913.44	4.0	1859.86	94.5
100_36	2107.68	6.4	2019.16	141.2
Average	**1207.32**	**1.5**	**1166.69**	**34.9**

TABLE III
COMPUTATIONAL RESULTS WITH THE SAVINGS ALGORITHM AND THE GA FOR THE OBP CONSIDERING TRAVERSAL POLICY WITH $Q = 48$.

TRAVERSAL	SAVINGS		GA	
48	UB	CPU	UB	CPU
10_48	249.68	0.0	238.69	1.6
20_48	408.24	0.0	392.73	2.9
30_48	576.64	0.0	557.61	5.8
40_48	748.32	0.1	718.39	8.6
50_48	889.04	0.2	847.26	11.8
60_48	1046.24	0.6	1015.90	19.2
70_48	1206.48	1.1	1173.91	28.3
80_48	1342.72	2.2	1271.56	55.6
90_48	1532.16	3.9	1489.26	99.1
100_48	1664.24	6.5	1620.97	156.2
Average	**966.376**	**1.5**	**932.63**	**38.9**

IV. CONCLUSION

We have addressed the order batching problem (OBP) which is known to be $\mathcal{NP}$-hard. The proposed Genetic Algorithm (GA) has been tested on randomly generated instances and they have been compared with the savings algorithm which is known to be one of the most promising construction heuristics for the OBP. From the experimental results, we observe that the proposed GA yields quite good upper bounds in reasonable CPU times. According to our computational experiments we have observed that the proposed GA yields a comparable performance to a recent Tabu Search algorithm for the OBP.

REFERENCES

[1] M. Albareda-Sambola, M. Alonso-Ayuso, E. Molina, and C. Simon de Blas, "Variable neighborhood search for order batching in a warehouse,"

TABLE IV
COMPUTATIONAL RESULTS WITH THE SAVINGS ALGORITHM AND THE GA FOR THE OBP CONSIDERING RETURN POLICY WITH $Q = 24$.

RETURN	SAVINGS		GA	
	UB	CPU	UB	CPU
10_24	471.72	0.0	458.52	1.2
20_24	855.12	0.0	818.3	2.2
30_24	1216.6	0.0	1171.6	4.0
40_24	1569.2	0.1	1489.2	6.4
50_24	1913.16	0.3	1813.7	8.7
60_24	2272.16	0.7	2188.1	15.1
70_24	2664.2	1.5	2584.3	35.3
80_24	2974.04	2.7	2881.8	46.0
90_24	3125.4	3.3	2947.3	93.6
100_24	3395.7	3.9	3253.1	131.4
Average	**2045.73**	**1.3**	**1960.6**	**34.4**

TABLE V
COMPUTATIONAL RESULTS WITH THE SAVINGS ALGORITHM AND THE GA FOR THE OBP CONSIDERING RETURN POLICY WITH $Q = 36$.

RETURN	SAVINGS		GA	
	UB	CPU	UB	CPU
10_36	369.32	0.0	352.0	1.3
20_36	672.64	0.0	649.8	2.2
30_36	931.48	0.0	890.5	4.7
40_36	1229.16	0.1	1191.1	7.2
50_36	1485.8	0.2	1429.3	12.6
60_36	1761.68	0.5	1700.0	19.4
70_36	2043.64	1.1	1953.7	25.3
80_36	2289.28	2.1	2190.8	59.7
90_36	2592.52	3.7	2509.6	115.3
100_36	2840.92	6.1	2724.4	166.6
Average	**1621.64**	**1.4**	**1559.12**	**41.4**

TABLE VI
COMPUTATIONAL RESULTS WITH THE SAVINGS ALGORITHM AND THE GA FOR THE OBP CONSIDERING RETURN POLICY WITH $Q = 48$.

RETURN	SAVINGS		GA	
	UB	CPU	UB	CPU
10_48	334.16	0.0	320.79	1.8
20_48	568.08	0.0	542.52	3.6
30_48	784.52	0.0	749.22	7.3
40_48	1022.12	0.1	982.26	8.9
50_48	1228.6	0.2	1161.03	13.9
60_48	1450.24	0.5	1387.88	22.7
70_48	1713.32	1.1	1661.92	31.1
80_48	1883.04	2.1	1794.54	68.9
90_48	2147.88	3.7	2061.96	121.9
100_48	2343.6	6.2	2268.60	167.1
Average	**1347.56**	**1.4**	**1293.07**	**44.7**

Asia-Pacific Journal of Operational Research, vol. 26, no. 5, pp. 655-683, 2009.

[2] Y. A. Bozer and J. W. Kile, "Order batching in walk-and-pick order picking systems," *International Journal of Production Research*, vol. 46, no. 1, pp. 1887-1909, 2008.

[3] M. -C. Chen, C. -L. Huang, K.-Y. Chen and H.-P. Wu, "Aggregation of orders in distribution centers using data mining," *Expert Systems with Applications*, vol. 28, no. 3, pp. 453-460, 2005.

[4] M. -C. Chen and H. -P. Wu, "An association-based clustering approach to order batching considering customer demand patterns," *Omega - The International Journal of Management Science*, vol. 33, no. 4, pp. 333-343, 2005.

[5] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, no. 4 , pp. 568-581, 1964.

[6] R. De Koster, E. Van Der Poort and M. Wolters, "Efficient orderbatching methods in warehouses," *International Journal of Production Research*, vol. 37, no. 7, pp. 1479-1504, 1999.

[7] R. De Koster, T. Le-Duc and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *European Journal of Operational Research*, vol. 182, no. 481-501, 2007.

[8] J. Drury, *Towards more efficient order picking*. IMM monograph no. 1, The Institute of Materials Managements: Cranfield, UK, 1988.

[9] E. A. Elsayed, "Algorithms for optimal material handling in automatic warehousing systems," *International Journal of Production Research*, vol. 19, no. 5, pp. 525-535, 1981.

[10] N. Gademann, J. Van den Berg and H. Van der Hoff, "An order batching algorithm for wave picking in a parallel-aisle warehouse," *IIE Transactions*, vol. 33, no. 5, pp. 385-398, 2001.

[11] N. Gademann and S. Van de Velde, "Order batching to minimize total travel time in a parallel-aisle warehouse," *IIE Transactions*, vol. 37, no. 1, pp. 63-75, 2005.

[12] D. R. Gibson and G. P. Sharp, "Order Batching Procedures," *European Journal of Operational Research*, vol. 58, pp. 57-67, 1992.

[13] M. Goetschalckx and H. D. Ratliff, "Order picking in an aisle," *IIE Transactions*, vol. 20, no.1, pp. 53-62, 1998.

[14] R. W. Hall, "Distance approximations for routing manual pickers in a warehouse," *IIE Transactions*, vol. 24, no. 4, pp. 76-87, 1993.

[15] S. Henn and G. Wäsher, "Tabu search heuristics for the order batching problem in manual order picking systems," *European Journal of Operational Research*, vol. 222, pp. 484-494, 2012.

[16] Y. -C. Ho, T. -S. Su and Z. -B. Shi, "Order-bathcing methods for an order-picking warehouse with two cross aisles," *Computers and Industrial Engineering*, vol. 55, no. 2, pp. 321-347, 2008.

[17] Y. -C. Ho and Y. -Y. Tseng, "A study on order-batching methods of order-picking in a distribution centre with two cross-aisles," *International Journal of Production Research*, vol. 44, no.17, pp. 3391-3417, 2006.

[18] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press: Ann Arbor, MI, 1975.

[19] C. -M. Hsu, K. -Y. Chen and M. -C. Chen, "Batching orders in warehouses by minimizing travel distance with genetic algorithms," *Computers in Industry*, vol. 56, no. 2, pp. 169-178, 2005.

[20] H. Hwang and D. G. Kim, "Order-batching heuristics based on cluster analysis in low-level picker-to-part warehousing system," *International Journal of Production Research*, vol. 43, no. 17, pp. 3657-3670, 2005.

[21] C. G. Petersen, "An evaluation of order picking routeing policies," *International Journal of Operations and Production Management*, vol. 17, no. 11, pp. 1098-1111, 1997.

[22] H. D. Ratliff and A. S. Rosenthal, "Orderpicking in a rectangular warehouse: a solvable case of the traveling salesman problem," *Operations Research*, vol. 31, pp. 507-521, 1983.

[23] K. J. Roodbergen and R. De Koster, "Routing methods for warehouses with multiple cross aisles," *International Journal of Production Research*, vol. 39, no. 9, pp. 1865–1883, 2001.

[24] R. A. Ruben and F. R. Jacobs, "Batch construction and storage assignment," *Management Science*, vol. 45, no. 4, pp. 575-596, 1999.

[25] W. M. Spears and K. A. "On the virtues of parameterizes uniform corssover," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1991, pp. 230-236.

[26] J. A. Tompkins, J. A. White, Y. A. Bozer and J. M. A. Tanchoco, *Facilities Planning*. 3rd ed. John Wiley & Sons, New Jersey, 2003.

[27] C. -Y. Tsai, J. J. M. Liou and T. -M. Huang, "Using a multiple-GA method to solve the batch picking problem: considering travel distance and order due time," *International Journal of Production Research*, vol. 46, no. 22, pp. 6533-6555, 2008.

[28] G. Wäsher, "Order picking: a survey of planning problems ans methods," in: H. Dyckoff, R. Lackes, J. Reeves (eds.) *Supply Chain Management and Reverse Logistics 2004*, Springer, Berlin, pp. 323-347.

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)