International Journal of Management Science
Vol 11, No 1, May 2005

Disassembly Scheduling for Products with Assembly Structure*

Dong-Ho Lee**

Department of Industrial Engineering
Hanyang University, Seongdong-gu, Seoul 133-791, Korea

(Received Dec. 2004 ; Revised Mar. 2005 ; Accepted Mar. 2005)

ABSTRACT

Disassembly scheduling is the problem of determining the ordering and disassembly schedules of used or end-of-life products while satisfying the demand of their parts or components over a certain planning horizon. This paper considers the case of the assembly product structure for the cost-based objective of minimizing the sum of purchase, setup, inventory holding, and disassembly operation costs. To represent and solve the problem optimally, this paper presents an integer programming model, which is a reversed form of the multi-level lot sizing formulation. Computational experiments on an example derived from the literature and a number of randomly generated test problems are done and the results are reported.

Keywords: Disassembly Scheduling, Cost-based Objective, Integer Program, Environmental Issues

1. INTRODUCTION

Environmental issues have been important for manufacturing firms due to increasing legislation pressures to collect and upgrade their products in an environmentally conscious way. Under such circumstances, a number of manufacturing firms are paying considerable attention to remanufacturing, which can be defined as an industrial process in which worn-out products are restored to like-new condition [13]. Through a series of industrial processes, discarded products are

* This work was supported by the research fund of Hanyang University (HY-2003). This support is greatly acknowledged.
** Email: leman@hanyang.ac.kr

partially or completely disassembled, and usable parts are cleaned, refurbished, and put into inventory. Then, the new product is reassembled from both old and new parts in order to produce a unit fully equivalent and sometimes superior in performance and expected lifetime to the original new product. See Guide [3] for more details of remanufacturing.

Among the remanufacturing processes, this paper focuses on disassembly, in which a used or end-of-life product is separated into its constituent parts, components or other groupings with necessary sorting operations. In general, disassembly can be applied to: (a) recover pure material fractions; (b) isolate hazardous substances; and (c) separate reusable parts and/or subassemblies. Disassembly is an important product recovery process since most products are disassembled before they are recycled, remanufactured, and even disposed.

There are a number of previous research articles on various disassembly problems [9, 15, 17]. For example, see Johnson and Wang [5], Kang *et al.* [6], Lambert [8], and Pnueli and Zussman [16] on disassembly planning. In general, disassembly planning is the problem of determining the disassembly level and the corresponding sequence of disassembly operations for a product. Here, the disassembly level is concerned with the decision to continue or stop more disassembly operations at each stage of disassembly process. Also, the end-of-life options, such as reuse, remanufacturing, recycling, disposal etc., are determined in disassembly planning.

Unlike a lot of research work on disassembly planning, not much work has been done on disassembly scheduling, i.e., the problem of determining the ordering and the disassembly schedules of used or end-of-life products while satisfying the demands of their parts or components. This problem is an important planning problem in disassembly systems. In other words, by solving the problem, we can determine which items (products or subassemblies), how many, and when to disassemble products in order to satisfy the demand of their parts or components. In general, disassembly scheduling can be regarded as a reverse material requirement planning (MRP) problem, since it is basically a reversed form of the ordinary MRP problem [4]. However, due to the difference in the number of demand sources, disassembly scheduling is more complicated than the ordinary MRP problem. That is, in the assembly environment, parts/components converge to a single demand source of the final product, while in the disassembly environment, products diverge to its multiple demand sources of parts/components. See Brennan *et al.* [2] for more details on the differences between assembly and disassembly systems.

Several research articles consider the disassembly scheduling problem.

Gupta and Taleb [4] define and characterize the basic disassembly scheduling problem for a single product type with assembly structure. They show that the problem can be regarded as a reversed form of material requirement planning (MRP) and then suggest an MRP-like algorithm. In the algorithm, the demand of items at one level of the product structure is translated into an equivalent demand of the items at the next level, and this is repeated from the part level to the product level. Later, Taleb *et al.* [19] extend the basic model with the consideration of parts commonality, i.e., general product structure, and suggest another MRP-like algorithm for the objective of minimizing the number of products to be disassembled. The extended problem is more complex than the basic one since the parts commonality results in one or more alternative procurement sources for each common part and hence creates dependencies between the components. For this extended problem, Neuendorf *et al.* [14] suggest an algorithm based on the Petri-net, and show using the example of Taleb *et al.* [19] that their algorithm gives a better solution than the MRP-like algorithm. Also, Taleb and Gupta [18] consider the case of multiple product types with parts commonality, and suggest a two-phase heuristic. Recently, Lee *et al.* [12] suggest an integer programming model for the basic case with resource capacity constraint, and Kim *et al.* [7] suggest a linear programming relaxation heuristic for the case with multiple product types and part commonality.

This paper focuses on the basic disassembly scheduling problem, i.e., a single product type of assembly structure. In fact, the problem considered here is the same as that of Gupta and Taleb [4] except for the objective function. While no explicit objective is considered in Gupta and Taleb [4], this paper considers various costs, such as purchase, disassembly operation, inventory holding, and setup costs. The extended version of the problem is more complex than the original one because of its similarities to the traditional multi-level lot sizing problems. To represent and solve the extended version of the problem, this paper presents an integer programming model using the fact that the problem considered in this paper can be regarded as a reverse form of the uncapacitated multi-level lot sizing problem. See Bahl *et al.* [1] for a literature review on the multi-level lot sizing problem. Computational experiments were done on the example (with additional cost factors) of Gupta and Taleb [4] and a number of randomly generated problems, and the results are reported, especially on the cost considerations in disassembly scheduling.

This paper is organized as follows. The next section describes the problem considered here in more detail, and the integer programming model is presented in Section 3. In Section 4, test results on the examples derived from Gupta and

Taleb [4] and a number of randomly generated test problems are summarized. Finally, Section 5 concludes the paper with a summary and discussion of future research.

2. PROBLEM DESCRIPTION

To describe the disassembly scheduling problem considered in this paper, this section begins with the definition of the disassembly product structure. Figure 1 shows an example, called the GT example in this paper, obtained from Gupta and Taleb [4]. In the disassembly product structure, the *root item* represents the product itself to be disassembled and each *leaf item* represents any item that cannot be further disassembled. Also, a *child item* represents any item that has a parent at the next lower level and a *parent item* is any item that has at least two child items at the next higher level. Note that in the structure of assembly type, each item has at most one parent, i.e., parts commonality is not allowed. In Figure 1, item 1 represents the root item, and items 6, 7, 8, 9, 10, 11, and 12 represent the leaf items. The numbers in parentheses represent the yield of the item when its parent is disassembled, e.g., item 5 is disassembled into 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 child items. Also, the *disassembly lead-time* (DLT) is the time required to disassemble a certain parent item.

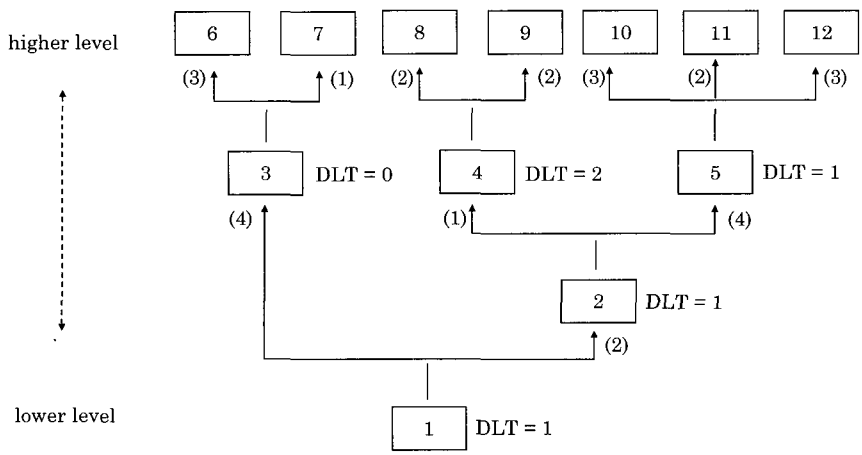

Figure 1. Disassembly product structure: an example

Now, the disassembly scheduling problem considered in this paper can be described as follows: *For a given disassembly product structure, the problem is to determine the ordering schedule of the root item and the disassembly schedule of all parent items while satisfying the demands of leaf items over a certain planning horizon.* The objective is to minimize the sum of purchase, setup, inventory holding, and disassembly operation costs. In particular, the time-varying purchase costs of used products are considered in this paper since the purchase cost of used or end-of-life products depend highly on the market situation. Here, the time-varying cost implies that cost values are different according to different periods in the planning horizon. When an item is disassembled, the equipment or tool required for that item has to be setup. This causes the item specific setup cost. It is assumed that the setup cost for an item in a period occurs if at least one disassembly operation for that item should be performed in that period. Also, the inventory holding cost occurs when items are held in order to satisfy future demand. Finally, the disassembly operation cost, which is proportional to direct labor time or machine processing time, is the cost incurred while performing the disassembly operation.

As stated earlier, the problem considered in this paper is the same as that of Gupta and Taleb [4] except for the objective function. While no explicit objective function is considered in Gupta and Taleb [4], this paper extends the problem by considering the cost-based objective function that is generally used in ordinary production planning problems. In fact, the disassembly scheduling problem is alternatively called the reverse MRP problem, since it is basically a reversed form of the regular MRP problem [4]. However, due to the difference in the number of demand sources, the disassembly scheduling problem is considerably more complicated than the MRP problem. That is, in the assembly environment, parts converge to a single demand source, i.e., the final product, while in the disassembly environment, parts diverge from the product to its multiple demand sources. See Gupta and Taleb [4] for more details of the comparison of the assembly and the disassembly environments.

It is assumed that the disassembly product structure is given from the corresponding disassembly plan that specifies all parts/subassemblies and their disassembly operations. Here, the disassembly plan is the solution of the disassembly planning problem described earlier. Also, it is assumed that there is no shortage of the root item (product). In other words, the required number of products can be supplied whenever they are ordered. The other assumptions made in this problem, which are similar to those of MRP, are as follows: (a) the planning horizon is divided into discrete planning periods; (b) demands of leaf items are given and de-

terministic; (c) backlogging is not allowed and hence demand should be satisfied on time; (d) the disassembly process is perfect, and hence any defective parts resulting from disassembly are not considered; (e) disassembly lead times with discrete time scale are given and deterministic; and (f) inventory holding costs are computed based on the end-of-period inventory.

Table 1. Data for the GT example

(a) Initial inventory

Item	1	2	3	4	5	6	7	8	9	10	11	12
Initial Inventory	1	6	2	45	12	10	30	55	20	120	90	80

(b) Scheduled receipts from external source

Item	Period									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	9	0	2	0
2	5	7	1	9	0	0	0	0	0	0
3	2	4	8	0	0	0	6	0	2	0
4	0	0	15	2	4	8	0	0	0	0
5	1	9	0	2	3	0	0	0	0	0
6	0	15	0	0	2	4	0	0	0	0
7	0	8	0	0	0	0	0	0	0	0
8	1	6	0	2	1	2	0	0	0	0
9	7	1	9	0	0	0	7	2	8	0
10	3	15	2	4	8	0	0	2	5	1
11	0	0	5	7	1	9	0	2	3	0
12	4	7	2	8	1	6	0	2	1	2

(c) Demand

leaf item	Period									
	1	2	3	4	5	6	7	8	9	10
6	0	0	35	0	55	0	45	20	0	188
7	0	0	0	0	15	65	0	36	120	44
8	0	0	0	0	25	0	31	44	96	320
9	0	0	0	0	35	15	0	66	44	96
10	0	0	65	0	35	50	180	720	264	576
11	0	0	50	0	55	70	0	110	480	176
12	0	0	0	0	80	65	0	220	720	264

To explain the problem considered here more specifically, the additional data for the GT example are summarized in Table 1. The planning horizon consists of 10 discrete periods. At the beginning of the planning horizon, there may be initial inventories that results from remaining products, parts, and subassemblies in the previous planning horizon (Table 1(a)). Table 1(b) shows the scheduled receipts from external sources over the planning horizon, i.e., the items that are expected to arrive from outside sources and not from disassembly. Finally, the demand of leaf items at each planning period is summarized in Table 1(c). Now, using the GT example, the problem considered here can be defined as the problem of determining the ordering schedule of root item 1 and the disassembly schedule (timing and frequency of each disassembly operation) of root item 1 and parent items 2, 3, 4, and 5, while satisfying the demands of all leaf items 6, 7, 8, 9, 10, 11, and 12 with the objective of minimizing the sum of purchase, setup, inventory holding, and disassembly operation costs.

3. INTEGER PROGRAMMING MODEL

This section presents an integer programming model that represents and solves the problem considered in this paper optimally. As stated earlier, the integer program suggested here is a reversed form of the uncapacitated multi-level lot sizing problem. Here, the multi-level lot sizing problem is the problem of determining delivery times and quantities of parts and subassemblies in order to satisfy the demand of the final product [1]. Due to the divergent property explained earlier, the disassembly scheduling problem considered here is more complicated than the uncapacitated multi-level lot sizing problem that is known to be NP-hard. In addition, the time-variant cost factors, especially the setup costs with binary variables, described in the objective function of the formulation given below, make the problem more complex. Therefore, it is easy to see that the problem considered here is also NP-hard.

To formulate the problem, all items are numbered in the topological order from bottom to top and from left to right, starting with number one for the root item. That is, without loss of generality, all items are numbered by the integers 1, $2, \cdots i_l, i_l + 1, \cdots, I,$ where the index $i_l + 1$ represents the first leaf item in the disassembly product structure and hence the indices that are larger than i_l represent leaf items. In the formulation, the following notations are used.

Parameters

C_t time varying purchase cost of the product in period t ($t = 1, 2, \cdots, T$)

h_i inventory holding cost of one unit of item i for one period ($i = 1, 2, \cdots, I$)

p_i operation cost of disassembling one unit of item i ($i = 1, 2, \cdots, i_l$)

s_i cost of setting up for item i ($i = 1, 2, \cdots, i_l$)

D_{it} demand of item i in period t ($i = i_l + 1, \cdots, I$ and $t = 1, 2, \cdots, T$)

a_{ij} number of units (yield) of item j obtained by disassembling one unit of item i ($i = 1, 2, \cdots, I$, $j = 1, 2, \cdots, I$, and $i < j$)

r_{it} scheduled receipt of item i in period t ($i = 1, 2, \cdots, I$ and $t = 1, 2, \cdots, T$)

$\varphi(i)$ parent of item i ($i = 2, 3, \cdots, I$)

l_i disassembly lead time (DLT) of item i ($i = 1, 2, \cdots, i_l$)

M an arbitrary large number

I_{i0} initial inventory of item i ($i = 1, 2, \cdots, I$)

Decision variables

Z_t purchase quantity of the product in period t ($t = 1, 2, \cdots, T$)

X_{it} amount of item i disassembled in period t ($i = 1, 2, \cdots, i_l$ and $t = 1,2, \cdots, T$)

Y_{it} = 1 if the setup occurs for item i in period t, and 0 otherwise ($i = 1,2, \cdots, i_l$ and $t = 1, 2, \cdots, T$).

I_{it} inventory level of item i at the end of period t ($i = 1, 2, \cdots, I$ and $t = 1, 2, \cdots, T$)

Now, the integer program is given as follows.

$$\text{Minimize} \quad \sum_{t=1}^{T} C_t \cdot Z_t + \sum_{i=1}^{i_l}\sum_{t=1}^{T} s_i \cdot Y_{it} + \sum_{i=1}^{i_l}\sum_{t=1}^{T} p_i \cdot X_{it} + \sum_{i=1}^{I}\sum_{t=1}^{T} h_i \cdot I_{it}$$

subject to

$$I_{1t} = I_{1,t-1} + r_{1t} + Z_t - X_{1t} \quad \text{for all} \quad t = 1, 2, \cdots, T \tag{1}$$

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

$$I_{it} = I_{i,t-1} + r_{it} + a_{\varphi(i),i} \cdot X_{\varphi(i),t-l_{\varphi(i)}} - D_{it} \quad \text{for all} \quad i = i_l + 1, \cdots, I \quad \text{and } t = 1, 2, \cdots, T \tag{3}$$

$$X_{it} \le M \cdot Y_{it} \quad \text{for all} \quad i = 1, 2, \cdots, i_l \quad \text{and } t = 1, 2, \cdots, T \tag{4}$$

$$Z_t \ge 0 \text{ and integer} \quad \text{for all} \quad t = 1, 2, \cdots, T \tag{5}$$

$$X_{it} \ge 0 \text{ and integer} \quad \text{for all} \quad i = 1, 2, \cdots, i_l \quad \text{and } t = 1, 2, \cdots, T \tag{6}$$

$$I_{it} \ge 0 \text{ and integer} \quad \text{for all} \quad i = 1, 2, \cdots, I \quad \text{and } t = 1, 2, \cdots, T \tag{7}$$

$$Y_{it} \in \{0,1\} \quad \text{for all} \quad i = 1, 2, \cdots, i_l \quad \text{and } t = 1, 2, \cdots, T \tag{8}$$

The objective function denotes the sum of purchase, setup, disassembly operation, and inventory holding costs. Here, setup and disassembly operation costs are incurred for the items representing the product and its subassemblies, i.e., $i = 1, 2, \cdots, i_l$, while inventory holding costs are incurred for all items, i.e., $i = 1, 2, \cdots, I$. Constraints (1), (2), and (3) express the inventory flow conservation that defines the inventory level of item i at the end of period t. Constraint (1) represents the inventory balance of the root item. That is, at the end of each planning period, we have inventory what we had the period before, increased by the scheduled receipt and the purchased quantity and decreased by the quantity of the disassembled product (the root item). Constraint (2) represents the inventory balance of the subassemblies that should be disassembled further. Here, this constraint is the same as (1) except that for each item, the quantity resulting from disassembling its parent, multiplied by its yield from its parent, is used instead of the purchase quantity (Z_t). Also, the inventory balance of each leaf item is represented by constraint (3), which is different from (2) in that the demand requirement (D_{it}) is used instead of the amount of items to be disassembled (X_{it}). Constraint (4), where M is an arbitrary large number, guarantees that a setup cost in a period is incurred when there is at least one disassembly operation at that period. Finally, the other constraint sets (5), (6), (7), and (8) represent the conditions of the decision variables. In particular, constraint (7) ensures that backlogging is not allowed.

4. COMPUTATIONAL EXPERIMENTS

To compare the integer programming approach to the existing MRP-like algorithm, a series of computational experiments were done and the results are reported in this section. First, the GT examples with randomly generated cost values were solved and then the solutions are compared to those obtained from the algorithm suggested by Gupta and Taleb [4], called the GT algorithm in this paper. Note that additional cost values were added to the GT example (obtained from Gupta and Taleb [4]) because the original has no specific objective function. Second, the comparison is done more generally for a number of randomly generated test problems. In the test, CPLEX 6.5, a commercial integer programming software package, was used to solve the formulated integer programs. Two performance measures were employed to evaluate the results: (a) number of problems that the integer programming approach outperforms the GT algorithm; and

(b) percentage improvement of the integer programming approach over the GT algorithm.

For the test on the GT examples, 10 problems with different cost values were generated randomly. Here, purchase, setup, inventory holding, and disassembly operation costs of the example problems were generated from $DU(100, 200)$, $DU(500, 1000)$, $DU(5,10)$, and $DU(50, 100)$, respectively, where $DU(a, b)$ denotes the discrete uniform distribution with range $[a, b]$. In this test, the GT algorithm and the program to generate integer programming formulations were coded in C and the test was done on a personal computer with a Pentium processor operating at 800 MHz clock speed.

Test results for the GT examples are summarized in Table 2. It can be seen from the table that the integer programming approach gives always better solutions than the GT algorithm. This is because CPLEX 6.5 solved all the example problems optimally. The GT algorithm always gave the same solution for all the example problems because it is based on demand, not on the cost factors. Note that the different objective values of the GT algorithm result from the different cost values of the example problems. As expected, the integer programming approach required longer CPU seconds than the GT algorithm. However, all the examples were solved optimally within 30 seconds.

Table 2. Test results for the GT example

Problem	IP[1]		GT[2]		% Improvement
	Objective value	CPU seconds	Objective value	CPU seconds	
1	168383	8.76	169926	0.00[3]	0.91
2	167352	52.17	170853	0.00	2.05
3	179284	4.3	182146	0.00	1.57
4	160262	50.39	161888	0.00	1.00
5	174157	24.36	176844	0.00	1.52
6	175681	32.5	179092	0.00	1.90
7	171204	18.1	173517	0.00	1.33
8	154351	4.58	155505	0.00	0.74
9	184519	6.29	185229	0.00	0.38
10	161432	6.3	162441	0.00	0.62

[1] integer programming approach (solved by CPLEX 6.5)

[2] the GT algorithm (Gupta and Taleb [4])

[3] CPU second that is less than 0.001 seconds

To do the comparison more generally, 900 problems were generated randomly i.e., 100 problems for each combination of the three levels of the number of items (10, 20, and 30) and three levels of the number of periods (10, 15, and 20). In this test, to avoid excessive computation times, CPLEX 6.5 was terminated when CPU seconds reached 3600 seconds. In the case that CPLEX 6.5 was terminated due to the time limit, the incumbent solutions were compared to those of the GT algorithm. For each level of the number of items, 10 disassembly product structures (and hence totally 30 structures) were randomly generated. In the disassembly structures, the number of child items was generated from $DU(2, 5)$ for each parent item and the yields from the parent items were generated from $DU(1, 3)$. For each disassembly product structure, 10 problems with different data were generated for each level of the number of periods. In this test, the required data were generated as follows: (a) disassembly lead times were set to 0, 1, and 2 with probabilities 0.2, 0.7, and 0.1, respectively; (b) the initial inventory levels were generated from $DU(0, 50)$ and $DU(0, 10)$ for leaf (parts) and non-leaf (product or subassemblies) items, respectively; (c) scheduled receipt from external source for each item in each period were set to 0 or $DU(5, 10)$ with probabilities 0.3 and 0.7, respectively; and (d) demand of each leaf item in each period was set to 0 or $DU(50, 200)$ with probabilities 0.1 and 0.9, respectively.

The results on the randomly generated test problems are summarized in Table 3. It can be seen from the table that the integer programming approach outperforms the GT algorithm for all the test problems except for two with 30 items. (In these problems, CPLEX 6.5 gave incumbent solutions due to the time limit.) The overall average percentage improvement of the integer programming approach ranged from 0.62 to 2.35, which implies that the GT algorithm does not work well for the cost-based objective function. Table 4 summarizes the CPU seconds and the numbers of nodes generated by CPLEX 6.5. In the table, the CPU seconds of the GT algorithm are not reported since they are always very short, i.e. below 0.01 seconds for all the test problems. Computation times of the integer programming approach depend highly on the problem data such as disassembly structure, cost values, etc. For example, although most problems with 10 and 20 items were solved within 60 seconds, the others (with 10 and 20 items) required much longer computation times. Also, there was a distinct increase in CPU seconds when the number of items increases to 30. In fact, we can see from the two tables that the integer programming approach is useful up to problems with 20 items, especially when considering computation times.

Table 3. Test results for randomly generated test problems

(a) Problems with 10 items

Product Structure	Number of periods					
	10		15		20	
	N_{out}†	% improve	N_{out}	% improve	N_{out}	% improve
1	10	2.68 (1.82)‡	10	2.14 (1.07)	10	1.90 (0.62)
2	10	1.32 (0.85)	10	1.55 (0.57)	10	1.24 (0.46)
3	10	2.93 (1.11)	10	2.26 (1.53)	10	2.05 (0.91)
4	10	3.25 (1.41)	10	2.64 (1.43)	10	2.51 (0.93)
5	10	1.64 (0.81)	10	1.76 (0.98)	10	1.63 (0.70)
6	10	2.18 (1.06)	10	2.40 (1.14)	10	2.02 (0.70)
7	10	2.22 (1.26)	10	2.30 (1.13)	10	2.28 (1.29)
8	10	2.60 (1.52)	10	2.77 (1.05)	10	2.16 (0.48)
9	10	2.57 (1.47)	10	1.76 (0.73)	10	1.90 (0.62)
10	10	2.17 (1.48)	10	2.20 (0.99)	10	1.68 (0.68)
Mean	10.0	2.35 (1.28)	10.0	2.18 (1.06)	10.0	1.94 (0.74)

† number problems that the IP approach outperforms the GT algorithm

‡ average and standard deviation (in parenthesis) of the percentage improvement of the IP over the GT algorithm

(b) Problems with 20 items

Product Structure	Number of periods					
	10		15		20	
	N_{out}†	% improve‡	N_{out}	% improve	N_{out}	% improve
1	10	1.25 (0.79)	10	1.44 (0.86)	10	0.93 (0.44)
2	10	1.92 (1.46)	10	1.67 (0.62)	10	1.89 (0.72)
3	10	1.63 (0.80)	10	1.41 (0.51)	10	1.02 (0.35)
4	10	1.49 (0.73)	10	1.34 (0.49)	10	0.63 (0.27)
5	10	1.30 (0.52)	10	1.32 (0.52)	10	1.50 (0.68)
6	10	1.18 (0.79)	10	0.84 (0.53)	10	0.74 (0.38)
7	10	1.43 (0.57)	10	1.25 (0.48)	10	1.29 (0.39)
8	10	1.17 (0.70)	10	1.29 (0.40)	10	0.74 (0.44)
9	10	0.47 (0.25)	10	0.58 (0.27)	10	0.41 (0.16)
10	10	0.51 (0.16)	10	0.34 (0.21)	10	0.29 (0.15)
Mean	10.0	1.24 (0.68)	10.0	1.15 (0.49)	10.0	0.95 (0.40)

See the footnotes of (a).

(3) Problems with 30 items

Product Structure	Number of periods					
	10		15		20	
	N_{out}†	% improve‡	N_{out}	% improve	N_{out}	% improve
1	10	0.67 (0.20)	10	0.51 (0.24)	10	0.36 (0.13)
2	10	0.93 (0.60)	10	0.71 (0.32)	10	0 .79 (0.37)
3	10	1.12 (0.69)	10	0.89 (0.45)	10	0.98 (0.50)
4	10	1.09 (0.98)	10	0.63 (0.21)	10	0.77 (0.31)
5	10	0.57 (0.25)	10	0.50 (0.23)	10	0.40 (0.16)
6	10	1.50 (0.73)	10	0.78 (0.34)	10	1.14 (0.48)
7	10	0.86 (0.51)	10	0.96 (0.49)	10	0.67 (0.25)
8	10	0.70 (0.31)	9	0.05 (0.67)	10	0.43 (0.34)
9	10	0.55 (0.57)	10	0.58 (0.35)	10	0.35 (0.16)
10	10	0.65 (0.20)	10	0.48 (0.21)	9	0.27 (0.20)
Mean	10.0	0.86 (0.50)	9.9	0.61 (0.35)	9.9	0.62 (0.29)

See the footnotes of (a).

Table 4. CPU seconds of the integer programming approach (using CPLEX 6.5)

(a) Problems with 10 items

Product Structure	Number of periods					
	10		15		20	
	mean (min, max)	NN†	mean (min, max)	NN	mean (min, max)	NN
1	0.0 (0.0, 0.2)*	8.4	0.1 (0.1, 0.1)	14.6	0.2 (0.1, 0.2)	22.1
2	3.4 (0.0, 32.8)	10183.6	17.8 (0.1, 175.7)	43299.3	28.0 (0.3, 266.2)	43573.7
3	0.1 (0.0, 0.2)	42.9	1.1 (0.2, 6.1)	1563.5	16.8 (0.4, 160.4)	18295.6
4	0.0 (0.0, 0.1)	13.1	0.1 (0.1, 0.1)	18.2	0.1 (0.1, 0.2)	20.2
5	0.0 (0.0, 0.0)	6.8	0.0 (0.0, 0.1)	13.2	0.1 (0.1, 0.1)	21.8
6	0.0 (0.0, 0.1)	22.6	0.2 (0.1, 0.3)	178.9	0.7 (0.2, 1.4)	608.5
7	0.0 (0.0, 0.0)	4.5	0.0 (0.0, 0.1)	8.3	0.1 (0.0, 0.1)	17.7
8	1.1 (0.1, 3.3)	1727.9	108.7 (0.2, 600.4)	130686.6	243.2 (5.3, 600.4)	264084.7
9	0.0 (0.0, 0.0)	2.5	0.0 (0.0, 0.0)	3.3	0.0 (0.0, 0.0)	4
10	0.0 (0.0, 0.0)	10.4	0.0 (0.0, 0.1)	10.5	0.1 (0.0, 0.1)	19.2

† Average number of nodes out of 10 problems (generated by the CPLEX 6.5)

* 0.0 implies that CPU second was below 0.1 seconds

(b) Problems with 20 items

Product Structure	Number of periods					
	10		15		20	
	mean (min, max)	NN†	mean (min, max)	NN	mean (min, max)	NN
1	0.1 (0.1,0.2)	17.1	1.4 (0.2, 11.8)	1845.2	0.6 (0.4, 2.4)	169.6
2	0.1 (0.1, 0.1)	17.3	0.3 (0.2, 0.4)	40.5	0.5 (0.3, 0.7)	58.7
3	26.5 (0.1, 173.5)	66866.3	13.6 (0.7, 126.0)	12643.1	64.7 (1.9, 327.5)	43683.9
4	0.1 (0.0, 0.1)	10.6	0.1 (0.1, 0.2)	18.9	0.3 (0.2, 0.5)	35.3
5	0.1 (0.1, 0.2)	46.2	0.5 (0.2, 1.1)	175.1	2.2 (0.5, 6.2)	819.9
6	0.1 (0.0, 0.1)	14.6	0.3 (0.1, 0.4)	54.6	0.4 (0.2, 0.9)	60.1
7	0.1 (0.1, 0.2)	30.3	0.4 (0.3, 0.5)	75	1.0 (0.6, 1.5)	195.6
8	0.1 (0.1, 0.1)	14.7	0.3 (0.2, 0.4)	34.9	0.5 (0.4, 0.6)	43.6
9	12.8 (0.1, 126.5)	20099.2	1.4 (0.3, 4.0)	777.8	162.7 (0.9, 1283.7)	104934
10	0.9 (0.2, 3.0)	918	24.4 (1.9, 86.6)	15458.6	166.7 (2.5, 601.8)	80555.1

See the footnotes of (a).

(c) Problems with 30 items

Product Structure	Number of periods					
	10		15		20	
	mean (min, max)	NN†	mean (min, max)	NN	mean (min, max)	NN
1	0.8 (0.2, 4.5)	455.4	5.5 (0.6, 20.6)	2257.4	44.6 (3.3, 371.8)	13177.5
2	0.2 (0.1, 0.5)	69.9	0.5 (0.3, 0.8)	127.5	3.2 (0.7, 9.7)	1140.5
3	0.2 (0.1, 0.3)	47.7	1.8 (0.8, 4.1)	824.8	10.5 (2.1, 22.6)	3610.3
4	0.3 (0.2, 0.9)	213.2	2.9 (0.9, 9.9)	1388.9	30.6 (2.6, 80.7)	11769.4
5	0.5 (0.3, 1.0)	240.4	54.7 (1.1, 528.9)	32435.7	119.8 (5.8, 350.0)	55411.8
6	2.5 (1.0, 5.4)	1835.4	476.3 (152.1, 601.0)	235612.3	601.7 (600.4, 611.4)	216802.6
7	1.0 (0.2, 6.7)	1210.7	13.5 (1.5, 52.2)	7656.6	374.9 (15.5, 600.6)	152462
8	11.1 (3.3, 24.6)	8217.7	3600.3 (3600, 3600.5)	3249980.2	3600.5 (3600.4, 3600.7)	2332745.1
9	55.0 (0.7, 265.9)	77172.4	395.8 (10.7, 1740.9)	259528.8	718.3 (490.2,1890.6)	269283.8
10	457.6 (4.8, 2033)	446268.9	843.5 (155.5, 3600.5)	418646.6	1521.8 (600.3, 3389.4)	353310

See the footnotes of (a).

5. CONCLUDING REMARKS

This paper considered disassembly scheduling, which is the problem of determining the ordering and the disassembly schedules of used or end-of-life products while satisfying the demands of their parts/components over a given planning horizon. A cost-based objective function, i.e., minimizing the sum of time-varying purchase, setup, disassembly operation, and inventory holding costs, was considered for a single product type with assembly structure. To represent and solve the problem optimally, an integer programming model was presented in this paper. To compare the integer programming approach to the existing MRP-like algorithm, computational experiments were done on the example derived from a literature (with additional cost values) and a number of randomly generated test problems, and the results showed that the integer programming approach outperforms the existing MRP-like algorithm.

Because the problem considered in this paper is the most basic form of the disassembly scheduling problem, this research can be extended in several ways. First, it is needed to consider the case with general product structure, i.e., parts commonality is allowed. As stated earlier, the parts commonality introduces one or more alternative procurement sources for each common part and hence makes the problem difficult to solve. For example, see Lee *et al.* [10]. Second, the resource capacity constraint should be considered as a practical concern [12]. Third, it can be seen from the test results of this paper that the commercial software like CPLEX fails to solve large sized problems, and hence it is needed to develop fast heuristics that can give near optimal solutions. For example, see Lee and Xirouchakis [11]. Finally, uncertainties such as stochastic demands, stochastic lead-times, are important factors to be considered in disassembly scheduling.

REFERENCES

[1] Bahl, H. C., L. P. Ritzman, and J. N. D. Gupta, "Determining Lot Sizes and Resource Requirements: a Review," *Operations Research* 35 (1987), 329-345

[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] Guide, Jr., V. D. R., "Production Planning and Control for Remanufactur-

ing: Industry Practice and Research Needs," *Journal of Operations Management* 18 (2000), 467-483.

[4] Gupta, S. M. and K. N. Taleb, "Scheduling Disassembly," *International Journal of Production Research* 32 (1994), 1857-1886

[5] Johnson, M. R. and M. H. Wang, "Economical Evaluation of Disassembly Operations for Recycling, Remanufacturing and Reuse," *International Journal of Production Research* 36 (1998), 3227-3252.

[6] Kang, J.-G., D.-H. Lee, P. Xirouchakis, and A. J. D. Lambert, "Optimal Disassembly Sequencing with Sequence Dependent Operation Times based on the Directed Graph of Assembly States," *Journal of the Korean Institute of Industrial Engineers* 28 (2002), 264-273.

[7] Kim, H.-J., D.-H. Lee, P. Xirouchakis, and R. Zust, "Disassembly Scheduling with Multiple Product Types," *CIRP Annals – Manufacturing Technology* 52 (2003), 403-406.

[8] Lambert, A. J. D., "Optimal Disassembly of Complex Products," *International Journal of Production Research* 35 (1997), 2509-2523.

[9] Lee, D.-H., J.-G. Kang, and P. Xirouchakis, "Disassembly Planning and Scheduling: Review and Further Research," *Proceedings of the Institution of Mechanical Engineers Part B: Journal of Engineering Manufacture* 215 (2001), 695-710.

[10] Lee, D.-H., H.-J. Kim, G.. Choi, and P. Xirouchakis, "Disassembly Scheduling: Integer Programming Models," *Proceedings of the Institution of Mechanical Engineers Part B: Journal of Engineering Manufacture* 218 (2004), 1357-1372.

[11] 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.

[12] Lee, D.-H., P. Xirouchakis, and R. Zust, "Disassembly Scheduling with Capacity Constraints," *CIRP Annals – Manufacturing Technology* 51 (2002), 387-390

[13] Lund, R. T., "Remanufacturing," *Technology Review* 87 (1984), 18-28.

[14] 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.

[15] 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.

[16] Pnueli, Y. and E. Zussman, "Evaluating the End-of-Life Value of a Product

and Improving it by Redesign," *International Journal of Production Research* 35 (1997), 921-942.

[17] Santochi, M., G. Dini, and F. Failli, "Computer Aided Disassembly Planning: State of the Art and Perspectives," *CIRP Annals – Manufacturing Technology* 51 (2002), 1-23.

[18] Taleb, K. N. and S. M. Gupta, "Disassembly of Multiple Product Structures," *Computers and Industrial Engineering* 32 (1997), 949-961.

[19] 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.