計測自動制御学会産業論文集
Vol.7, No.5, 33/39(2008)

Scheduling of Order-picking with Replenishment in a Warehouse Environment

Jie GONG,* Hirofumi TAMURA,# Toshimitsu HIGASHI,+ and Jun OTA*

In this paper, given the orders and stock in a warehouse, we present a metaheuristic-based scheduler that aims to solve a scheduling problem of order-picking with replenishment. Our objective is to minimize the maximum operation time of order-picking carts. The problem is described with respect to the carts to replenish and pick up products separately within the same period. Sub-problems in this research are (1) how to reduce delays among carts and (2) how to reduce the carts' travel distance. Several operators aiming to generate transitions in the schedule are implemented. After that, a local search procedure is used to solve the routing problem and reduce the delays among carts. The experiment results show that the metaheuristic can make significant improvements with respect to minimizing the operation time.

Key words: warehouse management, order-picking, replenishing, metaheuristics, scheduling

1. INTRODUCTION

1.1 Background

Improvements in warehouse management in recent years are being attributed to gains in logistics. Logistics improvements lead to increased efficiency in warehouse operations and to more reliable customer service. The main operations for warehouse include receiving, storing, order-picking, and shipping. One study has shown that order-picking accounts for as much as 55% of the total warehouse operating cost [1].

Pallets and products are picked from the storage locations by *pickers*. With common low-level order-picking systems, pickers are humans employing *carts*.

Nowadays, order-picking is the costliest activity in a warehouse. It is divided into a reserve area and a picking area (forward area). Because of the shortage products during intensive order-picking, replenishment is very important [2]. The replenishment activity becomes more frequent and crucial during a busy picking period. More attention is currently being given to such problems as timing and replenishment.

1.2 Literature Survey

Warehouse design has traditionally been divided into tactical or operational levels [3]. On a tactical level, replenishment is mainly discussed in the field of inventory theory. Many studies have been made in this field [4]-[6]. Such studies have dealt with a multi-retailer problem, multi-item problems, and the limited-capacity problem of warehouses. Several studies on layout design have also been conducted [7]. Another issue is storage assignment, which involves the number of products to be placed in forward and reserve areas. The decisions concerning the problems described are commonly called the *forward-reserve problem* [8]. In this problem, the regular replenishment quantity from the reserve to the forward area is concerned.

On the operational level, the picking problem has been addressed as a routing method problem, which aims to sequence the picking items to ensure a good route [9][10]. Such a picking problem can be represented in the form of the Traveling Salesman Problem (TSP) and solved with adequate algorithms [11]. In these studies, one important assumption is that the stock (product quantity in the storage location) is unlimited and there is no need to replenish the product supplies. The replenishment problem on the operational level has been discussed in [12]. The warehouse environment discussed in the paper is somewhat specific, whereas the replenishment area and the picking area are completely separate. Therefore, it is difficult to apply the method directly to a general warehouse environment.

In previous studies, the replenishment problem was mainly addressed on a tactical level, such as the forward-reserve problem. On the operational level, only a few studies have been conducted with regard to order-picking with replenishment. Thus, in this

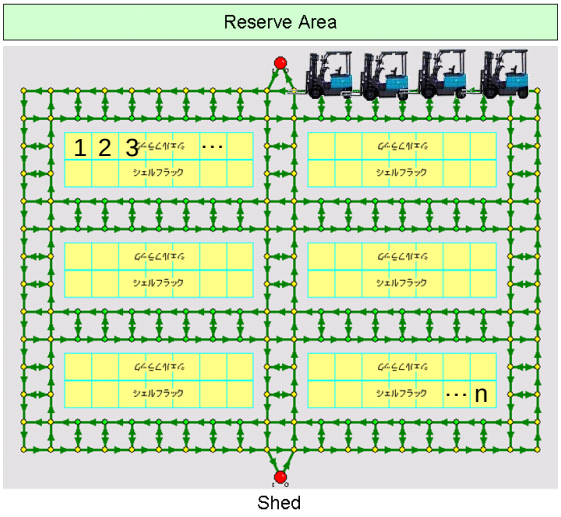


Fig. 1 A model of order-picking in a warehouse

* Department of Precision Engineering, Graduate School of Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
Murata Systems, Ltd.
3 Minamiochiai-cho, Kisshoin, Minami-ku, Kyoto 601-8326, Japan.
+ Murata Machinery, Ltd.
2 Nakajima, Hashizume, Inuyama-shi, Aichi 484-8502, Japan.

(Received December 28, 2007)

Table 1 An example of the order list by shops

Shop	Product 1	Product 2	Product 3	…	Product n
Shop A	10	2	0	…	0
Shop B	40	35	15	…	20
…	…	…	…	…	…
Shop M	0	10	0	…	60

Table 2 An example of the stock sheet

Location	Product	Quantity
1	AA	40
2	AB	50
3	AC	80
⋮	⋮	⋮
n	CY	20

research, our objective is to model and present a search-based metahueristic to solve the problems related to order-picking with replenishment on the operational level.

The problem of order-picking with replenishment differs from typical order-picking problems. Both of the replenishment operation and the picking operation are made in the common working area. Both the replenishment and picking operations occur in a common work area. A primary constraint is that the picking cart operation cannot be completed until the products that are in short supply are replenished.

1.3 A Warehouse Model

A warehouse model is shown in **Fig. 1**. The storage locations hold initial quantities of products (stock). Each location has its own location number, as shown in the figure. The items to be picked from given locations are obtained from the orders sent by shops. The items in short supply are transported by carts from an Automated Storage and Retrieval System (AS/RS) to replenish the storage sites. The items selected from a corresponding order are picked up and deposited at the shed for loading into trucks and delivery to the customers. Order-picking schedules are made by clustering tasks and by assigning them to carts assignments.

The rest of this paper is organized as follows: Section 2 offers an overview of the problem of order-picking with replenishment; Section 3 is a discussion of the methodology of the research; Section 4 is a presentation of the experiment results; and Section 5 is the conclusion.

2. Problem Statement

An overview of the warehouse picking problem is given first. The products are stored in stationary storage locations throughout the warehouse (Fig. 1). Each location holds quantities of unique types of products and has a unit quantity called a *case*. The shop order and the stock for every product serve as inputs to the picking problem, as shown in **Tables 1** and **2**, respectively. For one storage location, there is a capacity that is represented by CAP_s. If the quantity of products exceeds the capacity, the unloading time of the exceeding part is assumed to be N_{pu} times more than normal.

The orders placed on the warehouse should be picked up by the carts from the storage locations and transferred to the shed. A cart will start from the shed and pick up the products at the various locations. If the quantity reaches the capacity of the cart, represented as CAP_a, the cart will return to the shed and will unload the products. This is referred to as a *trip* and assigned to carts. **Figure 2** shows an example of the trip. In this case, the trip connects four picking locations.

As the products are picked up, the stock decreases, and it is necessary to replenish them at the storage sites. As in the pickup process, the products in shortfall are uploaded on the carts from the AS/RS and replenished to the sites. When the cart is empty, it returns to the AS/RS and uploads products for the next replenishment round. This circuitous route is called a replenishment trip. The carts undertake multiple pickup trips that are started and completed at a shed.

We then determine the cart sequence for pickup and replenishment, which is the output as shown in **Fig. 3**. Here, a schedule composed of two carts (C_1 and C_2) is shown. The numbers without parentheses correspond to storage locations, while the numbers in parentheses are the numbers of items to be picked or replenished at the corresponding locations. The start of trip is denoted by "s" or "r." Here, "s" represents the shed and "r" represents AS/RS. Here, we define the *action* as the element of one pickup/replenishment operation, which contains the cart activity information. The information includes the cart ID, its location ID, and the type of action: replenishment, pickup, or load quantity.

Carts are scheduled to complete the order-picking and replenishment operations as soon as possible. However, because of the existence of picking and replenishment tasks in the same period, several delays occur and increase the difficulty of solving the problem. There are several types of delays experienced by carts.

1) Loading Queue Delay (D_{lq}): This type of delay occurs when carts must load and unload products at the same site and at the same time. These interactions are not only variable but also unpredictable and they can cause a cart to diverge from its base time significantly. If the two carts have to do the same operations (replenishment or picking) at the same location in the same period, the carts operate in the order in which they arrive, that is, first in,

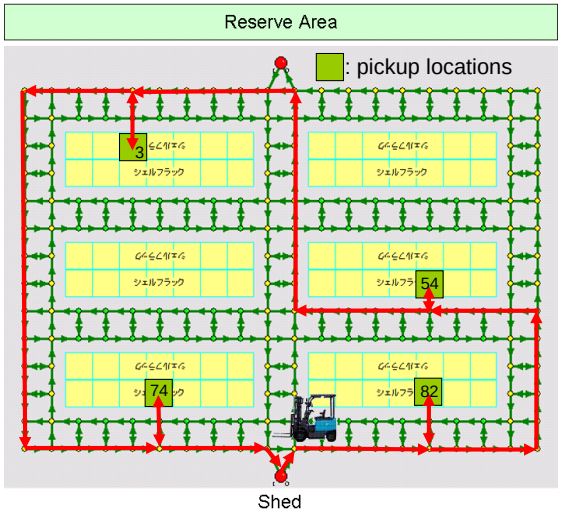


Fig. 2 An example of a cart routing trip

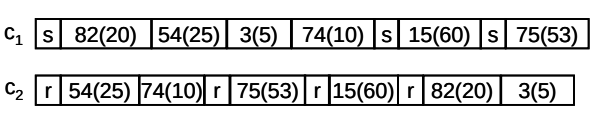


Fig. 3 Cart schedule example

first out. The waiting time is called the loading queue delay time.

2) Replenishment Wait Delay (D_{rw}): This type of delay occurs when a picking cart waits for the replenishment at the sites in short until the replenishment cart finishes its task. The waiting time is referred to as the replenishment delay.

3) Punishment Loading Delay (D_{pl}): This type of delay occurs when the number of products after replenishment exceeds the capacity at the storage location. The load or pick time for the products that exceeds the capacity takes N_{pu} (>1) times than the standard load or pick time. This type of delay is referred to as a punishment loading delay.

4) Collision Avoidance Delay (D_{ac}): This type of delay occurs when a cart requires extra time to avoid a collision with another cart.

As the problem is stated in this paper, the density of carts is relatively low and because mutual avoidance ability of cart drivers is so high, the degree of delay is not so large even two carts go through each other. Therefore, the fourth collision avoidance delay remains small compared to the total operation time and other delays. Thus, we ignore the collision avoidance delays and focus on the other three types of delays.

The problem is presented as follows. Let:

$C = \{c_1, c_2, \cdots, c_k\}$: a set of carts

$A_i = \{a_{i1}, a_{i2}, \cdots, a_{in_i}\}(i \in C)$: a set of actions of the cart i

k : total number of carts

n_i : total number of actions assigned to cart i $(i \in C)$

μ_i : operation time of cart i with delay time $(i \in C)$

$\max_{i \in C}(\mu_i)$: total makespan

Objective function: $M \rightarrow \min$.

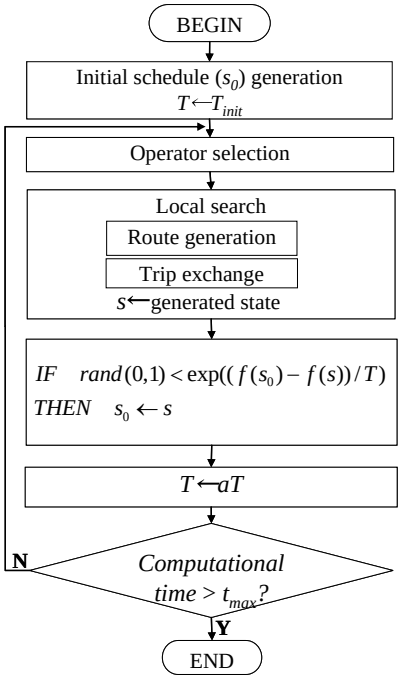


Fig. 4 Flowchart of the proposed scheduler

3. Research Methodology

3.1 Approach

Scheduling is defined as the allocation of carts to a collection of tasks. The transition from one solution to another can improve the efficiency of finding a better solution. In most scheduling problems, tasks have been determined beforehand and never change in the scheduling process. In the field of split delivery vehicle routing problems (SDVRP), the concept of task splitting is introduced for deriving solutions [11]. Similarly, in the paper, we propose several operators introducing task splitting during the search process.

In many warehouses, orders required for a specific day are just fixed several minutes before the warehouse opens. Then the planned/scheduled tasks must be finished in several minutes. Thus, maximum computation time t_{max} is an important condition.

The metaheuristic based on Simulated Annealing (SA) is described in this section. First, we introduce the Simulated Annealing structure, and then, a general heuristic to generate an initial schedule is introduced in paragraph 3.3. Operators to move action from cart to cart are described in detail in paragraph 3.4. Local Search as a procedure will be proposed in paragraph 3.5; it aims to minimize the routing cost and reduce the delay caused by interaction between carts.

3.2 SA Scheduler Structure

Simulated Annealing is a popular metaheuristic that has found extensive use for solving complex combinatorial optimization problems [13]. In SA, the search begins at an initial temperature T_{init} and proceeds until the stopping conditions are satisfied. We used one stopping condition when the current computation time reached the maximal computation time t_{max}. The flowchart of the SA scheduler is shown in detail in **Fig. 4**.

The algorithms about initial schedule generation, operator selection, and local search will be discussed in detail in paragraphs 3.3, 3.4, and 3.5, respectively.

3.3 Initial Schedule Generation

The initial schedule is generated by a strategy that is widely practiced due to its ease of implementation. The current scheduler generates trips by clustering the orders as a modified S-shaped pattern explained in Section 4.1. The products are picked up or replenished based on an S-shaped routing [14], which involves a round trip to all of the shelves, shed, and AS/RS, as shown in **Fig.**

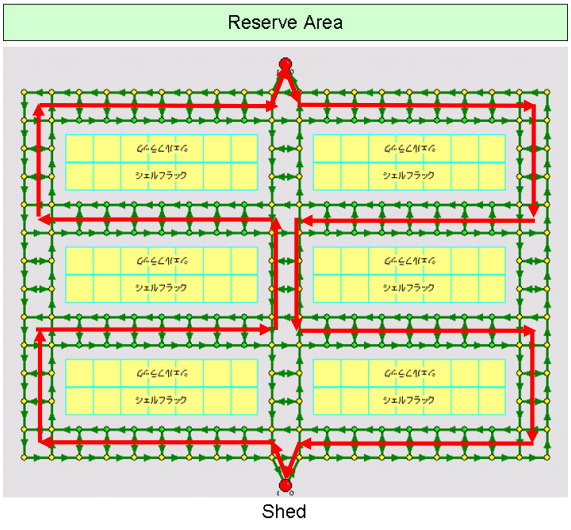


Fig. 5 An example of S-shape route

5. If the quantities to be picked up during the trip will exceed the capacity of cart (CAP_a) at the next picking location, the cart will stop picking and will return to the shed or AS/RS. At the beginning, all the carts start at the AS/RS to replenish products. The next generated trip will be assigned to the cart that returns first. This is repeated until there are no more unassigned replenishments in the warehouse. At that time, the carts tasked with replenishment will be reassigned to pick-up trips from the last site of replenishment. If there are no more picking trips to be assigned, no further activity is required.

3.4 Operator Selection

The operators in a scheduling problem relocate actions from one cart to another. The three types of operators are defined below: *Action Transfer* (total transfer), *Action Exchange* (total action swap), and *Action Split* (partial transfer).

1) Action Transfer: Action Transfer is a move wherein A_{ik} of C_i is transferred from C_i to another cart C_j. The transfer is done such that a new trip is constructed for A_{ik} at the end of the action sequence of C_j. Here, *i, j,* and *k* are randomly selected.

2) Action Exchange: An Action Exchange is a move in which the A_{ik} from C_i is relocated to another C_j. Similarly, the A_{jl} of C_j is relocated to C_i. After this operation, both C_i and C_j will add one new trip containing A_{jl} and A_{ik} at the end of the respective action sequences. Here, *i, j, k,* and *l* are randomly selected.

3) Action Split: If actions are split, a better balance can be achieved among the carts. The Action Split operator is an extension of the Action Transfer that allows partial relocation of an action from one cart to another, as shown in **Fig. 6**.

We can estimate the amount to items i^* of A_{ik} to be moved from C_i to another cart C_j in order to achieve a better *makespan* balance between two carts.

$$i^* = \frac{\mu_i - \mu_j - T_p}{2T_{load}} \quad (1)$$

In (1), μ_i and μ_j are the *makespans* of C_i and C_j before the action split operation. T_p is the increased travel cost of the new trip of a new split action, which is moved to C_j. T_{load} is *the Loading time per one case*. Equation (1) is constructed to create a balance between μ_i and μ_j.

We propose the Split_I methodology to implement the above three operations. The algorithm will be explained as follows.

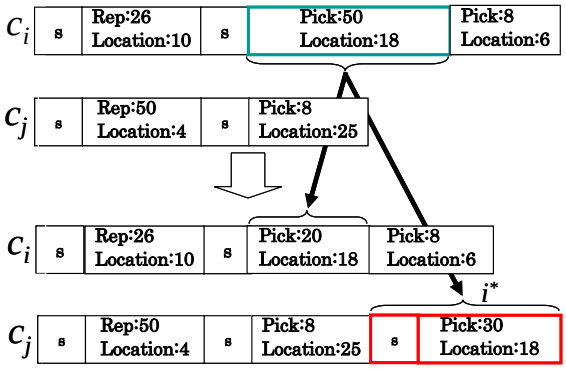


Fig.6 Action split operation that can produce a better makespan

Table 3 Four methods

Swap Condition (Step3) \ Split Condition (Step1)	Split	Non-split
Decided by *i**	Split_I	Non-split_I
Randomly Decided	Split_R	Non-split_R

Step 1: Randomly select *i*, *j*, *k*, and *l*. Calculate i^*, if $0 \le i^* \le num_{ik}$ (the number of items in A_{ik}), and then go to **Step 2**. Otherwise, go to **Step 3**.

Step 2: Create a new action A_{ik} for i^* and set the new action to C_i, and go to **Step 4**.

Step 3: If $i^* > num_{ik}$, go to **Step 4**. Otherwise, go to **Step 5**.

Step 4: Do Action Transfer to C_j. End.

Step 5: Generate A_{jl} in C_j, Do Action Exchange. End.

In the Split_I and Non-split_I, we make a decision to select the action exchange or the action transfer by utilizing i^* at Step 3. On the other hand, in Split_R and Non-split_R, we select the action exchange and the action transfer randomly. We can consider other classes to include the split operation or not. Split_I and Split_R include the split operation, and Non-split_I and Non-split_R do not, and we proceed from Step 1 to Step 3, skipping Step 2. The relationship of the four methods is shown in **Table 3**.

3.5 Local Search

The proposed operators just described are easy to implement. However, these operators will cause the following two problems.

a) The travel distance for the related carts will be affected and result in longer *makespans*.

b) Some delays may occur in the unchecked new schedule.

The next two parts deals with "a)" and "b)" respectively.

The first problem is to determine how to change the sequence of the actions and how to cluster the actions into some trips to minimize the traveling cost.

After the routing problem has been solved, the next step is to minimize the delays caused by the interactions between carts.

Trip Exchange is a simple process that aims to minimize the delays by exchanging the trip sequence inside the corresponding cart. The Trip Exchange is different according to the type of delay.

1) Determine which trip causes loading queue delays and exchange this trip with the last trip.
2) Determine which replenishment trip to remedy a product shortfall produces a delay (causing replenishing waiting delay) and exchange that trip with the first trip.

4. SIMULATION AND RESULTS

4.1 The Reference Scheduler Here, for a comparison, we introduce the reference scheduler that is currently employed by the industrial partner in this research [15]. The strategy is selected

Table 4 Simulation environment

Item	Quantity
Cart speed	1 meter/second
Loading time per one case (T_{lead})	1 second/case
N_{pu}	2
Average stock	30 cases/location
Cart number	4
Location size	1×1 m^2
Number of locations	84
Location capacity (CAP_s)	60 cases
Cart capacity (CAP_a)	60 cases
Loading time at shed or AS/RS	60 seconds
T_{init}	1000
A	0.9

due to the following two reasons: (a) there have been no former algorithms in the literature that deal with the same problem statement in this paper, and (b) the problem is still solved in real warehouses with a specific rule-based algorithm. Therefore the proposed algorithm is modeled as the reference scheduler in order to evaluate the proposed algorithm quantitatively. The reference scheduler is similar to the heuristic of the Initial Schedule mentioned in Section 3. The carts based on the S-shaped routing will add products to the trip until the capacity of the trip is reached. The remaining products will be picked up on the next trip. When order-picking starts, all carts will start from the AS/RS to unload replenishments. The picking tasks will not start until all replenishing tasks are finished. The advantage of this scheduler is that the Replenishment Waiting Delay will not occur because stock products are sufficient after replenishing.

4.2 Simulation Results

First, we evaluate the performance of the metaheuristic-based SA scheduler with different operators in computer experiments. The maximal computation time t_{max} is set to be 3 minutes under the environment of a Pentium4 1.2GHz with 1GB RAM. **Table 4** shows the simulation environment. The input orders are assigned with respect to *shops* that represent the regions to which products are delivered. A comprehensive set of data instances is used to evaluate the schedulers. The basic data instances are 3 data as follows:

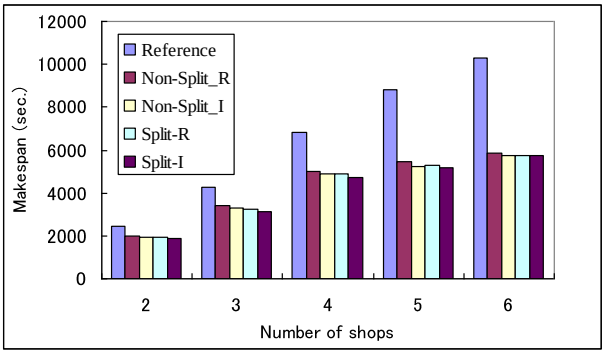
- Instance 1: 720 cases/shop distributed in 75 locations
- Instance 2: 720 cases/shop distributed in 45 locations
- Instance 3: 720 cases/shop distributed in 30 locations

These data are created based on the orders of real warehouses. The number of shops is set at 2 to 6. Restriction in generating trips is as follows:

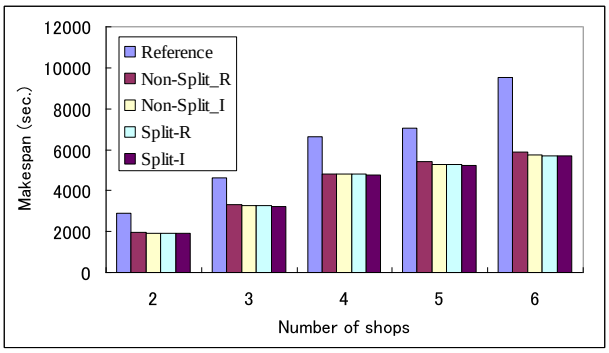
- For the picking tasks, trips must consist of products from the same shop.
- For the replenishment tasks, no restriction exists as to the products in the same trip.

Figure 7 shows the average values of *makespan* (in seconds) with input of Instance 1 to 3. All four sets of operators showed better performances than the reference scheduler. If we compare Split-I with the reference scheduler, a significant improvement (21.9% - 45.3%) is achieved. Both Split_R and Split_I show better performance than Non-Split_R and Non-Split_I. Comparing with Non-Split, Split can achieve, on average, a 2.4% improvement. Thus, the Action Split operator is proved to be efficient to balance the cart's makespans. Both Non-Split_I and Split_I show better results than Non-Split_R and Split_R. Especially, Split_I achieved the best result, i.e., a 1.9 % improvement, compared with nonSplit_R. It is a feasible to determine the operator by calculating the i^*.

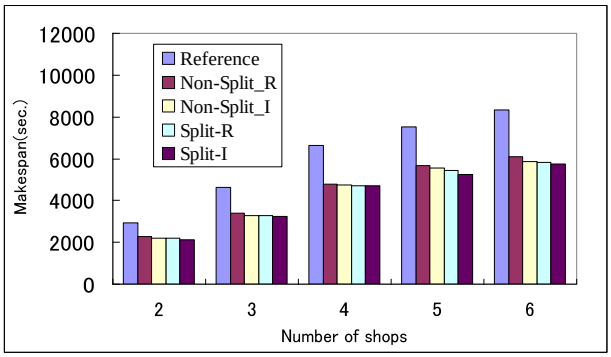
Then, we evaluate the improvement of 3 instances by increasing the number of shops. The trend is for the makespans of the reference scheduler to increase in a linear manner as the number of items increases. Split-I can achieve more improvement by increasing the number of shops. It is possible that, as the picking intensity increases, the items tend to concentrate on several specific locations and loading que delay will increase. Under this condition, the proposed metaheurstic is more efficient to reduce the delays and results of minimizing the makespans. The improvement of 3 instances does not show much difference when the number of shops is low (e.g., 2 or 3). However, when the number of shops increases (e.g., when the number of shops is 5 or



(a) For Instance 1



(b) For Instance 2



(c) For Instance 3

Fig. 7 Makespans of 5 schedulers with respect to the number of shops

6), the low-density instance (Instance 1) shows better improvement than the high-density instance (Instance 3).

In a multi-shop picking environment, orders from different shops with a few items each will be combined into one picking instance to improve the picking efficiency. When the number of shops is not large, combining their orders into one picking event will not have a significant effect. On the other hand, as the number of shops increases, combination of the order can produce a high efficiency.

The obtained results of Split-I method has varied even with the same condition because it is a kind of a random-search method. In order to see the influence of the variance, simulations are made for Split-I with 30 times when the number of shops is 2 and in the case of Instance 1, in which the difference between the result of Split-I and that of the reference sheduler is smallest. As a result, the standard score of the value for the reference scheduler has the standard score of -13.6, which means that the result of Split-I and that of the reference schduler is significantly different.

The reason can be concluded from the comparison of three delays, total D_{lq}, total D_{pl}, and total D_{rw} in the reference scheduler and Split-I. This is the result when the number of shops is 2 and in the case of Instance 1 (**Figure 8**). In the reference scheduler, although there are no D_{rw} in the reference scheduler because of *first replenishment next picking policy*, many D_{lq} and D_{pl} make the *makespan* longer. The reasoning behind this improvement can be applied to the following items.

- The proposed algorithm can produce a good schedule with shorter makespans.
- Local Search has succeeded in restraining D_{lq} and D_{pl} through Trip Exchange.

For other instances and for other number of shops, almost the same tendency can be obtained with the above-mentioned discussions.

5. CONCLUSION

In this paper, the problem of ordering-picking with replenishment has been addressed.

This research aims to generate the agents' scheduling and make the agents finish the tasks as soon as possible, that is, minimizing makespans. An efficient search-based metaheuristic is proposed because it is known to give optimal or near-optimal results.

In the metaheuristic, the transition from one solution to a new solution is a serious problem, especially in a complex system. Thus, we presented operators that aim to implement the transition from one schedule to another by way of relocating (moving) the actions between agents. Three operators, Action Transfer, Action Exchange, and Action Split, are identified. All operators are involved in the transfer of action between agents. The split and swap conditions are two determinations with which we are concerned. According to the two conditions, we conclude that there are four sets of operators for the transition of a schedule.

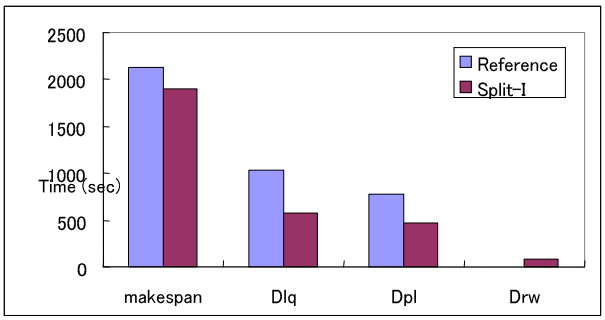


Fig. 8 Comparison of the Split-I scheduler and the reference scheduler for the case of Instance 1; the number of shops is 2.

Instances are arranged according to varying the number of shops. Many experiments have been conducted in different instances and in a class-based storage assignment condition. Experiment results proved that a kind of SA scheduler, Split-I had a little improvement than three other scheduler (Split-R, Non-Split-I, Non-Split-R). By comparison with the reference scheduler, the Split-I SA scheduler was efficient enough to balance the makespans between agents. Experimental results revealed that the Split-I SA scheduler can achieve a significant improvement (averaged about 31%) in comparison to the reference scheduler. The trend is for the average improvement increase as the number of shops increases. When the pickup intensity increases, so does the volume of stock in one location, and, as a result, the greater the possibility of a long delay. Under this condition, the proposed metaheuristic is more efficient to reduce the delays and result of minimizing the makespans.

References

[1] Tompkins, J.A., et al.: Facilities Planning, John Wiley & Sons,(1996).

[2] De Koster, R., Le-Duc, T., and Roodbergen, K.J.: Design and Control of Warehouse Order-picking: A Literature Review, European Journal of Operational Research, **182**, 481/501 (2007).

[3] Rouwenhorst, B., Reuter, B., Stockrahm, V., Van Houtum, G.J., Mantel, R.J., and Zijm, W.H.M.: Warehouse Design and Control: Framework and Literature Review, European Journal of Operational Research, **122**, 515/533 (2000).

[4] Levi, R., and Sviridenko, M.: Improved Approximation Algorithm for the One-Warehouse Multi-Retailer Problem, APPROX and RANDOM 2006, Diaz, J. et al. (Eds.), Lecture Note on Computer Science, **4110**, 188/199 (2006).

[5] Minner, S., and Silver, E.A.: Replenishment Policies for Multiple Products with Compound-Poisson Demand that Share a Common Warehouse, International Journal of Production Economics, **108**, 388/398 (2007).

[6] Choi, J., Cao, J.J., Romeijn, E.R., Geunes, J., and Bai, S.X.: A Stochastic Multi-Item Inventory Model with Unequal Replenishment Intervals and Limited Warehouse Capacity, IIE Transactions, **37**, 1129/1141 (2005).

[7] Caron, F., Marchet, G., and Perego, A.: Optimal Layout in Low-Level Picker-to-Part Systems, International Journal of Production Research, **38**-1, 101/107 (2000).

[8] Van den Berg, J.P., Sharp, G.P., Gademann, A.J.R.N., and Pochet, Y.: Forward-Reserve Allocation in a Warehouse with Unit-Load Replenishments, European Journal of Operational Research, **111**, 98/113 (1998).

[9] Roodbergen, K.J., and De Koster, R.: Routing Order Pickers in a Warehouse with a Middle Aisle, European Journal of Operational Research, **133**, 32/43 (2001).

[10] Makris, P.A., and Giakoumakis, I.G.; *k*-Interchange Heuristic as an Optimization Procedure for Material Handling Applications, Applied Mathematical Modeling, **27**, 345/358 (2003).

[11] Rubrico, J.I.U., Ota, J., Tamura, H., Akiyoshi, M., and Higashi, T.: Route Generation for Warehouse Management using Fast Heuristics,

Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2093/2098 (2004).
[12] Kim, B., Heragu, S.S., Graves, R.J., and St. Onge, A.: Realization of a Short Cycle Time in Warehouse Replenishment and Order Picking, International Journal of Production Research, **41**- 2, 349/364 (2003).
[13] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.: Optimization by Simulated Annealing, Science, **220**, 671/680 (1983).
[14] Roodbergen, K.J., and De Koster, R.: Routing Methods for Warehouses with Multiple Cross Aisles," International Journal of Production Research, **39**-9, 1865/1883 (2001).
[15] Gong, J., Ota, J., Tamura, H., and Higashi, T.: A Model and Efficient Heuristics of Order-Picking with Replenishment in a Warehouse, Preprints of the 16th Intelligent Systems Symposium, 161/166 (2006).

Jie Gong

Mr. Jie Gong graduated from master course of Graduate school of Engineering, the University of Tokyo in 2007. His research interest was warehouse management system and logistics during his stay in the university.

Hirofumi Tamura

Mr. Hirofumi Tamura works for Murata Systems, Ltd. He is engaged in research and development on logistic information systems.

Toshimitsu Higashi (Member)

Dr. Toshimitsu Higashi works for Murata Machinery Ltd. He is engaged in research and develoment on the control system of logistic systems, sway control systems and distributed autonomous robotic systems.

Jun Ota (Member)

Professor Jun Ota is an associate professor at Dept. of Precision Engineering, Graduate School of Engineering, the University of Tokyo. His research interest is multi-agent robotics and logistic systems.