Graph-based heuristics for operational planning and scheduling problem in automatic picking system

Shinji IMAHORI* and Yohei HASE**
* Faculty of Science and Engineering, Chuo University
1-13-27 Kasuga, Bunkyo-ku, Tokyo, 112-8551, Japan
E-mail: imahori@ise.chuo-u.ac.jp
** Graduate School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi, 464-8603, Japan

Received 12 November 2015

Abstract
In this paper, we study an operational planning and scheduling problem in an automatic picking system. This problem was introduced as a practical benchmark problem arising in logistics and involves assignment and scheduling tasks. We first show the NP-hardness of the problem. We then propose a graph-based heuristic algorithm for computing a good schedule of requests for a given assignment of products. The computational results for benchmark instances show that good schedules of requests are obtained in short computation time.

Key words: Scheduling, Automatic picking system, Logistic industry, Graph theory, Combinatorial optimization

1. Introduction

Recently, reduction of working time is needed in the logistics industry because of intense price competition. Delivery requests from retails, including convenience stores, are mainly small quantity and large variety. For high quality service in this situation, good mathematical models of practical problems and sophisticated algorithms should be developed. An automatic picking system sorts products in a distribution center. In this paper, we study an operational planning and scheduling problem in the automatic picking system. This problem was introduced by Kondo et al. (2012) as a practical benchmark problem arising in logistics. The problem is known to be very difficult because it involves both of assignment and scheduling tasks. We explain the details of this problem in the next section, and prove the NP-hardness of the problem.

In general, for difficult (NP-hard) optimization problems, it is impractical to persist in an optimal solution (i.e., developing exact algorithms). Therefore, we need some approximation algorithms for difficult problems, which run fast and compute good solutions. Metaheuristics is one of the promising techniques for developing such practical algorithms and has been successfully used for many combinatorial optimization problems. However, even though metaheuristic techniques are utilized, finding good solutions is not easy without the knowledge on the problem.

For the operational planning and scheduling problem in the automatic picking system, Iima et al. (2015) proposed an efficient algorithm to solve a subproblem of the original problem; when an assignment of products into lanes and a schedule of requests to be processed are given, the algorithm determines how to operate the automatic picking system optimally. By using this algorithm, the search space can be decreased effectively and it becomes easier to search for a good assignment of products and a good schedule of requests by some strategies (such as local search and metaheuristics).

In this paper, we propose a graph-based heuristic algorithm that computes a good schedule of requests for a given assignment of products into lanes. By combining this algorithm with a heuristic algorithm, including local search and metaheuristics, for searching for good assignments of products into lanes, it is expected to find a good solution of the original problem in short computation time. To evaluate the proposed algorithm in practical sense, computational experiments with benchmark instances are conducted and the computational results are reported.
2. Automatic picking system

In this section, we explain the operational planning and scheduling problem in an automatic picking system introduced by Kondo et al. (2012). Readers could refer to papers by Iima et al. (2015) and Kondo et al. (2012) for more details. We then prove the NP-hardness of the problem in Section 2.3.

2.1. Problem description

As shown in Fig. 1, an automatic picking system is composed of a set of lanes and a conveyor. Products, symbols such as ⊙ and △, are stacked in the lanes. Each product belongs to one of the product types, and products of the same product type can be stacked in a lane. The conveyor is located in front of lanes, and it moves from left to right. Each lane can supply one product on the conveyor in front of the lane once a second.

![Fig. 1 Automatic picking system.](image)

There are many shops that send multiple requests to a distribution center. The automatic picking system in the distribution center sorts products according to requests from shops. We note that all the requests from a shop must be processed sequentially (i.e., requests from different shops cannot be mixed). We summarize the problem as follows and the notations in Table 1.

- There are multiple lanes with the same width $W$.
- A lane is filled by products with a product type.
- Products in a lane are not out of stock.
- Products of the same product type can be allocated in multiple lanes.
- Each lane can supply one product with one second intervals on the conveyor in front of the lane.
- Supply of products is synchronized with the lanes.
- Products cannot be piled.
- The conveyor moves from left to right with speed $\alpha W$.
- The interval between products belonging to different requests is at least $\beta W$ on the conveyor.
- The distance between products belonging to the same request is less than $\beta W$ on the conveyor.
- Requests from a shop are processed sequentially.

| notation | definition |
|----------|------------|
| $N_L$    | number of lanes ($l = 1, 2, \ldots, N_L$), |
| $N_P$    | number of products, |
| $N_K$    | number of product types ($k = 1, 2, \ldots, N_K$, where $N_K \leq N_L$), |
| $N_S$    | number of shops ($s = 1, 2, \ldots, N_S$), |
| $N_R(s)$ | number of requests belonging to shop $s$ ($s = 1, 2, \ldots, N_S$, $r = 1, 2, \ldots, N_R(s)$), |
| $W$      | width of a lane (hereafter $W = 1$), |
| $\alpha$ | speed of the conveyor, |
| $\beta$  | minimum interval between different requests, where $\beta \geq \alpha$, |
| $T$      | total time to sort products. |
Figure 2 shows an example of how to operate the automatic picking system. Let $\alpha = 2$, $\beta = 3$ and a request with six products in the figure should be processed. At time $t' = 0$, one product ▲ for the previous request remains on the conveyor. At this moment, ○, △ and □ can be supplied from lanes but ◇ cannot due to the constraints. At $t' = 1$, both of the lanes with product type ◇ cannot supply a product due to the minimum interval constraint, the lane with △ also cannot supply a product because another product ○ is located on the conveyor in front of the lane. Thus, only a product ○ is supplied. No products can be supplied at $t' = 2$, and then, ◇ and △ are supplied at $t' = 3$. At this moment all products of the request are supplied on the conveyor and the distance between products in this request is less than $\beta W$.

We now summarize the input, output and the objective of the problem. Figure 3 shows an example of the delivery request card in which requests from all of the shops are summarized. In this figure, $s$, $r$, $k$ denotes the shop, the request, and the product type, respectively. Column $p$ denotes the number of products of the corresponding product type, request, and shop.

**Input**
- $N_L, N_P, N_K, N_S, N_R(s), \alpha$ and $\beta$,
- a delivery request card shown in Fig. 3.

**Output**
- assignment of products into lanes,
- schedule of requests,
- time for supplying products from each lane.

**Objective**
- minimizing the total time $T$ of operation to sort all products according to the requests.

| $p$ | $s$ | $r$ | $k$ |
|-----|-----|-----|-----|
| 6   | 1   | 1   | 3   |
| 3   | 1   | 1   | 5   |
| 2   | 1   | 2   | 1   |
| 2   | 1   | 2   | 3   |
| 3   | 2   | 1   | 4   |
| 2   | 2   | 1   | 6   |
| 4   | 2   | 2   | 4   |
| \* | \* | \* | \* |
| \* | \* | \* | \* |

**2.2. Grouping of lanes**

In the original setting of the problem, the automatic picking system equips $N_L$ lanes, and $N_K$ types of products are stacked in the lanes. For each lane, we decide when products in the lane are supplied to the conveyor. It is difficult to
understand the positions of products on the conveyor since products are supplied to different positions at different time points and the conveyor is moving. For intuitive understanding, we (originally Kondo et al. (2012)) assume that the conveyor expands to the upper stream (i.e., left in figures) infinitely, and use the converted locations of products at time $t = 0$. Under this assumption, instead of minimizing the total time $T$, we minimize the leftmost converted location of products at time $t = 0$.

It was explained in Section 2.1 that only products of the same product type can be filled in a lane, and each lane can supply products on the conveyor in front of the lane. This means that only one type of products can be supplied for a position on the conveyor. However, when we consider the converted locations, various products of different product types can be supplied at a position at time $t = 0$. Therefore, instead of assigning $N_k$ product types into $N_L$ lanes, we assign $N_k$ product types into $\alpha$ groups of lanes. If $N_L$ can be divided by $\alpha$, each group of lanes can accommodate $N_L/\alpha$ product types. This idea can reduce the solution space of the problem without sacrificing essentially different solutions. We note that, if a schedule of requests and an assignment of product types into $\alpha$ groups of lanes are given, it is easy to assign product types into appropriate lanes.

2.3. Complexity of the problem

The operational planning and scheduling problem in an automatic picking system problem was introduced as a hard benchmark problem arising in logistics. We show the NP-hardness of the problem with uniform graph partitioning problem (UGP in short), which is known to be NP-hard (Garey et al., 1976).

Uniform Graph Partitioning

**Input:** Graph $G = (V, E)$, where $|V| = 2n$, and a positive integer $K$.

**Question:** Is there a partition of $V$ into disjoint sets $V_1$ and $V_2$

such that $|V_1| = |V_2|$ and the number of edges between $V_1$ and $V_2$ is no more than $K$?

This problem is also known as minimum cut into bounded sets in general. If we consider the complement graph, we can see that it is also difficult to find a uniform graph partition such that the number of edges between two sets is at least $K$. We will transform this problem to the operational planning and scheduling problem in an automatic picking system.

Let $V = \{v_1, v_2, \ldots, v_{2n}\}$ be the set of nodes and $E = \{e_1, e_2, \ldots, e_m\}$ be the set of edges in an arbitrary instance of UGP (of maximizing edges between sets). We design the following instance of the operational planning and scheduling problem: $N_L = N_K = 2n$, $N_k = 1$, $N_k(1) = m$, $W = 1$, $\alpha = \beta = 2$. Each request $i$ has two types of products; if edge $e_i$ connects nodes $v_x$ and $v_y$, then request $i$ has $p$ products of type $x$ and $p$ products of type $y$, where $p > m$.

Now we consider a property of the operational planning and scheduling problem. As stated in the previous subsection, we consider $\alpha$ (in this case $\alpha = 2$) groups of lanes. For each request with product types $x$ and $y$, if $x$ and $y$ are stored in different lane groups, this request can be sorted in $p$ seconds; otherwise it takes at least $2p - 1$ seconds. Thus minimizing the total time $T$ of operation to sort all products of all the requests is equivalent to finding a uniform partition of graph nodes such that the number of edges between two sets is maximized. The uniform graph partitioning problem is NP-hard, and hence our problem is also NP-hard.

3. Assignment of products

In this section, we consider how to assign products into lanes (or groups of lanes). One possible strategy is random assignment; we assign $N_K$ product types into $N_L$ lanes randomly. Iima and Kawano (2014) applied random assignments repeatedly, and reported statistical computational results on benchmark instances. Another possible and promising strategy is local search (and metaheuristics). If we construct an appropriate initial solution and a neighborhood structure, it is expected to find a good solution by local search.

In this study, however, we apply a simple heuristic algorithm for finding an assignment of products into lane groups. One of the reasons why we adopt a simple heuristics for assignment of products is that our graph-based heuristic algorithm (which will be proposed in Section 5) for scheduling requests needs a certain computation time. Another reason is that we want to evaluate the performance of our graph-based heuristics; if we use a sophisticated search strategy for assignment of products, it is difficult to measure contributions of two new techniques (i.e., assignment of products and scheduling of requests).

We use the following simple heuristic algorithm. As explained in Section 2.2, instead of treating $N_L$ lanes, we consider $\alpha$ groups of lanes. If the number of lanes $N_L$ is greater than the number of product types $N_K$, some product types can be assigned to multiple lane groups. We first determine product types using multiple lanes by the following rule. If a
request contains many products of an identical product type, we use multiple lanes for this product type. A product type that appears in many requests is also assigned to multiple lane groups. We then assign products into lanes (more precisely, groups of lanes) randomly under the following two conditions: (1) the number of product types for each group of lanes is \( N_L / \alpha \) and (2) if a product type can be assigned to multiple lanes, it is assigned to different groups of lanes.

4. Time for supplying products

For a given schedule of requests and an assignment of products into lanes, we should decide when products are supplied to the conveyor from lanes. Iima et al. (2015) proposed a polynomial time algorithm that decides time points (or converted locations) of supplying products optimally. We note that they assumed that a lane group is designated for each product even if more than one lane groups can supply this product type. In this paper, we use a similar idea for computing when products are supplied from lanes and an ideal timing of processing for each request. The latter information is essential in our graph-based heuristic algorithm that computes a good schedule of requests for a given assignment of products.

Assume that we are given an assignment of products into lanes. As denoted in Section 2.2, we consider that products are assigned into \( \alpha \) groups of lanes. Let lane group \( i \) be the set of lanes whose lane ID is \( i \) (mod \( \alpha \)). We compute the necessary length of the conveyor (in other words, minimum time span) for a given request \( r \) from shop \( s \). For simplicity, we first consider the case that every product type in this request is assigned to only one lane.

We count the number of products for each lane group \( i \). The maximum number of products for a lane group has a strong influence on the minimum time span of this request. It is also important that which lane groups attain the maximum number of products for the request \( r \) from the shop \( s \). At this moment, it is not decided when this request is processed. That is, we do not know the lane from which we can sort products for this request. It is also not known the next request to be sorted. Thus, we consider the following possibilities.

Let \( d \) be the lane group from which we can start to sort products for a request, and \( u \) be the lane group at which all the products for the request are supplied on the conveyor. There are \( \alpha^2 \) possibilities for a pair of \( d \) and \( u \). We compute the necessary conveyor length \( C_{\text{min}}(s, r, d, u) \) for each combination of \( s, r, d \) and \( u \).

---

Fig. 4 How to compute \( C_{\text{min}}(s, r, d, u) \) (\( \alpha = 3 \); left: \( d = 2, u = 0 \), right: \( d = 0, u = 2 \)).

Figure 4 shows examples to explain how to calculate \( C_{\text{min}}(s, r, d, u) \). A product type □ is assigned to lane group 1. Product types □ and ● are assigned to lane group 2, and a product type △ is assigned to lane group 0 in this example. When we consider the case of \( \alpha = 3, d = 2 \) and \( u = 0 \), the conveyor with minimum length that attains \( d = 2 \) and \( u = 0 \) is depicted at Fig. 4 (1). However, we cannot put all the products of this request on this conveyor, and we extend the length of the conveyor with \( \alpha \) units as shown in Fig. 4 (2). We continue such extensions of the conveyor with \( \alpha \) units until all the products are located on the conveyor. Figure 4 (4) describes the minimum length conveyor satisfying a requirement, and \( C_{\text{min}}(s, r, 0, 2) = 11 \) for this case. We also show another example of how to compute the necessary conveyor length for \( d = 0 \) and \( u = 2 \) in this figure, and we have \( C_{\text{min}}(s, r, 0, 2) = 9 \). We note that the necessary conveyor length can be computed efficiently with the maximum number of products for a lane group and which lane groups attain it.

We compute \( C_{\text{min}}(s, r, d, u) \) for all the pairs of \( d \) and \( u \) as shown in Figure 5 for each request \( r \) from shop \( s \). Among all the pairs, we choose an ideal pair that attains the minimum value \( \min_{d, u} C_{\text{min}}(s, r, d, u) \). We choose \( C_{\text{min}}(s, r, 1, 1) = 7 \), \( C_{\text{min}}(s, r, 0, 0) = 2 \), and \( C_{\text{min}}(s, r, 2, 2) = 11 \).

---

[DOI: 10.1299/jamdsm.2016jamdsm0039]
for the case in Figure 5. It may happen that some combinations of \(d\) and \(u\) attain the minimum value. In this study, we choose a combination among them randomly. It may be possible to keep all the good combinations and use them when we construct a schedule of requests.

If some product types belonging to the current request are assigned to multiple lane groups, we need to compute the minimum time span and an ideal timing of processing for this request carefully. If \(\alpha\) is a small constant (\(\alpha = 4\) for benchmark instances we use in our experiments), it is easy to treat product types assigned to multiple lane groups.

5. Scheduling of requests

We propose a heuristic algorithm to decide schedule (i.e., processing order) of requests. We introduce the proposed model with a directed graph in Section 5.1 and explain our algorithm in Section 5.2.

5.1. Model with digraph and Eulerian path

We are given an assignment of products into lane groups, which is computed by the algorithm described in Section 3. For the assignment, every request computes its ideal timing for processing as explained in Section 4. By using this information, we determine the order of requests to be processed. As explained in Section 2.1, there are many shops, and requests from a shop must be processed sequentially. Thus, our algorithm first decides the processing order of requests for each shop. Moreover, we also compute the ideal timing of processing requests for each shop. By using them, we next determine the processing order of the shops. Algorithms for these two rounds are essentially the same and we only explain the algorithm for the first round.

Let us consider a shop \(s\) and its \(N(s)\) requests. To process all the requests, the sum of the minimum time spans for all requests is necessary. We also need time for the minimum interval \(\beta\) between requests. They are denoted as

\[
\sum_{r=1}^{N(s)} \min_{d,u} C_{\text{min}}(s, r, d, u) + \beta(N_{R}(s) - 1)
\]

This is a lower bound of the processing time for the shop and we design a schedule achieving (or close to) this value.

Let us focus on a request \(r\) of the shop \(s\). Figure 6 shows an example (where \(\alpha = \beta = 3\)). We are given a request with six products (\(\bigcirc, \bigcirc, \bigcirc, \Box, \triangle, \bullet\)); ideally we start the processing for this request at a lane of group 1 and finish it at another lane of group 1. If the last product of this request is supplied from a lane of group 1, we can supply products for the next request from a lane of group 2.

![Directed edge for a request.](image)

We describe this situation on a digraph \(G = (V, E)\). Each vertex \(v \in V\) denotes a lane group. Thus there are \(\alpha\) vertices. For a request, we put a directed edge; the tail of this edge denotes the lane group where we want to put the first product,
and the head denotes the lane group where we can supply products for the next request. We put such edges for all the requests of a shop on a graph $G$.

If a directed graph contains an Eulerian path (i.e., one-stroke sketch for all the edges), we call it Semi-Eulerian graph. (We note that Eulerian graph contains an Eulerian cycle.) If the digraph $G$ for a shop is a Semi-Eulerian graph, we can find a schedule of requests for the shop with the time span denoted as Eq. (1) by using an Eulerian path. Namely, if the requests are served along the Eulerian path, then all the intervals between two requests become $\beta$.

### 5.2. Graph-based heuristic algorithm

In general, however, the graph $G$ for a shop does not contain an Eulerian path. In this case, it is not possible to schedule requests with the minimum time span denoted in Eq. (1). In this subsection, we explain how to decide a schedule of requests for this case. We note that it is easy to check a graph contains an Eulerian path or not by checking degrees of vertices and connectivity of the graph (Bondy and Murty, 2008).

If we put an extra space between two requests, we can change the start point to serve a request. For example, in Fig. 6, we have a request with six products that occupies the conveyor from a lane of the first group to another lane of the same group as depicted with a dotted box. When we start to serve the next request with the minimum interval $\beta$, a lane of group 2 is the first position for the next request. With an extra space between this and the next requests, we can start to serve the next request from a lane of group 0.

This extra space can be represented as an extra edge on $G$; if the tail of an edge is $d$, then the head is $d + 1$ for $d = 0, 1, \ldots, \alpha - 1$ where vertex $\alpha$ means vertex 0. All the extra edges consist a directed cycle with $\alpha$ vertices. Each extra edge has length one because it denotes one extra space between requests. See Fig. 7 for an example. The given digraph Fig. 7 (a) is not a Semi-Eulerian graph. Figure 7 (b) shows possible extra edges, i.e., a cycle for $\alpha$ nodes. By adding two extra edges (dotted edges in Fig. 7 (c)) to the graph, the resulting graph contains an Eulerian path.

![Fig. 7 Adding extra edges to make a graph Semi-Eulerian: (a) given graph that does not contain an Eulerian path, (b) cycle of extra edges, (c) Semi-Eulerian graph that contains all the given edges and some extra edges.](image)

We propose an algorithm to make a directed graph Sub-Eulerian with the smallest number of extra edges. We first add the smallest number of extra edges to make the in-degree of a vertex is equal to the out-degree of the vertex for all vertices. If the resulting graph is unconnected, we add a directed cycle (i.e., $\alpha$ extra edges) to make the graph connected; this graph always contains an Eulerian cycle. Finally, we remove a longest path (whose length is at most $\alpha - 1$) that contains only extra edges under the constraint that the resulting graph keeps the connectivity.

At the end of this section, we summarize our graph-based heuristic algorithm and analyze its time complexity. For a given data (i.e., a delivery request card and parameters), we first assign products into $\alpha$ lane groups as explained in Section 3. We then compute an ideal pair $(d, u)$ that attains the minimum value $\min_{d, u} C_{\min}(s, r, d, u)$ for each request $r$ of shop $s$ as explained in Section 4. We use randomness to choose a pair $(d, u)$ if there are two or more candidates. These two steps need linear time of the size of the input data. To decide a schedule of requests, we first compute an ideal sequence of requests for each shop $s$. That is, a graph with $\alpha$ nodes and $N_{s}(s)$ edges is constructed, and then we make it a Semi-Eulerian graph and find an Eulerian path. If we use a multigraph without weight, it takes $O(\alpha N_{s}(s))$ time for shop $s$; hence we need $O(\alpha \sum_{s} N_{s}(s))$ time in this part. Finally, we decide an order of shops in a similar way with $O(\alpha N_{s})$ time. When we are given an assignment of products into $\alpha$ lane groups and a schedule of requests and shops, it is easy to assign products into lanes in $O(N_{L})$ time appropriately.

### 6. Computational results

We now report computational results obtained by our algorithm. The algorithm was implemented in the C programming language and ran with a 3.1 GHz Intel Core i5-2400 processor and 8 GB memory. Our algorithm was tested on
Table 2 Objective values of the proposed algorithm.

| instance | random | CP   | proposed (avg.) | proposed (best) |
|----------|--------|------|-----------------|-----------------|
| 00       | 4132.25| 4149.5| 3887.80         | 3835.50         |
| 01       | 4020.00| 4047.0| 3598.48         | 3487.00         |

Table 3 Surplus values for each task.

| instance | assignment | scheduling |
|----------|------------|------------|
|          | requests   | shops      |
| 00       | 577.98     | 57.65      | 4.18       |
| 01       | 275.03     | 65.63      | 9.83       |

benchmark instances (called 00 and 01) provided on the website of optimization benchmark problems for industrial applications. These two instances have the following common parameter values: \( N_L = 100, N_P = 9000, N_K = 90, N_S = 50, \alpha = 4 \) and \( \beta = 8 \). Instance 00 is a uniform instance with \( N_P(s) = 10 \) for every shop \( s \) (i.e., there are 500 requests in total). Instance 01 is a diversified instance with \( 1 \leq N_P(s) \leq 36 \), where the total number of requests is same as instance 00. These two instances have the same simple lower bound values 3248, where the simple lower bound is calculated by

\[
(N_P + \beta \sum_{s} N_P(s) - 1)/\alpha.
\]

For more details of these benchmark instances, readers could refer to the website (we give its URL in the list of references).

We ran our algorithm ten times for each instance with different random seeds. We show the computational results in Table 2. In Table 2, the column of random is the best objective values among 100,000 random assignments of products and scheduling requests reported by Iima and Kawano (2014). The column of CP shows the best objective values using a constraint programming model by Miyamoto et al. (2015). The columns of proposed are the average and the best objective values by our algorithm. The average computation time of our algorithm was 0.09 seconds for instance 00 and 3.65 seconds for instance 01.

For both of instances, our algorithm generated the best solutions in very short computation time. Our heuristic algorithm was especially effective for instance 01. For more detailed analysis, we evaluated surplus values for each task and results are reported in Table 3. As explained in Section 3, we use a simple heuristic algorithm for assignment of product types into lanes. For an assignment, the minimum time span for each request is computed as denoted in Section 4. Here we define the surplus of a request for an assignment by

\[
(\text{minimum time span}) - (\text{number of products})/\alpha.
\]

The column of assignment in Table 3 shows the sum of surpluses for all the requests (here we report the average values of 10 runs with different random seeds). We can observe that the average surplus for instance 01 is less than half of that for instance 00. This difference comes from the characteristics of two instances. Instance 00 is a uniform instance and each product type appears in the delivery request card uniformly. On the other hand, in instance 01, some product types are requested frequently; for instance, the most popular product type accounts 7%, and top five product types among 90 product types account more than 30% of all the products. Our heuristic algorithm in Section 3 assigns multiple lanes to such product types, and it makes the surplus of this part small.

The columns of scheduling show extra space between requests and shops. Namely, we compute the extra space between requests for each shop with the algorithm in Section 5.2, and report the sum of them in the column scheduling–requests. We also report the sum of extra spaces between shops (there are 50 shops in these benchmark instances) in Table 3. If we decide a schedule of requests randomly, the expected values of surplus are 168.75 for requests and 18.375 for shops. From these computational results, we can say that the proposed graph-based heuristic algorithm computes a good schedule of requests for each shop and a good schedule of shops.

7. Conclusions

We considered an operational planning and scheduling problem in an automatic picking system proposed by Kondo et al. (2012). It is a benchmark problem arising in practical logistics to evaluate heuristic and metaheuristic algorithms. In this paper, we first proved the NP-hardness of the problem. For a given assignment of products into lanes, we proposed a graph-based heuristic algorithm for computing a good schedule of requests and shops. The computational results for benchmark instances showed that good solutions are obtained in very short computation time by our algorithm.
As future directions of research, we are planning to devise an efficient method of searching for good assignments of products. Local search and metaheuristics may successfully find good assignments. Other heuristic techniques using the characteristics of the problem also seem to be effective.

References

Bondy, J.A. and Murty, U.S.R., Graph Theory (2008), Springer-Verlag, London.
Garey, M.R., Johnson, D.S. and Stockmeyer, L., Some simplified NP-complete graph problems, Theoretical Computer Science, Vol. 1 (1976), pp. 237–267.
Iima, H. and Kawano, Y., Operational planning and scheduling benchmark problem in an automatic picking system, Proceedings of National Convention of the Institute of Electrical Engineers of Japan, Vol. 4 (2014), pp. S21(9–12) (in Japanese).
Iima, H., Kawano, Y. and Koguma, Y., Operational planning and scheduling benchmark problem in an automatic picking system, IEEJ Transactions on Electronics, Information, and Systems, Vol. 135 (2015), pp. 1270–1278 (in Japanese).
Kondo, C., Kawano, Y. and Iima, H., An optimization benchmark problem for automatic picking system operational planning and scheduling, Proceedings of Technical Meeting C of the Institute of Electrical Engineers of Japan, (2012), pp. 326–331 (in Japanese).
Miyamoto, T., Mori, K., Kitamura, S. and Izui, Y., Constraint programming model for operational planning and scheduling problem in automatic picking system, Proceedings of International Symposium on Scheduling, (2015), pp. 163–168. Optimization benchmark problems for industrial applications (online), available from <http://www.ssl.te.chiba-u.jp/~okamoto/BP-IA/en/>, (accessed at 12 November, 2015).