International Journal of Management Science
Vol 12, No 1, May 2006

A Two-Stage Heuristic for Disassembly Scheduling with Capacity Constraints[*]

Hyong-Bae Jeon
Department of R&D Planning and PR. Korea Institute of Machinery and Materials
Yusong-gu, Daejon 305-343, KOREA

Jun-Gyu Kim
Department of Industrial Engineering, Hanyang University
Sungdong-gu, Seoul 133-791, KOREA

Hwa-Joong Kim
Institute of Production and Robotics (STI-IPR-LICP)
Swiss Federal Institute of Technology (EPFL)
Lausanne, CH-1015, SWITZERLAND

Dong-Ho Lee[**]
Department of Industrial Engineering, Hanyang University
Sungdong-gu, Seoul 133-791, KOREA

(Received Sep. 2005; Revised Nov. 2005; Accepted Nov. 2005)

ABSTRACT

Disassembly scheduling is the problem of determining the quantity and timing of disassembling used or end-of-life products while satisfying the demand of their parts and/or components over a planning horizon. The case of assembly product structure is considered while the resource capacity constraints are explicitly considered. A cost-based objective is considered that minimizes the sum of disassembly operation and inventory holding costs. The problem is formulated as an integer programming model, and a two-stage heuristic with construction and improvement algorithms is suggested in this paper. To test the performance of the heuristic, computational experiments are done on randomly generated problems, and the results show that the heuristic gives near optimal solutions within a very short amount of computation time.

Keywords: Disassembly Scheduling, Capacity Constraints, Two-Stage Heuristics

[*] This research was supported by Korea Research Foundation Grant (KRF-2004-03-D00468) funded by Korea Government. This support is gratefully acknowledged.
[**] Corresponding author, Email: leman@hanyang.ac.kr

1. INTRODUCTION

For the last decades, growing concerns on environmental issues have stimulated the industry to develop various material and product recovery processes. Disassembly, one of the essential material and product recovery processes, is the process of separating used or end-of-life products into their constituent parts, subassemblies, or other groupings. Due to its importance in material and product recovery, previous research has been done on various disassembly problems such as design for disassembly, disassembly process planning, and disassembly scheduling. For literature reviews on these problems, see Boothroyed and Alting [1], Jovane *et al.* [4], Lambert [10], Lee *et al.* [13], O'shea *et al.* [18], and Santochi *et al.* [19].

This paper focuses on disassembly scheduling which is one of the important mid-term or short-term planning problems in disassembly systems. In general, disassembly scheduling can be defined as the problem of determining the quantity and timing of disassembling used or end-of-life products to satisfy the demand of their parts and/or components over a planning horizon. In other words, from its solution, one can determine which products, how many, and when to disassemble. That is, the problem corresponds to the production planning problem in assembly systems. However, due to the difference in the number of demand sources, disassembly scheduling is more complicated than the ordinary production planning problem. That is, in the assembly environment, parts/components converge to a single demand source of the final product, while in the disassemble environment, products diverge to its multiple demand sources of parts/components. See Brennan *et al.* [2] and Gupta and Taleb [3] for more details.

Most previous research on disassembly scheduling is uncapacitated ones, i.e., resource capacity restrictions are not considered. Gupta and Taleb [3] consider the basic case, i.e., single product type without parts commonality, and suggest a simple algorithm without explicit objective function, and Lee [12] shows the significance of cost consideration in solving the disassembly scheduling problems. Lee and Xirouchakis [15] suggest a heuristic algorithm for the objective of minimizing the costs related with disassembly process, and Kim *et al.* [5] suggest a branch and bound algorithm based on the Lagrangean relaxation that can give optimal solutions for small-sized problems. For the extended models with parts commonality, see Kim *et al.* [7, 8], Langella [11], Neuendorf *et al.* [17], Taleb and Gupta [20], and Taleb *et al.* [21]. Also, Lee *et al.* [14] present integer programming models for all uncapacitated cases together with their performances using a commercial software package.

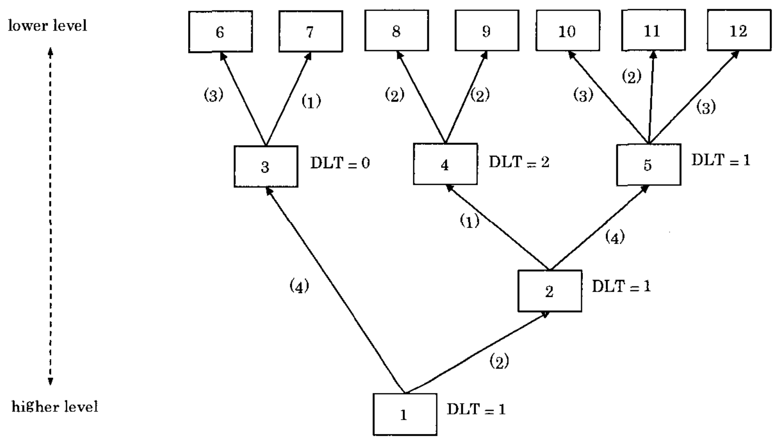
As in the production planning problems in assembly systems, the resource capacity restrictions should also be considered in disassembly scheduling so that the resulting schedule is more applicable. Several research articles consider capacitated disassembly scheduling for the case with single product types without parts commonality. Lee *et al.* [16] suggest an integer programming model that considers various cost factors occurred in disassembly processes. Although the optimal solutions can be obtained from the model, its application is limited only to the small-sized problems. In fact, the computational results show that it is not adequate for practical sized problems due to its excessive computation time. Recently, Kim *et al.* [9] suggested an optimal algorithm that minimizes the number of products disassembled, and Kim *et al.* [6] suggested a Lagrangean heuristic algorithm for the objective of minimizing the sum of setup, disassembly operation, and inventory holding costs.

This paper considers capacitated disassembly scheduling with the basic case of single product type without parts commonality for the objective of minimizing the sum of disassembly operation and inventory holding costs. In the aspect of problem considered, therefore, this paper extends the model of Kim *et al.* [9] that minimizes the number of product disassembled. Note that the cost-based objective is more general than the objective considered in Kim *et al.* [9]. Also, the problem considered here is a special case of the model of Kim *et al.* [6] that consider the objective of minimizing the sum of setup, disassembly operation, and inventory holding costs. In other words, Kim *et al.* [6] consider the additional setup cost in the objective function. Unlike this, we suggest a simple and efficient heuristic for the special case that setup is not a significant factor (e.g., automated disassembly systems). The problem is formulated as an integer programming model, and a two-stage heuristic, which consists of construction and improvement algorithms, is suggested in this paper. Computational experiments are done on a number of randomly generated problems, and the test results are reported.

This paper is organized as follows. In the next section, the problem is described in more detail with an integer programming model. Section 3 presents the two-stage heuristic and computational results on randomly generated test problems are reported in Section 4. Finally, Section 5 gives concluding remarks and discussion of future research.

2. PROBLEM DESCRIPTION

Before describing the problem, this section explains the disassembly product structure. In the disassembly structure, the root item represents the product itself to be disassembled and each leaf item is the part or component not to be disassembled further. A child item represents any item that has a parent item which has at least two child items. Note that a child item has only one parent item in the problem considered in this paper, i.e., the case without parts commonality. Figure 1 shows an example of disassembly product structure, obtained from Gupta and Taleb [3]. Item 1 is the root item, and items 6 to 12 are leaf items. The number in parenthesis represents the yield of the item when its parent is disassembled, e.g., disassembly of one unit of item 5 derives three units of item 10, two units of item 11, and three units of item 12. Here, item 5 is called parent item, while items 10, 11 and 12 are called its child items. Also, disassembly lead time (DLT) is the time required to disassemble a certain parent item. In general, the disassembly structure can be obtained from the disassembly process plan that describes disassembly level and sequence. See Lambert [10], Lee *et al.* [13], O'Shea *et al.* [18], and Santochi *et al.* [19] for more details on disassembly process planning.



DLT: disassembly lead time
(•): yield from the corresponding parent item

Figure 1. Disassembly product structure: an example

The capacitated disassembly scheduling problem considered in this paper can be described as follows: *for a given disassembly product structure, the problem is to determine the quantity and timing of disassembling each parent item (including the root item) to meet the demands of leaf items over a planning horizon while satisfying the resource capacity constraint in each period of the planning horizon.*

Here, the capacity constraint implies the available time in each period, and each disassembly operation consumes a portion of the available time. The objective is to minimize the sum of disassembly operation and inventory holding costs. The disassembly operation cost, which is proportional to the required labor or machine processing time, is the direct cost incurred to perform the disassembly operation. Also, the inventory holding cost occurs when items are stored to satisfy future demand, and they are computed based on the end-of-period inventory. It is assumed that disassembly operation and inventory holding costs are time invariant, i.e., cost values are the same over the planning horizon.

The disassembly product structure is assumed to be given by the corresponding disassembly process plan that specifies all disassembly operations with precedence relations and their processing times. Additional assumptions made in this problem are summarized as follows: (a) there is no shortage of the root items, i.e., products can be delivered whenever they are needed; (b) demands occur only for leaf items (not intermediate parent items) and they are given in advance and deterministic; (c) backlogging is not permitted and hence demands are satisfied on time; (d) parts and/or components obtained from disassembly are perfect in quality, i.e., no defectives; (e) each disassembly operation is done in only one period and cannot be done over two or more periods; and (f) disassembly lead times are given in advance and deterministic.

The problem can be formulated as an integer programming model. In the formulation, without loss of generality, all items are numbered with integer 1, 2, ... i_l, ... I. Here, i_l denotes the index for the first leaf item, and therefore the indices that are larger than or equal to i_l represent leaf items. The notations used in this paper are summarized below.

Parameters

p_i	disassembly operation cost of item i
h_i	inventory holding cost of item i
g_i	disassembly processing time of parent item i
C_t	available capacity (processing time) in period t
D_{it}	demand requirement of leaf item i in period t
a_{ij}	number of units (yield) of item j obtained by disassembling one unit of item i $(i < j)$
s_{it}	external scheduled receipt of item i in period t
$\varphi(i)$	parent of item i
l_i	disassembly lead time (DLT) of item i
I_{i0}	initial inventory of item i

Decision variables

X_{it} amount of item i disassembled in period t

I_{it} inventory level of item i at the end of period t

Now, the integer programming model is given below.

[P] Minimize $\sum_{i=1}^{i_l-1}\sum_{t=1}^{T} p_i \cdot X_{it} + \sum_{i=2}^{I}\sum_{t=1}^{T} h_i \cdot I_{it}$

subject to

$$I_{it} = I_{i,t-1} + s_{it} + a_{\varphi(i),i} \cdot X_{\varphi(i),t-l_{\varphi(i)}} - X_{it} \quad \text{for } i = 2, 3, \ldots i_l - 1 \text{ and } t = 1, 2, \ldots T \quad (1)$$

$$I_{it} = I_{i,t-1} + s_{it} + a_{\varphi(i),i} \cdot X_{\varphi(i),t-l_{\varphi(i)}} - D_{it} \quad \text{for } i = i_l, i_l + 1 \ldots I \text{ and } t = 1, 2, \ldots T \quad (2)$$

$$\sum_{i=1}^{i_l-1} g_i \cdot X_{it} \le C_t \quad \text{for } t = 1, 2, \ldots T \quad (3)$$

$$X_{it} \ge 0 \text{ and integers} \quad \text{for } i = 1, 2, \ldots i_l - 1 \text{ and } t = 1, 2, \ldots T \quad (4)$$

$$I_{it} \ge 0 \text{ and integers} \quad \text{for } i = 2, 3, \ldots I \text{ and } t = 1, 2, \ldots T \quad (5)$$

The objective function denotes the sum of disassembly operation and inventory holding costs. Constraint (1) represents the inventory balance of each parent item. That is, at the end of each period, the inventory level of the parent item is what we had before the period, increased by the external scheduled receipt and the quantity obtained by disassembling its corresponding parent item, and decreased by the quantity of the item disassembled in that period. Here, the inventory balance constraint of the root item is not included because it is not necessary to have surplus inventory of the root item. Also, the inventory balance of each leaf item is represented by constraint (2), which is different from (1) in that the demand requirement is used instead of the amount of items disassembled. Also, constraint (3) represents the resource capacity restriction in each period. That is, the sum of processing times of disassembly operations assigned to each period should be less than or equal to the given capacity of that period. Finally, the constraints (4) and (5) represent the conditions of the decision variables. Particularly, constraint (5) guarantees that backlogging is not permitted.

3. TWO-STAGE HEURISTIC

Although the optimal solutions can be obtained by solving problem [P] using a commercial software package, it requires an excessive and inconsistent amount of computation time. (The computational results will be reported in Section 4.) Therefore, we suggest a heuristic consisting of two stages: a construction algorithm and an improvement algorithm. The construction algorithm gives an initial feasible solution (especially, with respect to the capacity constraint), and the improvement algorithm modifies the initial solution by iteratively changing and evaluating the current disassembly schedules while considering cost changes.

3.1 Stage 1: Constructing an initial solution

The initial feasible solution is obtained as follows. First, an infeasible solution (with respect to the capacity constraints) is obtained using the algorithm of Gupta and Taleb [3], to be called the GT algorithm hereafter. Then, it is modified iteratively into a feasible solution that satisfies the capacity constraints. Note that the GT algorithm gives the minimal latest disassembly schedule in that it satisfies the demands of leaf items as latest as possible with the minimum amount of disassembly operations. (See Lee and Xirouchakis [15] for more details on the GT algorithm and the minimal latest disassembly schedule.) Therefore, if the solution obtained from the GT algorithm satisfies the capacity constraint, it minimizes the sum of disassembly operation and inventory holding costs, and hence it is optimal for the problem [P]. Otherwise, the infeasible solution should be modified into a feasible one with respect to the capacity constraints.

The modification method suggested in this paper is based on the backward moves of disassembly operations. Here, the backward move implies that the disassembly operations assigned to a later period is moved to an earlier period while considering the capacity constraint of the earlier period. Note that a forward move from an earlier period to a later period results in an infeasible solution because the disassembly schedule obtained from the GT algorithm is the latest one. In this paper, the backward move is done from the root item (to the parent items) until the capacity overload is eliminated. (The method to determine the amounts of moves will be explained later.) If there are no movable items while the capacity is still overloaded, the disassembly operations of the root item assigned to one period earlier is moved backwardly so that the inventories of non-root parent items can be provided. Note that each of the non-root parent items can be moved only if its inventory level is sufficient and the move of the root item provides the inventories of non-root parent items.

Before explaining the detailed method to modify the infeasible solution obtained from the GT algorithm, we first define the overloaded capacity (O_t) with a threshold value β as follows:

$$O_t = \begin{cases} L_t - \beta & \text{if } L_t \geq \beta \\ L_t & \text{otherwise} \end{cases}$$

where $L_t = \sum_{i=1}^{i_l-1} g_i \cdot X_{it} - C_t$, i.e., the overloaded capacity at period t. (Note that X_{it}'s for $i = 1, 2, \ldots i_l - 1$ are given from the solution of the GT algorithm.) Here, the overloaded capacity is modified with the threshold value for the objective of distinguishing the amounts of moves. That is, if the overloaded capacity (L_t) is more than or equal to the threshold value, the amount of move is set to the overloaded capacity subtracted by the threshold value. Otherwise, the moves are done one by one so that the remaining capacity after the move is maintained as least as possible.

As explained earlier, the modification is done from the backward move of the root item. Let t denote the current period for which the backward move is considered. Then, the amount of its move is determined as

$$q_1 = \min\{\left\lceil \frac{O_t}{g_1} \right\rceil, \left\lfloor \alpha \cdot X_{1t} \right\rfloor\},$$

where $\lceil \bullet \rceil$ is the smallest integer that is greater than or equal to $\bullet$, $\lfloor \bullet \rfloor$ is the largest integer that is less than or equal to $\bullet$, and α is a parameter, used to avoid excessive inventories, with the range [0, 1]. If the capacity is still overloaded after the move, the backward moves for parent items are done into one period earlier. To do this, we first select the item with

$$i^* = \arg\min_{i \in \Phi} \{\left\lceil \frac{O_t}{g_i} \right\rceil - \frac{O_t}{g_i}\},$$

where $\Phi = \{i \mid \min(X_{it}, I_{i,t-1}) > 0,\ i = 1,\ 2,\ \ldots\ i_l - 1\}$. Also, the amount of its move is determined as

$$q_{i^*} = \begin{cases} \min\{\left\lceil \frac{O_t}{g_{i^*}} \right\rceil, X_{i^*t}, I_{i^*,t-1}\} & \text{if } L_t \geq \beta \\ 1 & \text{otherwise} \end{cases}$$

In the case that there are no movable items while the capacity is still overloaded, the backward move of the root item is done from period $t-1$ to period $t-1-l_1$ so that the parent items in the current period t can be moved. Note that the move of the root item can provide the indispensable inventories of non-root parent items in the period $t-1$. Here, the amount of its move is determined as

$$A = \min[X_{1,t-1}, \max_{i \in H(1)}\{\min(\left\lceil \frac{L_t}{a_{1i} \cdot g_i} \right\rceil, \frac{X_{it}}{a_{1i}})\}],$$

where $H(i)$ denotes the set of child items of parent i. Here, the max term gives the amount of move required for eliminating the overloaded capacity.

Now, the overall procedure of the first stage is summarized below.

Procedure 1. (*Construction of an initial solution*)

Step 1. Obtain an uncapacitated solution using the GT algorithm and calculate the amount of overload for each period, i.e., L_t for all t. If the solution satisfies the capacity constraints for all periods, then stop (The solution is optimal.).

Step 2. Set $a = 0, \beta = 0$, $\gamma = \sum_{i=1}^{i_l-1} g_i / (i_l - 1)$ and $L_{\max} = \max_{t=1,2,\ldots T}\{L_t\}$.

Step 3. Do the following steps:

(a) Set $t' = 1$.

(b) Set $t = t'$.

(c) If $L_t \le 0$, i.e., not overloaded, go to (g). Here, if $L_1 > 0$, i.e., overloaded in period 1, go to Step 4.

(d) Perform the move for the root item using the method explained earlier. If the overload in period t is eliminated, go to (g). Otherwise, go to (e).

(e) If there are movable parent items in period t, perform their moves repeatedly using the method explained earlier until eliminating the overload, and go to (g). Otherwise, go to (f).

(f) If there are movable root item in period $t-1$, perform its move from period $t-1$ to $t-1-l_1$ using the method explained earlier, and go to (e). Otherwise, go to Step 4.

(g) Set $t = t - 1$. If $t \ge 1$, go to (c).

(h) Set $t' = t' + 1$. If $t' \le T$, go to (b). Otherwise, stop and save the solution if the amounts of overloads for all periods are eliminated.

Step 4. Set $a = a + 0.1$. If $a \le 1$, go to Step 3.

Step 5. Set $a = 0$ and $\beta = \beta + \gamma$. go to Step 3. If $\beta > L_{\max}$, stop. (The algorithm may not be able to find a feasible solution or the problem is infeasible.)

3.2 Stage 2: Improvement

This second stage improves the initial solution while considering the changes in total cost. The improvement method suggested in this paper is based on two types of moves, *forward* and *backward moves,* at the same time. The forward move implies that a portion of the disassembly quantity assigned to a period is moved to one period later, while the backward move is vice versa. Before explaining the moves, we make the following assumption

$$h_i \le \sum_{k \in H(i)} h_k \cdot a_{ik} \quad \text{for} \quad i = 2, 3, \dots i_l{-}1,$$

which implies that the item's value may increase as the disassembly operations progress. Note that this assumption is not very restrictive since the inventory holding cost is directly related to the item's value.

3.2.1 Forward move

Let X_{it}, for $i = 1, 2, \dots i_l - 1$ and $t = 1, 2, \dots T$, be the current disassembly schedule. Suppose that n disassembly operations of parent item i in period t are moved to period $t + 1$. Then, new disassembly quantities X'_{it} and $X'_{i,t+1}$ of item i in periods t and $t + 1$ become

$$X'_{it} = X_{it} - n \text{ and } X'_{i,t+1} = X_{i,t+1} + n ,$$

and the inventory levels of the non-root parent item i and its child items are changed into

$$I'_{it} = I_{it} + n \text{ and } I'_{k,t+l_i} = I_{k,t+l_i} - n \cdot a_{ik} \text{ for } k \in H(i).$$

Note that the inventory level of the root item is not considered in this paper since the root items can be delivered whenever they are required. (See assumption (a).) Then, the possible range of n becomes

$$0 \le n \le \min\{ \min_{k \in H(i)} \left\lfloor \frac{I_{k,t+l_i}}{a_{ik}} \right\rfloor, X_{it} \}, \tag{6}$$

which can be obtained from the constraint that the disassembly quantities and inventory levels should be nonnegative. The forward move decreases the inventory holding cost, which can be represented as

$$n \cdot (\sum_{k \in H(i)} a_{ik} \cdot h_k - h_i).$$

Note that the forward move results in no change in disassembly operation cost since the total amount of disassembly operations remains the same after the move and the disassembly operation costs are time-invariant.

3.2.2 Backward move

Suppose that m disassembly operations of item j are moved from period $t + 1$ to t. Then, new disassembly quantities X'_{jt} and $X'_{j,t+1}$ of item j in periods t and $t + 1$ become

$$X'_{jt} = X_{jt} + m \text{ and } X'_{j,t+1} = X_{j,t+1} - m,$$

and the inventory levels of non-root parent item j and its child items are changed into

$$I'_{jt} = I_{jt} - m \text{ and } I'_{k,t+l_j} = I_{k,t+l_j} + m \cdot a_{jk} \qquad \text{for } k \in H(j).$$

As in the forward move, the possible range of m becomes

$$0 \le m \le \min\{X_{j,t+1}, I_{jt}\}, \tag{7}$$

which can be obtained from the nonnegativity constraints explained in the forward move. However, unlike the forward move, the backward move increases the inventory holding cost, which can be represented as

$$m \cdot (h_j - \sum_{k \in H(j)} a_{jk} \cdot h_k).$$

Also, as in the forward move, the backward move results in no change in disassembly operation cost.

As stated earlier, the improvement is done by the simultaneous forward and backward moves. In other words, an improved solution can be obtained if the cost decrease obtained from the forward move is greater than the cost increase from the backward move. Here, required is the method to determine the amounts of moves while keeping the feasibility for the capacity constraints.

Suppose that n disassembly operations of parent item i are moved from period t to $t + 1$, i.e., forward move, and m disassembly operations of item j are moved from period $t + 1$ to t, i.e., backward move, at the same time. Then, after the simultaneous forward and backward move, the remaining capacities in periods t and $t + 1$ can be obtained as

$$R'_t = R_t + n \cdot g_i - m \cdot g_j \text{ and } R'_{t+1} = R_{t+1} - n \cdot g_i + m \cdot g_j,$$

where $R_t = C_t - \sum_{i=1}^{i_t-1} g_i \cdot X_{it}$. Therefore, we can see that the simultaneous move is feasible if $R_t' \geq 0$ and $R_{t+1}' \geq 0$. This gives the range for the amount m of backward move for a given amount n of forward move.

$$\frac{n \cdot g_i - R_{t+1}}{g_j} \leq m \leq \frac{n \cdot g_i + R_t}{g_j} \tag{8}$$

We explain the method to determine the amounts m and n of the simultaneous forward and backward move. Here, it is better to set the value of m to the lower limit in (8) since the backward move increases the total cost, while the value of n to the upper limit for its possible range since the forward move decreases the total cost. Then, based on (7) and (8), we can consider four cases given below.

i) If $(n \cdot g_i + R_t)/g_j < 0$, then there exists no possible amount n.

ii) If $(n \cdot g_i - R_{t+1})/g_j \leq 0 \leq (n \cdot g_i + R_t)/g_j$, then it is better to set m = 0. Also, the range of n can be determined as $n \leq R_{t+1}/g_i$ from $(n \cdot g_i - R_{t+1})/g_j \leq 0$. Therefore, we can obtain

$$m^* = 0,\ n^* = \min\{\min_{k \in H(i)} \left\lfloor \frac{I_{k,t+l_i}}{a_{ik}} \right\rfloor, \left\lfloor \frac{R_{t+1}}{g_i} \right\rfloor, X_{it}\} \tag{9}$$

Here, n^* is calculated with the range (6).

iii) If $0 < (n \cdot g_i - R_{t+1})/g_j \leq \min\{X_{j,t+1}, I_{jt}\}$, then we can obtain the range of n as

$$\frac{R_{t+1}}{g_i} \leq n \leq \frac{g_j \cdot \min\{X_{j,t+1}, I_{jt}\} + R_{t+1}}{g_i}.$$

Therefore, the amount of forward move becomes

$$n^* = \min\{\min_{k \in H(i)} \left\lfloor \frac{I_{k,t+l_i}}{a_{ik}} \right\rfloor, \left\lfloor \frac{g_j \cdot \min\{X_{j,t+1}, I_{jt}\} + R_{t+1}}{g_i} \right\rfloor, X_{it}\}, \tag{10-1}$$

since it is better to set the value of n to the upper limit while considering the range (6). Also, for a given n^*, the amount of backward move becomes

$$m^* = \left\lceil \frac{(n^* \cdot g_i - R_{t+1})}{g_j} \right\rceil. \tag{10-2}$$

iv) If $\min\{X_{j,t+1}, I_{jt}\} < (n \cdot g_i - R_{t+1})/g_j$, then there exists no possible amount m due to the capacity limitation.

Now, the overall procedure of the second stage is summarized below.

Procedure 2. (*Improvement*)

Step 1. Set $i = 1$

Step 2. For item i, do the following steps:
- (a) Set $t = 1$
- (b) Set $j = i + 1$
- (c) Perform the simultaneous backward and forward move between items i and j by the amount n^* and m^* given in (9), (10-1) and (10-2). If this reduces the total cost, update the solution and go to Step 1.
- (d) Set $j = j + 1$. If $j > i_l - 1$, set $t = t + 1$ and go to (b). Otherwise, go to (c) Here, if $t \geq T$, go to Step 3.

Step 3. Set $i = i + 1$, if $i > i_l - 1$, stop. Otherwise, go to Step 2.

4. COMPUTATIONAL EXPERIMENTS

To test the performance of the two-stage heuristic, computational experiments were performed on a number of randomly generated test problems. Two performance measures were used in this test: percentage deviations from the optimal solution values and CPU seconds. Here, optimal solution values were obtained from solving the integer program [P] directly using CPLEX 9.0. Also, the two-stage heuristic is compared with the algorithm of Kim *et al.* [9] (without and with the improvement method). Although the existing algorithm is designed for the objective of minimizing the number of products disassembled, the comparison is done to show the importance of cost-based objective. The algorithm and the program to generate integer programs were coded in C and the tests were done on a personal computer with a Pentium processor operating at 2.0 GHz clock speed.

For the test, 750 problems were generated in total, i.e., 25 problems for each combination of two levels of capacity tightness (loose and tight), five levels of the number of items (10, 20, 30, 40, and 50), and three levels of the number of periods (10, 20, and 30). For each level of the number of items, 5 disassembly products structures (and hence totally 25) were randomly generated. In the disassembly structure, the number of child items for each parent and its yield were generated from *DU*(2, 5) and *DU*(1, 3), respectively. Here, *DU*(*a*, *b*) is the discrete uniform distribution with range [*a*, *b*]. Also, disassembly lead times were set to 0, 1, and 2 with probabilities, 0.2, 0.7, and 0.1, respectively. For each disassembly structure, 5 problems with different data were generated for each level of the number of periods. Disassembly operation costs were generated from *DU*(50, 100) and inventory holding costs were generated from *DU*(5, 10). The capacity of each period was set to 400, 480, and 540 with probabilities 0.2, 0.5, and 0.3, respectively, and disassembly times were generated from *DU*(1, 4). To consider the capacity tightness, the demands were generated using the following procedure:

(1) The initial demands were generated from 0 or *DU*(50, 200) with probabilities 0.1 and 0.9, respectively;
(2) The problem with the initial demands was solved using the GT algorithm;
(3) The overall capacity usage (*CU*) was calculated using $CU = \sum_{i=1}^{i_l-1}\sum_{t=1}^{t=T} g_i \cdot X_{it}$, where X_{it} is the solution of the GT algorithm.
(4) The demands were regenerated using $d_{it} = \lfloor \tau \cdot TC/CU \cdot d'_{it} \rfloor$ where τ is a parameter that represents capacity tightness (τ is set to 0.7 and 0.9 for the cases of loose and tight capacity tightness, respectively), *TC* is the sum of capacities over the planning horizon, and d'_{it} is the demand generated initially in (1).

Finally, in the generated problems, external scheduled receipts and initial inventory levels were set to 0 without loss of generality.

Test results are summarized in Table 1 that shows average percentage deviations from optimal solution values and average CPU seconds. It can be seen from the table that the two-stage heuristic can give very near optimal solutions. In fact the overall averages were within 0.053% and 0.65% for the cases of loose and tight capacities, respectively. Also, the two-stage heuristic suggested in this paper significantly outperformed the algorithm of Kim *et al.* [9] that minimizes the number of products disassembled. This implies that the existing algorithm does not work well for the cost-based objective since it assigns the disassembly operations as many as possible from the first to the last period so that the inventory holding cost increases excessively. Also, it can be seen from the table that significant improvements are obtained from the second stage. This shows the effectiveness of the improvement procedure. Finally, the CPU seconds of the two-stage heuristic were much smaller than those of the CPLEX. Although we can obtain the optimal solutions from the CPLEX, its application is limited only to the small-

Table 1. Test results for the two-stage heuristic

(a) Case of loose capacity

Number of	Number of	Percentage deviations		CPU seconds	
items	periods	Two-stage	Kim *et al*[†]	Two-stage	CPLEX
10	10	0.73 (0.11)[‡]	26.0 (24.3)	0.00[*]	0.02
10	20	0.52 (0.01)	43.5 (42.6)	0.00	0.03
10	30	0.14 (0.07)	40.9 (40.5)	0.00	0.04
20	10	0.59 (0.08)	39.9 (35.3)	0.00	0.04
20	20	0.14 (0.02)	55.1 (51.8)	0.00	0.04
20	30	0.17 (0.01)	59.7 (57.1)	0.00	0.10
30	10	0.40 (0.02)	33.2 (29.5)	0.00	0.03
30	20	0.21 (0.02)	49.0 (44.7)	0.00	0.06
30	30	0.07 (0.003)	54.5 (51.0)	0.00	0.11
40	10	0.65 (0.17)	30.8 (23.2)	0.00	6.96
40	20	0.73 (0.02)	46.1 (40.1)	0.00	0.11
40	30	0.04 (0.001)	54.2 (49.0)	0.00	37.73
50	10	1.43 (0.17)	28.4 (22.3)	0.00	0.11
50	20	0.16 (0.01)	47.7 (40.8)	0.00	0.12
50	30	0.03 (0.003)	58.0 (52.5)	0.00	0.21
average		0.36 (0.053)	44.5 (40.3)		

† The algorithm of Kim *et al.* (2005)

‡ average percentage deviation out of 25 problems without and with (in parenthesis) the improvement stage procedure

* Average CPU seconds is less than 0.005s

(b) Case of tight capacity

Number of	Number of	Percentage deviations		CPU seconds	
items	periods	Two-stage	Kim *et al.*	Two-stage	CPLEX
10	10	4.39 (1.04)	7.7 (6.8)	0.00	0.8
10	20	2.60 (0.67)	13.9 (13.2)	0.00	164.0
10	30	2.92 (0.43)	14.6 (14.0)	0.00	4.3
20	10	3.55 (1.68)	12.5 (09.7)	0.00	6.7
20	20	3.47 (0.73)	17.8 (15.0)	0.01	226.9
20	30	3.25 (0.75)	22.4 (19.5)	0.01	532.8
30	10	4.12 (0.50)	10.2 (6.6)	0.00	0.4
30	20	3.26 (0.41)	14.9 (11.7)	0.00	21.6
30	30	1.74 (0.23)	17.2 (15.0)	0.01	0.7
40	10	3.63 (0.67)	8.4 (5.3)	0.00	3.5
40	20	3.41 (0.28)	16.9 (13.1)	0.01	436.1
40	30	2.01 (0.32)	20.6 (16.9)	0.01	443.1
50	10	4.38 (1.12)	8.5 (5.0)	0.01	0.2
50	20	4.04 (0.69)	18.0 (13.2)	0.02	20.5
50	30	1.98 (0.17)	24.2 (19.9)	0.02	2.9
average		3.25 (0.649)	15.19 (12.33)		

sized problems due to its long and inconsistent computation times. Therefore, we can argue that the two-stage heuristic suggested in this paper can be more effective in the practical sense.

5. CONCLUDING REMARKS

This paper considered the disassembly scheduling problem that determines the quantity and timing of disassembling used or end-of-life products while satisfying the demand of their parts and/or components over a given planning horizon. The resource capacity constraint was considered explicitly for the objective of minimizing the sum of disassembly operation and inventory holding costs. The problem was formulated as an integer programming model, and a two-stage heuristic was suggested that consists of construction and improvement algorithms. The construction algorithm gives an initial feasible solution, and the improvement algorithm modifies the initial solution by iteratively changing the disassembly schedules while considering cost changes. Computational experiments on a number of randomly generated test problems showed that the two-stage heuristic could give very near optimal solutions within a very short amount of computation time.

This research can be extended in several ways. First, it is needed to consider more general cases such as multiple product types and part commonality. Here, the part commonality introduces one or more procurement sources for each common part and hence makes the problem difficult to solve. Second, like other disassembly problems, uncertainties such as stochastic demand, defective parts or components, and stochastic disassembly operation times are important further considerations.

REFERENCES

[1] Boothroyd, G. and L. Alting, "Design for assembly and disassembly," *Annals of the CIRP* 41 (1992), 625-636.

[2] Brennan, L., S. M. Gupta, and K. N. Taleb, "Operations planning issues in an assembly/disassembly environment," *International Journal of Operations and Production Management* 14 (1994), 57-67.

[3] Gupta, S. M. and K. N. Taleb, "Scheduling disassembly," *International*

Journal of Production Research 32 (1994), 1857-1886.

[4] Jovane, F., L. Alting, A. Armoillotta, W. Eversheirm, K. Feldmann, G. Seliger, and N. Roth, "A key issue in product life cycle: disassembly," *Annals of the CIRP* 42 (1993), 651-658.

[5] Kim, H.-J., D.-H. Lee, and P. Xirouchakis, "An optimal algorithm for disassembly scheduling with assembly product structure," *Lecture Notes in Artificial Intelligence* 3698 (2005), 235-248.

[6] Kim, H.-J., D.-H. Lee, and P. Xirouchakis, "A Lagrangean heuristic algorithm for disassembly scheduling with capacity constraints," to appear in *Journal of the Operational Research Society* (2006).

[7] Kim, H.-J., D.-H. Lee, and P. Xirouchakis, "Two-phase heuristic for disassembly scheduling with multiple product types and parts commonality," *International Journal of Production Research* 44 (2006), 195-212.

[8] Kim, H.-J., D.-H. Lee, P. Xirouchakis, and R. Züst, "Disassembly scheduling with multiple product types," *Annals of the CIRP* 52 (2003), 403-406.

[9] Kim, J.-G., H.-B. Jeon, H.-J. Kim, D.-H. Lee, and P. Xirouchakis, "Capacitated disassembly scheduling: minimizing the number of products disassembled," *Lecture Notes in Computer Science* 3483 (2005), 538-547.

[10] Lambert, A. J. D., "Disassembly sequencing: a survey," *International Journal of Production Research* 41 (2003), 3721-3759.

[11] Langella, I. M., "Heuristics for demand-driven disassembly planning," to appear in *Computers and Operations Research* (2005).

[12] Lee, D.-H., "Disassembly scheduling for products with assembly structure," *International Journal of Management Science* 11 (2005), 63-78.

[13] Lee, D.-H., J.-G. Kang, and P. Xirouchakis, "Disassembly planning and scheduling: review and further research," *Proceedings of the Institution of Mechanical Engineers: Journal of Engineering Manufacture-Part B* 215 (2001), 695-710.

[14] Lee, D.-H., H.-J. Kim, G. Choi, and P. Xirouchakis, "Disassembly scheduling: integer programming models," *Proceedings of the Institution of Mechanical Engineers: Journal of Engineering Manufacture-Part B* 218 (2004), 1357-1372.

[15] Lee, D.-H. and P. Xirouchakis, "A two-stage heuristic for disassembly scheduling with assembly product structure," *Journal of the Operational Research Society* 55 (2004), 287-297.

[16] Lee, D.-H., P. Xirouchakis, and R. Züst, "Disassembly scheduling with capacity constraints," *Annals of the CIRP* 51 (2002), 387-390.

[17] Neuendorf, K.-P., D.-H. Lee, D. Kiritsis, and P. Xirouchakis, "Disassembly

scheduling with parts commonality using Petri-nets with timestamps," *Fundamenta Informaticae* 47 (2001), 295-306.

[18] O'Shea, B., S. S. Grewal, and H. Kaebernick, "State of the art literature survey on disassembly planning," *Concurrent Engineering: Research and Application* 6 (1998), 345-357.

[19] Santochi, M., G. Dini, and F. Failli, "Computer aided disassembly planning: state of the art and perspectives," *Annals of the CIRP* 51 (2002), 1-23.

[20] Taleb, K. N. and S. M. Gupta, "Disassembly of multiple product structures," *Computers and Industrial Engineering* 32 (1997), 949-961.

[21] Taleb, K. N., S. M. Gupta, and L. Brennan, "Disassembly of complex product structures with parts and materials commonality," *Production Planning and Control* 8 (1997), 255-269.