Mathematical model and solution algorithms for selective disassembly sequencing with multiple target components and sequence-dependent setups

Hee-Jong Han, Jae-Min Yu & Dong-Ho Lee

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

Department of Industrial Engineering, Graduate School of Technology and Innovation Management, Hanyang University, Seoul, 133-791, Republic of Korea

Published online: 14 Jun 2013.

To cite this article: Hee-Jong Han, Jae-Min Yu & Dong-Ho Lee (2013) Mathematical model and solution algorithms for selective disassembly sequencing with multiple target components and sequence-dependent setups, International Journal of Production Research, 51:16, 4997-5010, DOI: 10.1080/00207543.2013.788794

To link to this article: http://dx.doi.org/10.1080/00207543.2013.788794
Mathematical model and solution algorithms for selective disassembly sequencing with multiple target components and sequence-dependent setups

Hee-Jong Han, Jae-Min Yu, and Dong-Ho Lee*

*aDepartment of Industrial Engineering, Hanyang University, Seoul 133-791, Republic of Korea; bDepartment of Industrial Engineering, Graduate School of Technology and Innovation Management, Hanyang University, Seoul 133-791, Republic of Korea

(Received 12 June 2012; final version received 18 March 2013)

This study considers selective disassembly sequencing under the sequential disassembly environment in which one component is obtained at each disassembly operation. The problem is to determine the sequence of disassembly operations to obtain multiple target components of a used or end-of-life product for the purpose of repair, reuse, remanufacturing, disposal, etc. In particular, we consider sequence-dependent setups in which setup costs depend on the disassembly operation just completed and on the operation to be processed. The problem is represented as a disassembly precedence graph and then a new integer programming model is suggested for the objective of minimising the total disassembly cost. After it is proved that the problem is NP-hard, we suggest two types of heuristics: (1) branch and fathoming algorithm for small-to-medium-sized instances; and (2) priority-rule-based algorithm for large-sized instances. A series of computational experiments, i.e., effectiveness of the new integer programming model and performances of the two heuristic types, were done on various test instances, and the results are reported. In addition, to show the applicability of the mathematical model and the solution algorithms, a case study is reported on an end-of-life electronic calculator.

**Keywords:** selective disassembly sequencing; multiple target components; sequence-dependent setups; integer programming; heuristics

1. Introduction

Reprocessing used/end-of-life products has become an important environmental issue for sustainable society, mainly due to: (1) shortened product lives due to rapid product developments; (2) shortages of dumping and incineration sites; and (3) legislative pressure to protect the environment. Among others, various legislative pressure, such as eco-design requirements for energy using products (EuP), waste electrical and electronic equipment (WEEE), the restriction of the use of certain hazardous substance in electric and electronic equipment (RoHS), and directives for end-of-life vehicles (ELV), force manufacturing and service firms to be more responsible for reusing, recycling, remanufacturing, and even disposing of used/end-of-life products in more environmentally conscious ways.

One of the key points in effective and efficient product reprocessing is disassembly, which is defined as the process of separating products into parts and/or subassemblies with necessary inspection and sorting operations. This is because used/end-of-life products are generally recovered or even disposed of after being disassembled. In other words, disassembly is usually done as the first process after collection and peer inspection when used/end-of-life products are recovered or disposed of. In general, disassembly is done for several purposes, e.g., recovering material fractions, isolating hazardous substances, and separating reusable parts and/or subassemblies.

Due to its theoretical and practical importance, a number of previous studies have been done on various decision problems in disassembly, such as design for disassembly (DFD) on easiness of disassembly operations, disassembly line balancing that assigns disassembly tasks to workstations in disassembly lines, disassembly process planning that prepares detailed operational instructions, and disassembly lot sizing or scheduling that determines the quantity and timing of disassembling used or end-of-life products to satisfy the demands of their parts and subassemblies. Here, the first two decisions, DFD and disassembly line balancing, are design issues, while the others are operational issues. For the details of the problems and relevant literature, see Jovane et al. (1993), Zhang et al. (1997), Gungor and Gupta (1999), Lee, Kang, and Xiouchakis (2001a), and Ilgin and Gupta (2010).

Among the disassembly problems explained above, this study focuses on disassembly process planning, i.e., specifying detailed operation instructions for separating used or end-of-life products into subassemblies and/or parts.

*Corresponding author. Email: leman@hanyang.ac.kr

© 2013 Taylor & Francis
Overall, disassembly process planning involves the following three main decisions (see O’Shea, Grewal, and Kaebernick 1998; Lee, Kiritsis, and Xirouchakis 2001b; Santoch, Dini, and Failli 2002; and Lambert 2003 for more details on disassembly process planning):

- **Disassembly level**: decision on whether more disassembly operations are performed or not at each stage of disassembling a product, which gives the disassembly structure that specifies parts and/or subassemblies to be obtained from disassembling products.
- **Disassembly sequence**: sequence of disassembly operations.
- **End-of-life options**: decision on how each part or subassembly is to be dealt with (e.g., reuse, refurbishing, remanufacturing, recycling, disposal, etc.).

The previous studies on disassembly process planning can be classified according to the three decision variables. Most earlier studies consider one decision variable. For examples, see Penev and de Ron (1996) and Meacham, Uzsoy, and Venkatadri (1999) for determining the disassembly level; see Gungor and Gupta (1997), Lambert (1997), Kara, Pompratipol, and Kaebernick (2005), Hui, Dong, and Guan (2008), and Adenso-Díaz, García-Carbayal, and Gupta (2008) for determining the disassembly sequence; and see Mangun and Thurston (2002), Bufardi et al. (2004), Gerner et al. (2005), and Jun et al. (2007) for selecting end-of-life options. Several articles consider two of the three decision variables. For examples, see Johnson and Wang (1995, 1998), Pnueli and Zussman (1997), Kang et al. (2001, 2002), and Lambert (2006) for determining the disassembly level and sequence. Finally, several studies consider the three decision variables at the same time. See Erdos, Kiritsis, and Xirouchakis (2001), Tenner (2006), Behdad et al. (2010) and Ma et al. (2011) for examples.

Among the three decisions on disassembly process planning, this study focuses on disassembly sequencing. In general, the previous studies on disassembly sequencing can be classified into two cases: (1) general without target components; and (2) selective with target components.


For selective disassembly sequencing, most previous studies consider the case with a single target component, i.e., a part or subassembly is obtained from disassembling a product. In particular, they are based on various geometric constraints obtained from computer aided design (CAD) data, and hence the basic approach is to enumerate all feasible solutions that satisfy the geometric constraints and select the best one, i.e., full enumeration. Srinivasan and Gadh (1998a, 1998b, 1999) suggest the wave propagation method that consists of disassembly waves to arrange the components topologically and intersection events between waves to determine the disassembly sequence that minimises the number of disassembled components. Chung and Peng (2005) extend the wave propagation method by incorporating both batch disassembly of components and tool accessibility to fasteners. Also, Kara, Pompratipol, and Kaebernick (2005, 2006) suggest another methodology that reverses and modifies an existing method for assembly and they report various case studies. Recently, Smith and Chen (2011) suggested a recursive method using the geometric and topological relationships between a part and its neighbouring parts. The method is based on disassembling to eliminate uncommon or unrealistic solutions. Also, to give good quality solutions within a reasonable amount of computation time, various meta-heuristics are suggested. For examples, see Smith and Smith (2002) and Smith (2004) for genetic algorithms, and Wang et al. (2003, 2007), Xue, Qiu, and Xiang (2007) and Zhang et al. (2007) for ant colony optimisation.

This study focuses on selective disassembly sequencing in which one or more components of interest are removed from a product for repair, reuse, remanufacturing, or even disposal. According to Woo and Dutta (1991), disassembly can be classified into serial disassembly and parallel disassembly according to the logical sequence of removing parts or subassemblies from a product. In the sequential disassembly, only one part or subassembly is disassembled at a time, while in the parallel disassembly, two or more parts or subassemblies can be disassembled at the same time. Of the two types, this study considers the problem under the sequential disassembly environment.

As an extension of the previous studies, we consider the general case in which multiple target components are obtained at the same time. Unlike complete disassembly in which all parts are obtained from disassembling a product, selective disassembly is done to obtain one or more specific target components. Therefore, in the case of selective
disassembly, disassembly levelling is not needed because we have target components to be obtained. In particular, we consider sequence-dependent setups in which setup costs depend on the disassembly operation just completed and on the disassembly operation to be processed.

To represent the problem mathematically, a new integer programming model is suggested after representing the problem as a disassembly precedence graph. After it is proved that the problem is NP-hard, two types of heuristics are suggested. They are: (1) the branch and fathoming (B&F) algorithm for small-to-medium-sized instances; and (2) the priority rule-based algorithm for large-sized instances. A series of computational experiments were done on the effectiveness of the new integer programming model and the performances of the two heuristic types and the results are reported. In particular, to show the applicability of the mathematical model and the solution algorithms, a case study is reported on an end-of-life electronic calculator.

The paper is organised as follows. In the next section, the problem is described in more detail with the integer programming model after representing the problem as the disassembly precedence graph. The two types of heuristics are presented in Section 3, and then computational results together with a case study are reported in Section 4. Finally, Section 5 concludes the paper with a summary and discussion of future research.

2. Problem description

This section explains the selective disassembly sequencing problem considered in this study. First, the disassembly precedence graph is explained to represent the problem graphically. Then, the integer programming model, together with the required assumptions, is presented. Finally, the NP-hardness of the problem is proved.

In this study, the disassembly precedence graph, which is suggested by Lambert (2006), is adopted to represent disassembly operations and their precedence relations. More formally, the disassembly precedence graph can be represented by $G = (N, A)$, where $N \setminus \{0\}$ is the set of nodes that represent disassembly operations, i.e., $N = \{0, 1, 2, \ldots, n\}$ and $A$ is the set of arcs that represent the precedence relations between two disassembly operations, i.e., $A = \{(i, j): i, j \in N$ and $i < j\}$. Node 0 denotes the root node representing the start of disassembly. Each disassembly operation is associated with a part or subassembly to be obtained from that operation since we consider the sequential disassembly, and hence the number in each node represents the corresponding part or subassembly. Also, we can see that the sequence-dependent setup cost between two disassembly operations can be represented by the value associated with the arc that represents the corresponding precedence relation. Note that the traditional and/or graph is not needed since we consider the sequential disassembly. Figure 1 shows a disassembly precedence graph for a product with nine components. In the graph, it can be seen that components 1, 5, and 7 can be obtained at the first stage of disassembling the product, and component 2 can be obtained directly after component 1 is obtained, i.e., there is a precedence relation between operations 1 and 2.

From the disassembly precedence graph, we can obtain a disassembly precedence matrix $DPM = [a_{ij}]$, where $a_{ij} = 1$ if disassembly operation $i$ precedes operation $j$, and 0 otherwise. For example, $a_{12} = 1$ since disassembly operation (component) 2 can be done directly after operation (component) 1. The resulting DPM for the disassembly precedence

![Disassembly precedence graph: An example.](image_url)
graph given in Figure 1 is shown in Figure 2(a). Also, we can specify the disassembly cost matrix DCM = [$c_{ij}$] using the disassembly precedence graph. That is, the disassembly cost $c_{ij}$ between two consecutive disassembly operations $i$ and $j$ ($a_{ij}$ = 1) consists of the sequence-dependent setup cost between the two operations and the operation cost to obtain the component associated with disassembly operation $j$. An example of the DCM for the example disassembly precedence graph is shown in Figure 2(b).

Now, the problem can be briefly explained as follows: for a given used or end-of-life product, the problem is to determine the sequence of disassembly operations to obtain multiple target components for the objective of minimising the total disassembly cost while satisfying the precedence relations among disassembly operations. In the disassembly precedence graph explained earlier, a feasible disassembly sequence can be represented as multiple paths from the root node to the nodes associated with target components. The disassembly cost, corresponding to an arc in the disassembly precedence graph, consists of the sequence-dependent setup cost and the operation cost to obtain the component. When disassembling a product or subassembly, sequence-dependent setup cost may occur due to changes in equipment or tools. Note that sequence-dependent setups are very important in disassembly systems since most disassembly operations are done manually. See Kang et al. (2001, 2002) for the importance of sequence-dependent setups in disassembly systems. Also, disassembly operation cost, which is assumed to be proportional to the labour or machine processing time, is required to perform a disassembly operation.

As with an earlier study on selective disassembly sequencing with multiple target components, this study considers the deterministic version of the problem, i.e., all problem data are deterministic and given in advance. It is assumed that the disassembly precedence graph is also given in advance, where the graph can be constructed from the product structure that specifies parts/subassemblies and their precedence relations.

To describe the problem more clearly, a new integer programming model is suggested. The model is similar to the modified two-commodity network flow model of Lambert (2006) in that both are based on the disassembly precedence graph. However, the new model is developed especially for selective disassembly sequencing with multiple target components. Before presenting the integer programming model, the notation used is summarised below.

(a) Disassembly precedence matrix (DPM)

```
0 1 0 0 0 1 0 1 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
```

(b) Disassembly cost matrix (DCM)

```
0 5 8 6 9 4 2 5 1 3
6 0 7 8 9 2 6 4 3 5
4 3 0 7 8 5 2 6 9 4
3 6 7 0 3 8 7 4 6 3
1 5 2 1 0 3 7 8 9 1
6 6 7 8 3 0 3 6 5 2
7 5 4 2 1 5 0 6 8 7
1 6 1 2 3 4 5 0 7 8
6 4 5 7 2 1 6 5 0 7
5 6 8 9 7 1 2 4 3 0
```

Figure 2. Matrix representation of the problem: An example.
Parameters

c_{ij} \quad \text{disassembly cost (sequence-dependent setup and operation costs) for operation } j \text{ directly after operation } i \text{ is done, } (i,j) \in A

PR_{ij} = 1 \text{ if operation } j \text{ follows operation } i \text{ according to the precedence relation, and } 0 \text{ otherwise}

T \quad \text{set of target components}

Decision variables

x_{ij} = 1 \text{ if disassembly operation } j \text{ is to be performed directly after operation } i, \text{ and } 0 \text{ otherwise}

y_{i} = 1 \text{ if disassembly operation } i \text{ is to be performed, and } 0 \text{ otherwise}

z_{ij} = 1 \text{ if disassembly operation } j \text{ is to be performed after operation } i, \text{ and } 0 \text{ otherwise}

Now, the integer programming model is given below. As explained earlier, \( N \) and \( A \) are the sets of nodes and arcs in the disassembly precedence graph.

\[ \text{[P]} \quad \text{Minimise} \quad \sum_{(i,j) \in A} c_{ij} \cdot x_{ij} \]

subject to

\[ \sum_{i \in N} x_{ij} = y_{j} \quad \text{for all } j \in N \quad (1) \]

\[ \sum_{j \in N} x_{ij} = y_{i} \quad \text{for all } i \in N \quad (2) \]

\[ y_{j} = 1 \quad \text{for all } j \in T \quad (3) \]

\[ PR_{ij} \cdot y_{j} \leq y_{i} \quad \text{for all } i,j \in N \quad (4) \]

\[ PR_{ij} \cdot z_{ji} = 0 \quad \text{for all } i,j \in N \quad (5) \]

\[ z_{ij} + z_{ji} = 1 \quad \text{for all } i,j \in N \setminus \{0\} \quad (6) \]

\[ z_{ij} + z_{jk} - z_{ki} \leq 1 \quad \text{for all } i \in N, j,k \in N \setminus \{0\} \quad (7) \]

\[ z_{ij} - x_{ij} \geq 0 \quad \text{for all } i,j \in N \quad (8) \]

\[ x_{ii} = 0 \quad \text{for all } i \in N \quad (9) \]

\[ x_{ij}, y_{i}, z_{ij} \in \{0,1\} \quad \text{for all } i,j \in N \quad (10) \]

The objective function represents the total disassembly cost, i.e., the sum of sequence-dependent setup and operation costs until the target components are obtained. Constraints (1) and (2) represent the relations between two decision variables \( x_{ij} \) and \( y_{j} \), i.e., input and output flow conservations in the disassembly precedence graph. Constraint (3) ensures that target components are obtained from the product. Constraints (4), (5), (6), and (7) represent the precedence relationships. More specifically, constraint (4) represents the precedence relationship between two consecutive operations, i.e., if an operation is performed, the operation directly preceding it must be performed, and constraint (5) implies that the reverse precedence is not allowed for two operations with a given precedence relation. Constraint (6) denotes that forward and reverse precedence relations are mutually exclusive, which is redundant because of constraint (5) but which increases the tightness of the formulation. Constraint (7) represents the triangular relations among three operations.
Constraint (8) denotes the coupling constraint between two decision variables \( z_{ij} \) and \( x_{ij} \), and constraint (9) denotes that there is no precedence relation for an operation. Finally, constraint (10) represents the condition of decision variables.

The problem \([P]\) can be solved using a commercial software package. However, it is impractical due to the heavy computational burden since the problem is NP-hard. The complexity of the problem is shown in the following theorem.

**Theorem 1:** The problem \([P]\) is NP-hard.

**Proof:** The asymmetric Hamiltonian path problem with precedence constraints (AHPP-PC) is the problem of determining the shortest path from a source to a destination node while visiting all nodes exactly once in a mixed network with directed and undirected arcs. Since it is not necessary to visit all nodes in problem \([P]\), we can construct the AHPP-PC using the required nodes in the disassembly precedence graph. Therefore, the AHPP-PC is a special case of the problem \([P]\). Since the AHPP-PC is proved to be NP-hard (Ascheuer 1995), the problem \([P]\) is NP-hard. 

3. Solution algorithms

This section presents the two types of heuristic algorithms: (1) the branch and fathoming algorithm; and (2) the priority-rule-based algorithm. As explained earlier, the branch and fathoming algorithm is useful for small-to-medium-sized instances, while the priority rule-based algorithm is for large-sized instances.

3.1 Branch and fathoming algorithm

The branch and fathoming algorithm enumerates all possible feasible sequences that satisfy the precedence relations and finds the best sequence while curtailing some sequences using the fathoming bound (Lee, Kiritsis, and Xirouchakis 2001). First, we explain the branching scheme that generates all possible sequences. Then, the methods to obtain the upper and fathoming bounds are explained.

3.1.1 Branching

The B&F tree is used to generate all possible (feasible and infeasible) sequences. In the tree, each node (except for the root node) denotes the disassembly operation, and each level corresponds to a position in disassembly sequences. For example, the nodes in the first level of the B&F tree correspond to the candidate operations for the first position of operation sequences. There is a single root node (level number 0), and a (parent) node of level \( l \) has child nodes of level \( l+1 \). Therefore, the largest level number in the B&F tree is the total number of disassembly operations. In fact, the number of all possible sequences is \( n! \), where \( n \) is the number of disassembly operations. Also, each node in the B&F tree corresponds to a partial sequence, which can be found by tracing a path from that node to the root node. Each time a new operation is added on a partial sequence, the B&F algorithm proceeds from one level to the next. Figure 3 shows an example of the B&F tree that represents all possible sequences for an instance with three disassembly operations.

**Figure 3.** B&F tree for an instance with three disassembly operations.
In the B&F tree, the number of feasible sequences is smaller than \( n! \) due to the target components and the precedence relations among disassembly operations. In Figure 3, for example, if a target component is extracted from operation 2, the sequence 0\( \rightarrow \)1\( \rightarrow \)2 is feasible since the target component is obtained by operations 1 and 2, and hence the complete sequence 0\( \rightarrow \)1\( \rightarrow \)2\( \rightarrow \)3 needs not be considered. Also, to eliminate infeasible sequences that violate precedence relations, it is needed to find the set of feasible candidates, i.e., a set of operations satisfying the precedence relation at each node of the B&F tree. Therefore, if the node violates the precedence relation, the node must be fathomed. For example, in Figure 3 if a partial solution of the B&F tree is 0\( \rightarrow \)1\( \rightarrow \)2, the set of feasible candidates at node 2 is \{3, 4, 5, 7\}.

The branch and backtrack method is used for node selection, i.e., branching. In this method, if the partial solution does not include all target components, the next node to be considered is one of its child nodes. Otherwise, it is needed to go back on the path from this node toward the root node until the first node with a child node is found that has not been considered yet. Note that the problem is to find the disassembly sequence that minimises the total disassembly cost from the root node to target components at the same time.

### 3.1.2 Fathoming

The B&F algorithm suggested in this study uses upper and fathoming bounds. Instead of using a lower bound to find the optimal solutions, we use another bound, called the fathoming bound, with which unpromising sequences are curtailed at each node of the B&F tree. As in the ordinary branch and bound algorithm, each node of the B&F tree can be deleted from further consideration (fathomed) if the fathoming bound at the node is greater than or equal to the current upper bound.

The initial upper bound is obtained by a greedy algorithm based on the nearest neighbour heuristic for the travelling salesman problem while considering the precedence relations. In other words, a feasible solution is obtained by specifying the set of feasible disassembly operations that can be performed directly after the last operation of the partial sequence and adding the one with the minimum disassembly cost until all target components are obtained. Note that the upper bound is updated whenever a better feasible solution is obtained.

The fathoming bound is obtained at each node of the B&F tree by adding the remaining operations to the corresponding partial sequence without considering the precedence relations. In other words, as in the method to obtain the initial upper bound, the operation with the minimum disassembly cost is sequentially added to the partial sequence corresponding to the current node of the B&F tree without considering the precedence relations. More formally, the fathoming bound at a node of the B&F tree can be represented as

\[
C_{PS} + C_{RO},
\]

where \( C_{PS} \) is the cost obtained from the partial sequence corresponding to that node and \( C_{RO} \) is the cost derived from the disassembly operations not in the partial schedule. Let \( i \) and \( RO \) denote the last operation of the partial sequence and the set of disassembly operations not in the partial schedule. Then, \( C_{RO} \) is calculated as

\[
C_{RO} = \alpha \cdot n \cdot c_{j^*},
\]

where \( n \) is the number of disassembly operations and \( j^* = \arg\min_{j \in RO} \{c_j\} \). In the fathoming bound, \( \alpha \) represents the tightness of the fathoming bound. In other words, a higher \( \alpha \) increases the fathoming bound and hence more nodes in the B&F tree are fathomed. However, it results in poor solution quality (the test results on different values of \( \alpha \) are given in the next section). As mentioned above, if the fathoming bound is larger than or equal to the current upper bound at a node, the node is fathomed.

### 3.2 Priority rule-based algorithm

Although the B&F algorithm explained above may give good quality solutions, it is adequate for small-to-medium-sized instances due to its excessive computation time, depending on the parameter \( \alpha \). Therefore, we suggest another fast heuristic, called the priority-rule-based algorithm, to solve practical large-sized instances. Unlike the B&F algorithm, the priority-rule-based algorithm uses a priority rule to select the next operation at each node of the B&F tree. Although the priority-rule-based algorithm has a disadvantage in that its performance is not guaranteed, it is much more applicable to
practical situations because it is easy to implement, simple enough to be understood, and also yields reasonable quality solutions with very short computation times.

In this study, we suggest the following five priority rules to select the next disassembly operation while satisfying the precedence relations. In the following, $i$ and $FO$ denote the last operation of the current partial sequence and the set of feasible disassembly operations that can be connected to the end of the current partial sequence, respectively. Also, $RC_j$ denotes the remaining cost at disassembly operation $j$, which is calculated by the nearest neighbour algorithm to obtain the initial upper bound of the B&F algorithm, i.e., the total cost is obtained by adding the disassembly operation with the minimum cost to the end of the current partial sequence starting from disassembly operation $j$ until all target components are obtained.

The initial upper bound is obtained by a greedy algorithm based on the nearest neighbour heuristic for the travelling salesman problem while considering the precedence relations. In other words, a feasible solution is obtained by specifying the set of feasible disassembly operations that can be performed directly after the last operation of the partial sequence and adding the one with the minimum disassembly cost until all target components are obtained.

RAN select an operation randomly (benchmark rule)

SC select an operation with the smallest disassembly cost, i.e., select an operation $j^*$ such that

$$ j^* = \arg\min_{j \in FO} \{c_j\} $$

SR select an operation with the smallest remaining cost, i.e., select an operation $j^*$ such that

$$ j^* = \arg\min_{j \in FO} \{RC_j\} $$

SC/R select an operation with the smallest value of disassembly operation cost divided by remaining cost, i.e., select an operation $j^*$ such that

$$ j^* = \arg\min_{j \in FO} \{c_j / RC_j\} $$

LC/R select an operation with the largest value of disassembly operation cost divided by remaining cost, i.e., select an operation $j^*$ such that

$$ j^* = \arg\min_{j \in FO} \{c_j / RC_j\} $$

4. Computational experiments

To test the performances of the new integer programming formulation and the two types of heuristics suggested in this study, computational experiments were carried out and the results are reported in this section. First, the new integer programming model is compared with the existing two-commodity network flow-based one by Lambert (2006). Second, the performances of the two types of heuristics are tested on a number of test instances. Finally, we report a case study on an end-of-life electronic calculator. The heuristics were coded in C, and the tests were done on a personal computer with a Pentium dual-core processor operating at 3.20 GHz.

The first test is on the performance of the new integer programming formulation $[P]$ that can give the optimal solutions for small-sized instances. For this purpose, we generated 60 instances randomly, i.e., 10 instances for each of the six combinations for the two levels of the number of disassembly operations (20 and 30) and three levels of the number of target components (1, 3, and 5). The disassembly precedence graphs were generated by randomly connecting child nodes to each parent node under the given total number of disassembly operations. Here, the number of child nodes for each parent was randomly generated from $DU(0, 5)$, where $DU(a, b)$ denotes the discrete uniform distribution with range $[a, b]$. Therefore, various concentration ratios of DPMs were considered in this test. Also, the disassembly operation costs (with sequence-dependent setup costs) were generated from $DU(20, 100)$. 
Test results are summarised in Table 1 which shows the number of instances where the CPLEX 11.0 found the optimal solutions within 3600 s (using the existing and the new formulations) and their CPU seconds. As can be seen in the table, unlike the existing model, the new model found the optimal solutions for all test instances. Also, its computation times were much shorter. In fact, the average computation times of the existing and the new formulations are 476.2 and 4.1 s. In summary, we can see that the new model is much better than the existing one.

For the second test on the performance of the B&F algorithm, we generated an additional 60 instances, i.e. 10 instances for each of the six combinations for the two levels of the number of disassembly operations (40 and 50) and three levels of the number of target components (1, 3, and 5). Hence, we tested the B&F algorithm on 120 instances in total. The disassembly precedence graphs and the problem data were generated using the method explained earlier. Also, we tested the B&F algorithm under three levels of parameter \( \alpha \), i.e., tightness of fathoming bound (0.025, 0.030, and 0.035), which were obtained from a preliminary test.

Test results are summarised in Table 2 which shows the percentage gaps from the optimal solution values and CPU seconds. In this test, the optimal solutions were obtained using CPLEX 11.0 within a time limit of 3600 s using the new integer programming model. It can be seen from the table that the performance of the B&F algorithm depends on the parameter \( \alpha \) in the fathoming bound, i.e., a lower (higher) \( \alpha \) value results in better (worse) performance and longer (shorter) computation time. In fact, the average percentage gaps (average computation times) were 1.77 (178.4), 2.72

![Figure 4. The electronic calculator.](image-url)

Table 1. Comparison of the existing and the new formulations.

<table>
<thead>
<tr>
<th>Number of operations</th>
<th>Number of targets</th>
<th>Existing formulation (Lambert 2006)</th>
<th>New formulation (Model [P])</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( N_{\text{opt}} )</td>
<td>CPU(^1)</td>
<td>( N_{\text{opt}} )</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>10 (2.6, 14.5)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9 (133.6, 500.5)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7 (268.6, 664.4)</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>7 (451.5, 1703.7)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4 (1222.8, 1725.5)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1 (769.9, 769.9)</td>
<td>10</td>
</tr>
<tr>
<td>Average</td>
<td>6.3</td>
<td>476.2</td>
<td>10</td>
</tr>
</tbody>
</table>

\(^1\)Number of instances out of 10 where the CPLEX found the optimal solutions within 3600 s.

\(^2\)Average CPU seconds of 10 instances (minimum and maximum in parenthesis).
(60.7), and 6.86 (9.0) when the parameter \( \alpha \) values were 0.025, 0.030, and 0.035, respectively. Therefore, it is needed to find an appropriate \( \alpha \) value when the B&F algorithm is used.

The third test is on the performances of the priority-rule-based algorithm that gives fast solutions. For this test, we generated an additional 150 instances, i.e. 10 instances for each of the fifteen combinations for the five levels of the number of disassembly operations (100, 200, 300, 400, and 500) and three levels of the number of target components (1, 3, and 5). The required data were generated using the method explained earlier. For an evaluation of the results, the

**Table 2. Performance of the B&F algorithm.**

<table>
<thead>
<tr>
<th>Number of operations</th>
<th>Number of targets</th>
<th>B&amp;F algorithm ((x = 0.025))</th>
<th>B&amp;F algorithm ((x = 0.030))</th>
<th>B&amp;F heuristic ((x = 0.035))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Gap¹ CPU²</td>
<td>Gap CPU</td>
<td>Gap CPU</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>0.02 (&lt; 0.01, &lt; 0.05) 4.41 &lt;0.01 (&lt; 0.01, 0.03)</td>
<td>0.03 (&lt; 0.01, 0.1) 0.26</td>
<td>0.1 (&lt; 0.01, 0.5)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.04 (&lt; 0.01, 0.11) 0.67 0.32 (&lt; 0.01, 1.1)</td>
<td>0.19 (0.03, 0.6)</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.2 (&lt; 0.01, 0.9) 1.58 2.85 (0.06, 11.3)</td>
<td>0.19 (0.03, 5.9)</td>
<td>3.62</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>0.07 0.6 (&lt; 0.01, 3.7) 0.67 0.32 (&lt; 0.01, 1.1)</td>
<td>0.25 0.1 (&lt; 0.01, 0.5)</td>
<td>13.87</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.21 5.4 (0.09, 18.3) 0.46 2.85 (0.06, 11.3)</td>
<td>0.19 (0.03, 11.3)</td>
<td>1.39</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.40 7.5 (0.05, 23.8) 1.58 3.81 (0.03, 15.9)</td>
<td>0.26</td>
<td>0.1 (&lt; 0.01, 0.5)</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>1.81 65.6 (0.11, 546.4) 0.67 4.28 (0.02, 113.5)</td>
<td>0.19 (0.03, 11.3)</td>
<td>10.62</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.14 222.7 (0.2, 1169.8) 4.35 58.51 (0.65, 408.9)</td>
<td>0.25</td>
<td>0.1 (&lt; 0.01, 0.5)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.05 145.8 (5.4, 801.1) 2.92 57.32 (1.67, 439.6)</td>
<td>1.39</td>
<td>1.3 (0.03, 4.6)</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>3.84 270.4 (1.8, 1113.90) 4.7 29.21 (0.2, 125.9)</td>
<td>0.25</td>
<td>0.1 (&lt; 0.01, 0.5)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.85 310.6 (5.0, 1480.7) 5.89 38.61 (1.9, 121.9)</td>
<td>0.25</td>
<td>0.1 (&lt; 0.01, 0.5)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.83 1112.0 (21.8, &gt; 3600) 7.01 533.1 (3.7, &gt; 3600)</td>
<td>10.36</td>
<td>47.4 (0.27, 192.0)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>1.77 178.4 2.72 60.7 6.86 9.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹Average percentage gap from the optimal solution values of 10 instances.
²Average CPU seconds of 10 instances (minimum and maximum in parenthesis).

**Table 3. Performance of the priority-rule-based algorithm: Comparison of priority rules.**

<table>
<thead>
<tr>
<th>Number of operations</th>
<th>Number of targets</th>
<th>RAN</th>
<th>SC</th>
<th>SR</th>
<th>SC/R</th>
<th>LC/R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>N_best</td>
<td>RPR</td>
<td>N_best</td>
<td>RPR</td>
<td>N_best</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>0¹</td>
<td>31.47²</td>
<td>7</td>
<td>0.33</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>28.24</td>
<td>8</td>
<td>0.48</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>28.79</td>
<td>8</td>
<td>0.46</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>0</td>
<td>28.50</td>
<td>8</td>
<td>0.52</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>25.28</td>
<td>8</td>
<td>0.23</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>30.23</td>
<td>8</td>
<td>0.07</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>0</td>
<td>27.19</td>
<td>7</td>
<td>0.30</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>27.81</td>
<td>8</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>26.62</td>
<td>9</td>
<td>0.18</td>
<td>0</td>
</tr>
<tr>
<td>400</td>
<td>1</td>
<td>0</td>
<td>27.71</td>
<td>8</td>
<td>0.08</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>26.42</td>
<td>8</td>
<td>0.18</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>27.88</td>
<td>10</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>1</td>
<td>0</td>
<td>28.39</td>
<td>6</td>
<td>0.47</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>27.26</td>
<td>8</td>
<td>0.21</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>26.71</td>
<td>8</td>
<td>0.09</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>0</td>
<td>27.90</td>
<td>7.93</td>
<td>0.24</td>
<td>0</td>
</tr>
</tbody>
</table>

¹Number of instances out of 10 where the priority rule gave better solutions.
²Average relative performance ratio of 10 instances.
relative performance ratio is used because we could not obtain the optimal solutions. Here, the relative performance ratio for a test instance is defined as

$$100 \cdot \frac{(C_a - C_{best})}{C_{best}},$$

where $C_a$ is the objective function value obtained using rule $a$ for the instance and $C_{best}$ is the best objective function value among those obtained from the five priority rules.

Test results are summarised in Table 3 which shows the average relative performance ratios of the five priority rules. It can be seen from the table that the SC and the SC/R rules outperform the others in overall average. Of the two best rules, the SC rule, which selects an operation with the smallest disassembly cost, was slightly better than the SC/R rule. In particular, it was much better than the benchmark RAN rule.

Finally, to illustrate the mathematical model and the algorithms suggested in this study, we report a case study on an end-of-life electronic calculator. As can be seen in Table 4, which summarises the required data, there are 26 disassembly operations that correspond to parts or subassemblies. Also, Figure 5 shows the disassembly precedence graph in which node 0 represents the root node. The disassembly costs, which include the sequence dependent setup costs, were generated randomly from $DU(20, 100)$. After the target components were set to the two rubber bars and the button

<table>
<thead>
<tr>
<th>Disassembly operations</th>
<th>Components</th>
<th>Material</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3, 4</td>
<td>Screw cover</td>
<td>Sponge</td>
</tr>
<tr>
<td>5, 6, 7, 8, 12, 13, 15, 16, 17, 18</td>
<td>Screw</td>
<td>Metal</td>
</tr>
<tr>
<td>9</td>
<td>Back plate</td>
<td>Plastic</td>
</tr>
<tr>
<td>10, 11</td>
<td>Rubber bar*</td>
<td>Rubber</td>
</tr>
<tr>
<td>14</td>
<td>Front plate</td>
<td>Plastic</td>
</tr>
<tr>
<td>19, 20</td>
<td>Electric wire</td>
<td>Wire</td>
</tr>
<tr>
<td>21</td>
<td>PCB board</td>
<td>Plastic</td>
</tr>
<tr>
<td>23</td>
<td>Liquid crystal</td>
<td>Glass</td>
</tr>
<tr>
<td>24</td>
<td>Button plate*</td>
<td>Rubber</td>
</tr>
<tr>
<td>25</td>
<td>Buttons</td>
<td>Plastic</td>
</tr>
<tr>
<td>26</td>
<td>Liquid crystal cover</td>
<td>Plastic</td>
</tr>
</tbody>
</table>

*Target components.

Figure 5. Disassembly precedence graph for the electronic calculator.
(b) Priority-rule-based algorithms

<table>
<thead>
<tr>
<th></th>
<th>RAN</th>
<th>SC</th>
<th>SR</th>
<th>SC/R</th>
<th>LC/R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap</td>
<td>0.56</td>
<td>0.28</td>
<td>0.54</td>
<td>0.25</td>
<td>0.80</td>
</tr>
</tbody>
</table>

See footnotes from Table 2.

5. Concluding remarks

In this study, we considered selective disassembly sequencing under the sequential disassembly environment in which only one component is obtained at each disassembly operation. The problem is to determine the sequence of disassembly operations to obtain multiple target components of a used/end-of-life product for the purpose of repair, reuse, remanufacturing, disposal, etc. In particular, sequence-dependent setup costs were additionally considered. After representing the problem as the disassembly precedence graph, a new integer programming model was suggested for the objective of minimising the total disassembly cost. We proved that the problem is NP-hard and then suggested two types of heuristics, called the branch and fathoming algorithm (for small-to-medium-sized instances) and the priority rule-based algorithm (for large-sized instances). A series of computational experiments were done on the new integer programming model and the performances of the two types of heuristics and the results can be summarised as follows. First, the new integer programming model is much more effective than the existing model. Second, the branch and fathoming algorithm gives good quality solutions, depending on the parameter \( \alpha \) value. Finally, the SC and the SC/R rules in the priority-rule-based algorithm are better than the others. Also, a case study was reported on an end-of-life electronic calculator to show the applicability of the new mathematical model and the solution algorithms.

The limitations and further research topics can be summarised as follows. First, in the theoretical aspect, it is needed to develop optimal algorithms after characterising the optimal solution properties. Also, it is necessary to devise more effective algorithms such as meta-heuristics. Second, this study considers the case of sequential disassembly in which only one component can be obtained at a time. Since the sequential disassembly environment is a restricted one, it is needed to extend the problem in a more general parallel disassembly environment in which two or more components can be obtained at the same time. In this case, the disassembly precedence graph suggested in this study must be substantially modified, e.g., and/or graph. Finally, since we consider a deterministic version of the problem, various uncertainty factors, such as defective components and stochastic costs, are important further considerations.

Acknowledgements

This work was supported by a National Research Foundation of Korea grant funded by the Korean government (NRF) (grant code: 2011-0015888). This is gratefully acknowledged.

References


