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. To optimally solve the basic case, i.e., single product type without parts commonality, an integer programming model is suggested for the objective of minimizing the sum of purchase, setup, inventory holding, and disassembly operation costs. Then, the basic model is extended to consider the cases of single and multiple product types with parts commonality. Computational experiments done on the examples obtained from the literature show that the integer programming approach suggested in this paper works well.

1. Introduction
Due to increasing legislation pressures to collect and recover used or end-of-life products in an environmentally conscious way, manufacturing firms have been paying considerable attention to material and product recovery processes. The material recovery, i.e., recycling, aims to recover the material content of used products, while the product recovery, i.e., remanufacturing, preserves product or its parts/components by performing the required disassembly, sorting, refurbishing and assembly operations (Gungor and Gupta 1999). In both types of recovery processes, disassembly is an important step since used or end-of-life products are disassembled before recycled or remanufactured.

This paper focuses on disassembly scheduling, which is the problem of determining the ordering and disassembly schedules of used or end-of-life products while satisfying the demand of their parts/components over a certain planning horizon. Disassembly scheduling is one of the important production planning problems in disassembly systems. In other words, from its solution, we can determine which products, how many, and when to disassemble products and their subassemblies.

There are several previous research articles on the disassembly scheduling problem. Gupta and Taleb (1994) consider the basic case, i.e., single product type without parts commonality, and suggest an MRP-like algorithm. In their 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, which is repeated from the level of parts/components to the product level. Later, Taleb et al. (1997) extend the basic case by considering the parts commonality, i.e., single product type with parts commonality, and suggest another MRP-like algorithm for the objective of minimizing the number of products to be disassembled. Here, the extended problem is more complicated than the original one since the parts commonality may result in two or more alternative procurement sources for each common item and hence create additional
interdependencies between parts/components. For this extended problem, Neundorf et al. (2001) suggest an algorithm based on the Petri-net, and show using the example of Taleb et al. (1997) that their algorithm gives a better solution than the existing MRF-like algorithm. Taleb and Gupta (1997) consider the case of multiple product types with parts commonality, and suggest a two-phase heuristic algorithm for two independent objectives of minimizing the number of products to be disassembled and minimizing the disassembly costs. Recently, Lee et al. (2002) suggest an integer programming model for the problem with the resource capacity constraint.

Three cases are considered in this paper. The most basic one is single product type without parts commonality, which is the same as that of Gupta and Taleb (1994) except for the objective function. Unlike the simple objective of minimizing the number of products to be disassembled, we consider various costs, such as purchase, setup, inventory holding, and disassembly operation costs, which are commonly used in ordinary production planning problems. The next one is single product type with parts commonality, which is the same as that of Taleb et al. (1997) and Neundorf et al. (2001) except for the objective function. In general, the parts commonality is a prevalent characteristic of products since it gives a lower production cost and a more flexible production system, and hence it should not be overlooked in disassembly scheduling (Taleb et al. 1997). Finally, we consider the case of multiple product types with parts commonality, which is the same as that of Taleb and Gupta (1997) except for the objective function.

To represent and optimally solve the basic case of the problem, an integer programming model is suggested for the cost-based objective function. The basic idea is that the disassembly scheduling problem considered here can be regarded as a reverse form of the multi-level lot-sizing problem. Note that multi-level lot-sizing is the problem of determining the number of parts/components to be assembled in order to satisfy the demand of the final product. See Bahl et al. (1987) for a review on various multi-level lot-sizing problems. Also, based on the model for the basic case, two more integer programming models are suggested for the cases with parts commonality, i.e., single and multiple product types with parts commonality. As stated earlier, the objective is to minimize the sum of purchase, setup, inventory holding, and disassembly operation costs. Note that each of the three integer programs can be easily modified to consider the existing objective of minimizing the number of products to be disassembled or minimizing product disassembly costs. To show the performance of the integer programming approach suggested in this paper, computational experiments are done on the examples obtained from the literature and the results are reported.

2. Problem Description

This section begins with the definition of the disassembly product structure. In this structure, the root item represents the product itself to be disassembled and each leaf item represents an item that cannot be further disassembled. Also, a child item represents any item that has a parent at the next higher level and a parent item is any item that has at least two child items at the next lower level. Three examples of the disassembly product structure are shown in Fig. 1. Fig. 1(a) shows an example of single product type without parts commonality, to be denoted as the GT example in this paper, which is obtained from Gupta and Taleb (1994). In this figure, item 1 represents the root item, and items 6, 7, 8, 9, 10, 11, and 12 represent the leaf items. The number in parenthesis represents 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. Also, the disassembly lead-time (DLT) is the time required to disassemble a certain parent item. Note that each item has at most one parent in the case without parts commonality. Fig. 1(b) shows an example of single product type with parts commonality, to be denoted as the TGB example in this paper, which is obtained from Taleb et al. (1997). In this figure, the shaded boxes represent common items. For example, item 5 is a common item since it has more than one parent, i.e., item 5 can be obtained from either item 2 or 3. Finally, an example of multiple product types with parts commonality, to be denoted as the TG example in this paper, is shown in Fig. 1(c), which is obtained from Taleb and Gupta (1997). Note that the TG example has three root items.

Now, the problem considered in this paper can be described as follows: for a given disassembly product structure, we want to determine the ordering schedule for the root item and the disassembly schedules for all parent items while satisfying the demand of leaf items over a certain planning horizon with the objective of minimizing the sum of purchase, setup, inventory holding, and disassembly operation costs. Here, the disassembly product structure can be obtained from the disassembly process plan that specifies all parts/subassemblies of a given product and necessary disassembly operations. For reviews on disassembly process planning, see O'Shea et al. (1998), Lee et al. (2001), and Santochi et al. (2002).
items, i.e., the used or end-of-life products can be supplied whenever they are ordered. Also, as assumed in previously published research articles, we consider scheduled receipts from external sources, i.e., the items that are expected to arrive from outside sources and not from disassembly. Note that scheduled receipts from external sources are zero if all items are derived strictly from disassembly. Other assumptions made in this problem can be summarized as follows: (a) the planning horizon is divided into discrete planning periods; (b) demands for leaf items are given and deterministic; (c) backlogging is not allowed and hence demand should be satisfied on time; (d) disassembly processes are perfect, and hence no defective parts and components are obtained from disassembly; (e) disassembly lead times are given and deterministic; and (f) inventory holding costs are computed based on the end-of-period inventory.

3. The Integer Programming Approach

This section presents the integer programming models that represent and optimally solve the three cases of the disassembly scheduling problem. An integer program is suggested for the basic case of single product type without parts commonality, and then it is extended to consider the more general cases of single and multiple product types with parts commonality.

3.1 The case of single product type without parts commonality

To formulate this basic case, all items in the disassembly structure 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 integers 1, 2, ..., i, i + 1, ..., I, where i + 1 is the index for the first leaf item and hence the indices that are larger than i; represent leaf items.

The notations used in the formulation are summarized below.

Parameters

- \( c_t \): purchase cost of the product in period \( t \)
- \( s_i \): set up cost of item \( i \)
- \( h_i \): inventory holding cost of item \( i \)
- \( p_i \): disassembly operation cost of item \( i \)
- \( D_t \): demand of item \( i \) in period \( t \)
- \( a_y \): number of units (yield) of item \( i \) obtained from disassembling one unit of item \( i \)
- \( r_t \): scheduled receipt of item \( i \) in period \( t \)
- \( \alpha(i) \): parent of item \( i \)
Decision variables

\[ Z_t \quad \text{purchase quantity of the root item in period } t \]
\[ X_{it} \quad \text{disassembly quantity of item } i \text{ in period } t \]
\[ Y_{it} = 1 \text{ if there is a setup for item } i \text{ in period } t, \]
\[ 0 \text{ otherwise} \]
\[ I_u \quad \text{inventory level of item } i \text{ at the end of period } t \]

Now, the integer program [P1] for the case of a single product type without parts commonality is given below.

\[ \text{Minimize } \sum_{t=1}^{T} Z_t + \sum_{i=1}^{T} T \cdot Y_{it} + \sum_{i=1}^{T} I_u + \sum_{i=1}^{T} P_i \cdot X_{it} \]
subject to
\[ I_u = I_{u-1} + Y_{it} - X_{it} \quad \text{for } t = 1, \ldots, T \quad (1) \]
\[ I_u = I_{u-1} + Y_{it} - X_{it} \quad \text{for } i = 2, \ldots, I \text{ and } t = 1, \ldots, T \quad (2) \]
\[ I_u = I_{u-1} + Y_{it} - X_{it} \quad \text{for } i = i_1 + 1, \ldots, I \text{ and } t = 1, \ldots, T \quad (3) \]
\[ X_{it} \leq M \cdot Y_{it} \quad \text{for } i = 1, \ldots, I \text{ and } t = 1, \ldots, T \quad (4) \]
\[ Z_t \geq 0 \text{ and integer for } t = 1, \ldots, T \quad (5) \]
\[ X_{it} \geq 0 \text{ and integer for } i = 1, \ldots, I \text{ and } t = 1, \ldots, T \quad (6) \]

The objective function denotes the sum of purchase, setup, inventory holding, and disassembly operation costs. Note that the objective of the GT example obtained from Gupta and Taleb (1994) is to minimize the number of products to be disassembled and can be represented as

\[ \text{Minimize } \sum_{t=1}^{T} Z_t \quad (9) \]

Constraint (1) represents the inventory balance for the root item (product). That is, at the end of each planning period, we have the inventory that we had before, increased by the amount of scheduled receipt and purchased quantity, and decreased by the disassembly quantity of the product. Constraint (2) represents the inventory balance for the parent items that should be disassembled further. This is the same as (1) except that for each parent item, the disassembly quantity of its parent, multiplied by its yield from its parent, is used instead of the purchase quantity. Also, the inventory balance for each leaf item is represented by constraint (3), which is different from (2) in that the demand requirement is used instead of the disassembly quantity. Constraint (4), where \( M \) is an arbitrary large number, guarantees that a setup cost in each period is incurred whenever at least one disassembly operation is performed in that period. Finally, the other constraints (5), (6), (7), and (8) represent conditions on the decision variables. In particular, constraint (7) ensures that backlogging is not allowed. Note that constraints (4) and (8) are not needed when the GT example is formulated because its objective of minimizing the number of products to be disassembled does not include setup.

3.2 The case of single product type with parts commonality

This case can be formulated by modifying the inventory balance constraints (2) and (3) of [P1]. To do this, we first define an additional notation \( q(i) \) to represent the parts commonality. Unlike \( \phi(i) \) in [P1], \( q(i) \) denotes the set of parents of item \( i \). Recall that there may be two or more parents for each item in the case with parts commonality.

The integer program [P2] for the case of a single product type with parts commonality is given below.

\[ \text{Minimize } \sum_{t=1}^{T} Z_t + \sum_{i=1}^{T} T \cdot Y_{it} + \sum_{i=1}^{T} I_u + \sum_{i=1}^{T} P_i \cdot X_{it} \]
subject to
\[ I_u = I_{u-1} + Y_{it} + \sum_{k \in q(i)} a_{ik} \cdot X_{ik} - X_{it} \quad \text{for } i = 2, \ldots, I \text{ and } t = 1, \ldots, T \quad (2') \]
\[ I_u = I_{u-1} + Y_{it} + \sum_{k \in q(i)} a_{ik} \cdot X_{ik} - D_i \quad \text{for } i = i_1 + 1, \ldots, I \text{ and } t = 1, \ldots, T \quad (3') \]
and (1), (4), (5), (6), (7), and (8)
4. Computational Experiments

This section presents the test results on the three integer programs suggested in this paper. In these tests, CPLEX 6.5, a commercial software package, was used to solve the integer programs. The TGB and TG examples with the original objectives are solved, and their solutions are compared with those of Taleb et al. (1997) and Taleb and Gupta (1997), respectively. Note that no comparison was done for the GT example (the case of single product type without parts commonality) since the MP-based algorithm of Gupta and Taleb (1994) already gives the optimal solution. In the tests, the program to generate integer programs was coded in C and the tests were done on a personal computer with a Pentium processor operating at 800 MHz clock speed.

Test results for the TGB example are given in Table 1, which summarizes the ordering schedules obtained from Taleb et al. (1997) (original and corrected), Neuenendorf et al. (2001), and the integer programming approach, respectively. Note that Neuenendorf et al. (2001) report the round-off errors in the original solution of Taleb et al. (1997) and suggest a corrected solution. The optimal number of products to be disassembled, obtained from the integer programming approach, is 551, which is the same as that of the Petri-net based algorithm suggested by Neuenendorf et al. (2001). It can be seen from the table that the optimal solution obtained from the integer programming approach is different from that of Neuenendorf et al. (2001). This implies that the TGB example has multiple optimal solutions, and hence it can be argued that the cost-based objective suggested in this paper is needed to evaluate different solutions with the same number of products to be disassembled. Finally, the CPU second required to obtain the optimal solution was 0.02.

<p>| Table 1. Comparison of solutions: TGB example |</p>
<table>
<thead>
<tr>
<th>Approaches</th>
<th>Period</th>
<th>Objective value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGB(^1)</td>
<td>77</td>
<td>49</td>
</tr>
<tr>
<td>TGB(^2)</td>
<td>77</td>
<td>49</td>
</tr>
<tr>
<td>NLKX(^3)</td>
<td>77</td>
<td>15</td>
</tr>
<tr>
<td>IP(^4)</td>
<td>154</td>
<td>388</td>
</tr>
</tbody>
</table>

\(^1\) Original solution (with round-off errors) of Taleb et al. (1997)
\(^2\) Corrected solution of Taleb et al. (1997) (reported in Neuenendorf et al. (2001))
\(^3\) Solution of Neuenendorf et al. (2001)
\(^4\) Solution of the integer programming approach with the objective of minimizing the number of products to be disassembled (CPU seconds: 0.02)
Table 2 summarizes the test results for the TG example, i.e., the case of multiple product types with parts commonality. In this test, as in Taleb and Gupta (1997), the disassembly costs were set to 20, 6, and 21 for products 1, 2, and 3, respectively. The integer programming approach outperformed the two-phase heuristic of Taleb and Gupta (1997). In fact, the amounts of improvement were 33.3% and 3.75% for the cases with minimizing the number of products to be disassembled and minimizing the disassembly cost of products, respectively. Finally, CPLEX 6.5 required 0.07 seconds to obtain the optimal solution for each of the two cases.

<table>
<thead>
<tr>
<th>Products</th>
<th>Period</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1 2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1 1</td>
<td>2</td>
</tr>
</tbody>
</table>

Number of product disassembled = 6
Total disassembly cost = 20*1 + 6*2 + 21*2 = 80

(b) Integer programming approach

(Minimizing the number of products disassembled)

<table>
<thead>
<tr>
<th>Products</th>
<th>Period</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Number of product disassembled = 4
CPU seconds: 0.07

(Minimizing disassembly cost of products)

<table>
<thead>
<tr>
<th>Products</th>
<th>Period</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4 1 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Total disassembly cost = 20*1 + 6*6 + 21*1 = 77
CPU seconds: 0.07

5. Concluding Remarks

This paper considered the problem of determining the ordering and disassembly schedules of used products while satisfying the demand of their parts/components over a given planning horizon. The objective is to minimize the sum of purchase, setup, inventory holding, and disassembly operation costs, which generalizes the previous simple objective of minimizing the number of products to be disassembled. Three integer programs were suggested to represent and optimally solve three cases of the problem, i.e., single product type without and with parts commonality and multiple product types with parts commonality. Computational experiments were done on the examples obtained from the literature (for the original objectives) and the test results showed that our integer programming approach can give improved (optimal) solutions for the examples.

References


