IIE Transactions

ISSN: 0740-817X (Print) 1545-8830 (Online) Journal homepage: http://www.tandfonline.com/loi/uiie20

Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph

Ali Koc , Ihsan Sabuncuoglu & Erdal Erel

To cite this article: Ali Koc , Ihsan Sabuncuoglu & Erdal Erel (2009) Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph, IIE Transactions, 41:10, 866-881, DOI: 10.1080/07408170802510390

To link to this article: https://doi.org/10.1080/07408170802510390

Published online: 03 Aug 2009.

Submit your article to this journal

Article views: 377

Citing articles: 58 View citing articles

Full Terms & Conditions of access and use can be found at
http://www.tandfonline.com/action/journalInformation?journalCode=uiie21

IIE Transactions (2009) **41**, 866–881
Copyright © "IIE"
ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/07408170802510390

Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph

ALI KOC[1], IHSAN SABUNCUOGLU[2,*] and ERDAL EREL[3]

[1]*Graduate Program in Operations Research, The University of Texas at Austin, Austin, TX 78712-0292, USA*
E-mail: alikoc@mail.utexas.edu
[2]*Department of Industrial Engineering, and* [3]*Faculty of Business Administration, Bilkent University, 06533 Bilkent, Turkey*
E-mail: {sabun,erel}@bilkent.edu.tr

Received May 2006 and accepted August 2008

In this paper, the disassembly line balancing problem, which involves determining a line design in which used products are completely disassembled to obtain useable components in a cost-effective manner, is studied. Because of the growing demand for a cleaner environment, this problem has become an important issue in reverse manufacturing. In this study, two exact formulations are developed that utilize an AND/OR Graph (AOG) as the main input to ensure the feasibility of the precedence relations among the tasks. It is also shown that traditional task precedence diagrams can be derived from the AOG of a given product structure. This procedure leads to considerably better solutions of the traditional assembly line balancing problems; it may alter the approach taken by previous researchers in this area.

Keywords: Disassembly line balancing, assembly line balancing, task precedence diagram, AND/OR graph

1. Introduction

In the last two decades environmental issues have become a major concern for industry. Governments have passed legislation against the indiscriminate disposal of hazardous products. Companies have started to comply with these rulings by changing their infrastructure and production processes (Inman, 2002). As a result, reverse manufacturing has become a common practice and disassembly is one of the key processes in this approach. Disassembly is defined as a systematic method for separating a product into its constituent parts, subassemblies or other groupings (Gupta and Taleb, 1994). The disassembly can be partial or complete. If the quality of the remanufactured product is to be the same as that of the new product, it is completely disassembled; otherwise, partial disassembly is sufficient. Demanufacturing can also include selective disassembly, in which only the profitable and/or hazardous components of the product are disassembled. At the operational level, disassembly is also classified as being either destructive or non-destructive (Jovane *et al.*, 1993). Destructive disassembly is used if technological constraints such as irreversible connections or damaged parts make it either impossible or unprofitable to disassemble the product without destruction.

Although assembly and disassembly seem to be the reverse of each other, they differ in terms of operational and production planning. The differences in the operational aspect are due to the variety of the tasks and their differing durations. The times required for disassembly tasks are more variable than in assembly, because of both the irreversibility of some assembly operations and uncertainty in the quality of the cores. The differences in the production planning arise for two reasons. First, disassembly is a divergent process resulting in multiple items, in contrast to the convergent nature of the assembly process (Brennan *et al.*, 1994). Disassembly can be partial or complete whereas assembly is always complete. Second, type, quality, and quantity of the inputs of the disassembly process (cores) show higher uncertainty than the inputs of the assembly process (subassemblies) (Brennan *et al.*, 1994).

There are several research areas in the disassembly literature with sequencing and line balancing being major research streams. Several papers study disassembly

*Corresponding author

sequencing problems in which the disassembly process sequence is generated for a given product on a single station (Gungor and Gupta, 2001b; Lambert, 2003). In this paper, however, we study the DisAssembly Line Balancing (DALB) problem which is the reverse of the well known Assembly Line Balancing (ALB) problem. The DALB problem is the process of allocating a set of tasks to an ordered sequence of stations in such a way that some performance measures (e.g., cycle time, number of stations) are optimized by using the AND/OR Graph (AOG) of the product. The ALB problem has been extensively studied in the literature (see Becker and Scholl (2006) and Scholl and Becker (2006)), but the DALB literature is relatively sparse.

In this study, we develop two exact formulations (Dynamic Programming (DP) and Integer Programming (IP)) for the DALB problem. The proposed models utilize an AOG as the main input to ensure the feasibility of precedence relations among the tasks. We prove that using an AOG instead of a precedence diagram leads to better solutions of the traditional ALB problems. In this specific area, there are only two studies: Altekin *et al.* (2008) proposes an IP formulation for profit maximization using an AOG to solve the disassembly leveling and line balancing problems simultaneously. The second study from Gungor and Gupta (2001a) defines the DALB problem for a part precedence diagram. To cope with the uncertainties in the disassembly process, they propose a solution using the shortest-path formulation adopted from the ALB problem.

Our study differs from the previous studies in several ways. First, we develop an IP formulation to effectively solve the DALB problem. Second, we develop a DP formulation. Third, we prove that several Task Precedence Diagrams (TPDs) can be derived from the AOG. Fourth, the ALB problem solved using the AOG results in better designs than the traditional ALB problem solved using a TPD. Finally, we compare the proposed IP and DP formulations with respect to various problem parameters.

The rest of this paper is organized as follows. Section 2 contains both a formal definition of the DALB problem and the related literature. This is followed by the IP formulation of the problem in Section 3. We discuss the benefits of using an AOG in both assembly and disassembly processes in Section 4. A DP formulation of the problem is given in Section 5 with an illustrative example. In Section 6, we test the relative performance of the IP and DP formulations on a set of problems. Finally, we give conclusions and further research directions in Section 7.

2. Problem definition and the related literature

In this section, we define the DALB problem and give the main characteristics of the AOG along with the relevant literature.

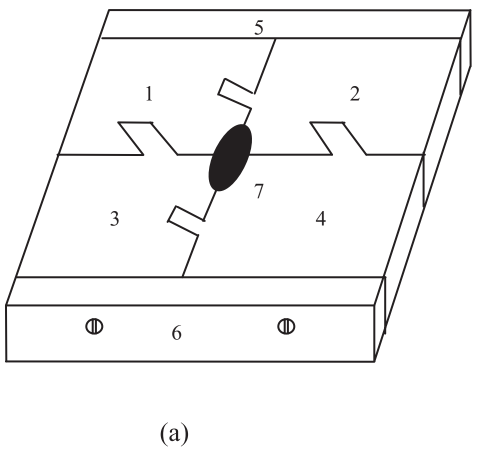
2.1. *The AOG*

An AOG is a graph that depicts all the possible ways of completely disassembling a product into its basic components. Nodes represent subassemblies and components and the arcs correspond to the disassembly tasks. In this representation, it is typically assumed that each disassembly takes apart the product or subassembly into exactly two new subassemblies (this assumption can easily be relaxed to three or more subassemblies in our methodology). Hence, a term, hyper-arc, is used to denote a link connecting two subassemblies of the disassembly task with the input node. Each hyper-arc is adjacent to one input node and adjacent to two output nodes. The node to which none of the hyper-arcs are adjacent is called the initial node, and the nodes from which none of the hyper-arcs are adjacent are called terminal nodes. (The reader may refer to De Mello and Sanderson (1990, 1991a, 1991b) for further discussions of these concepts).

In order to explain how to get an AOG from a product, an assumed product is constructed with seven components and their connections (or contacts) are displayed in Fig. 1. This example is created in such a way that two TPDs are generated from the AOG of the sample product. The Graph of Connections (GOC) in this figure presents the information regarding the contacts between the components. The AOG representation of this sample product is given in Fig. 2. Note that there are 13 nodes and 23 hyper-arcs (disassembly tasks) to completely disassemble the sample product. The numbers within the nodes represent the related subassembly. For example, in node 2 ($\underline{2}$) the component 6 is taken away from the product and the resulting subassembly consists of components 1 through 5 (i.e., 1/5) and component 7. For simplicity, subassemblies with only one component are not shown in the graph.

To give a formal definition of the AOG, let K be a set of elements and $\Pi\ (K)$ be the set of all subsets of K. Consider a product A with parts $P_A = \{p_1, p_2, \ldots, \mathrm{p}_N\}$, where N is the number of parts. There is a unique AOG of assembly/disassembly sequences for A defined as $< S_A, D_A >$ such that $S_A = \{\theta \in \Pi(\mathrm{P}_A) | sa(\theta) = \text{“}T\text{”}$ and $st(\theta) = \text{“}T\text{”}\}$ is the set of nodes (subassemblies) in the AOG and $D_A = \{(\theta_k, \{\theta_i, \theta_j\}) | [\theta_k, \theta_i, \theta_j \in S_A]$ and $[\tau(\theta_i, \theta_j) = \theta_k]$ and $gf(\tau) = \text{“}T\text{”}$ and $mf(\tau) = \text{“}T\text{”}\}$ is the set of hyper-arcs, T stands for ‘true,’ τ is the assembly task, sa and st are subassembly and stability predicates, and gf and mf are geometrical and mechanical feasibility predicates.

We use the predicate notation *sa*(.) to denote whether or not a subset of parts constitutes a subassembly. For instance, in Fig. 1 *sa*(component 1, component 2) = *true* means that {component 1, component 2} is a subassembly, whereas *sa*{component 5, component 6} = *false* means that {component 5, component 6} is not a subassembly. Note that the value of this predicate for any subset of components can be easily determined from the GOC. A subassembly is said to be *stable* if components maintain their



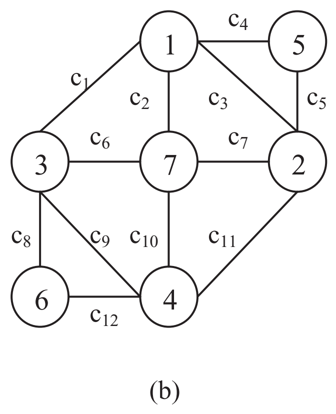


Fig. 1. A sample product: (a) overview; and (b) the graph of connections.

relative positions and do not break spontaneously. Hence, a subassembly should also satisfy the stability predicate *st*(.) that determines whether or not a subassembly described is stable. A subassembly is said to be feasible if both of the *sa* and *st* are true.

Assembly/disassembly tasks have three properties. First, given two input subassemblies ϕ and θ, we say that joining them is an assembly task if the output, $\{P_\phi, P_\theta\}$, is also a subassembly (i.e., *sa* (P_ϕ, P_θ) is true). This property can be reversed for the disassembly task. Second, an assembly task should be geometrically feasible, implying a collision-free path to joining the two subassemblies. We use the geometrical feasibility predicate *gf*(.) to denote whether or not an assembly task is geometrically feasible. Third, an assembly task should be mechanically feasible—it should be possible to establish the attachments between the two subassemblies. We use the mechanical feasibility predicate *mf*(.) to denote whether or not an assembly task is mechanically feasible. A task is said to be feasible if it is both geometrically and mechanically feasible.

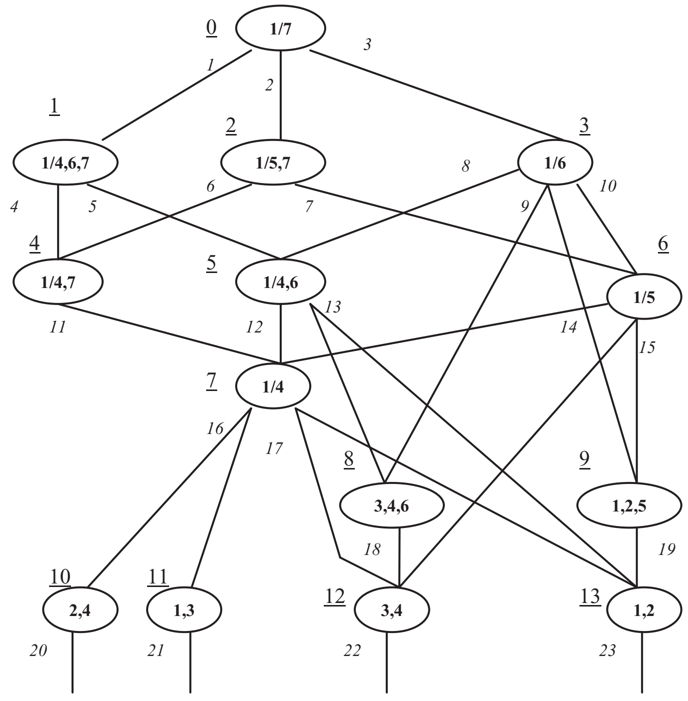


Fig. 2. AND/OR graph of the sample product.

We define an AND/OR path in the graph as a set of hyper-arcs with k (>0) elements and their corresponding input and output nodes such that there are no two hyper-arcs arising from the same node, and there is only one initial node in the path. An AND/OR path that has $k = N-1$ nodes is called a Disassembly Tree (DT) of the product. Note that, a DT should have $\{p_1, p_2, \ldots, p_N\}$ as its initial node and $\{p_1\}$, $\{p_2\}, \ldots, \{p_N\}$ as its terminal nodes. A DT with N–1 hyper-arcs (disassembly tasks), represents a way of completely disassembling the product. There are 17 unique DTs corresponding to the AOG of the sample product in Fig. 2; two of them are depicted in Fig. 3.

Note that only a subset of the tasks in the AOG is used to completely disassemble the product. The relation between TPDs and DTs will become clearer in Section 4.1. Broadly stated, one TPD may contain the precedence information of more than one DT. For instance, only two different TPDs (please refer to Fig. 6) can be generated for the example product in Fig. 1, whereas the same product has 17 different DTs. This compactness of TPDs comes at a price. Namely, we cannot differentiate the tasks of an AOG that disassemble the same contacts but from different subassemblies. Since we label the tasks that apply to different subassemblies as the same task, we take the maximum of task durations over all tasks that we unified. From these discussions it is clear that the studies in the ALB literature that employ TPDs to solve the problem may not lead to an optimum solution for the overall problem because of two reasons. First, there is more than one TPD for the product, i.e., one TPD does not contain all feasible

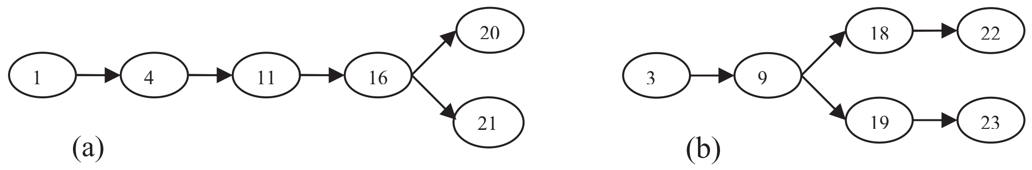


Fig. 3. Two representative DTs of the sample product: (a) DT_1; and (b) DT_{14}.

precedence relations; or second, as mentioned above, when we unify the tasks we take the maximum of the durations so that even if a single TPD would contain all feasible precedence relations, it will impose higher task times. One of our main contributions in this paper is to prove this important property (by the theorem of suboptimality in the following sections).

2.2. *The DALB using an AOG*

We now define the DALB using the concepts introduced in the previous section. Given: (i) a product with a list of parts (or components); (ii) the tasks required to completely disassemble the product; (iii) task times (including time to unfasten the fasteners); and (iv) the precedence relations among them, the problem is to allocate these tasks to a sequence of stations such that some performance measure is optimized. In our case, we minimize the number of stations for a given cycle time.

There are numerous ways of completely disassembling a product. We generate the DTs to consider all these possibilities. Note that the ALB or DALB problem with a given TPD can be solved using procedures (exact or heuristic) available in the literature. This has been the standard approach to solving the ALB problem for the last 50 years. However, this approach may lead to suboptimality if all possible DTs are not considered. Hence, the optimization problem defined in this paper ranges over all DTs, in contrast to the optimization problem defined in the literature for a given TPD.

A typical AOG shows all the possible subassemblies (not precedence relations among the disassembly tasks), whereas a DT contains information about the tasks needed to completely disassemble the product. In order to effectively formulate the DALB problem, we introduce another graph called a *Transformed AOG* (TAOG); this is a modified version of an AOG containing full information on all the DTs (i.e., an AOG with explicit information on the precedence relations between tasks). There are several graphs (DT and disassembly graphs) that represent assembly/disassembly-related information. The term DT (Johnson and Wang, 1995, 1998; Krikke *et al.*, 1998) is used to refer to a Bill of Materials (BOM) structure of the product. Hence, it does not have any direct relation to an AOG. A disassembly graph (Lambert, 1997, 2003) is the same as an AOG—another name for the same concept. A TAOG is an alternative representation of AOG that makes the IP formulation more efficient. Note that we also use the term disassembly to represent alternative disassembly sequences (not AOG or BOM).

A TAOG is formed as follows. Each node in the AOG corresponding to a subassembly is represented by an artificial node in the TAOG and each hyper-arc in the AOG is represented by a normal node in TAOG, while maintaining the precedence relations between the subassemblies (artificial nodes) and the tasks (normal nodes). We label the artificial nodes A_is and normal nodes B_is as shown in Fig. 4. An artificial node may be preceded or succeeded by more than one normal node. However, only one of the predecessors and one of the successors should be processed.

We identify two types of arcs in a TAOG: AND-type and OR-type. The AND-type arc imposes the regular precedence relation whereas the OR-type arc allows the selection of any of the follower arcs. To differentiate between the AND-type and OR-type relations, we put a small curve as an indicator of OR-type relations in Fig. 4. Using the OR-type arcs in an AOG allows us to consider only a subset of tasks to completely disassemble a product.

2.3. *Relevant literature*

The literature on the DALB problem is scarce and there are only two relevant studies. Altekin *et al.* (2008) consider a profit-oriented DALB problem. The authors define the problem for an AOG that tries to answer both the disassembly leveling and line balancing problems simultaneously. Merging tactical and operational level problems, the authors construct a huge IP problem that cannot be solved optimally even for very small products.

In the other study, Gungor and Gupta (2001a) define the DALB problem for a Part Precedence Diagram (PPD). They do not focus on methods of solving the problem; but focus rather on uncertainties in the disassembly lines using a solution procedure given in the literature.

Our study differs from the above studies in several respects. First, we focus on a complete disassembly process rather than partial disassembly, even though our proposed methodology applies also to partial disassembly. Second, we use a TAOG instead of the AOG or PPD. Third, we develop a more compact and efficient IP formulation for the DALB problem. Fourth, we also develop a DP formulation

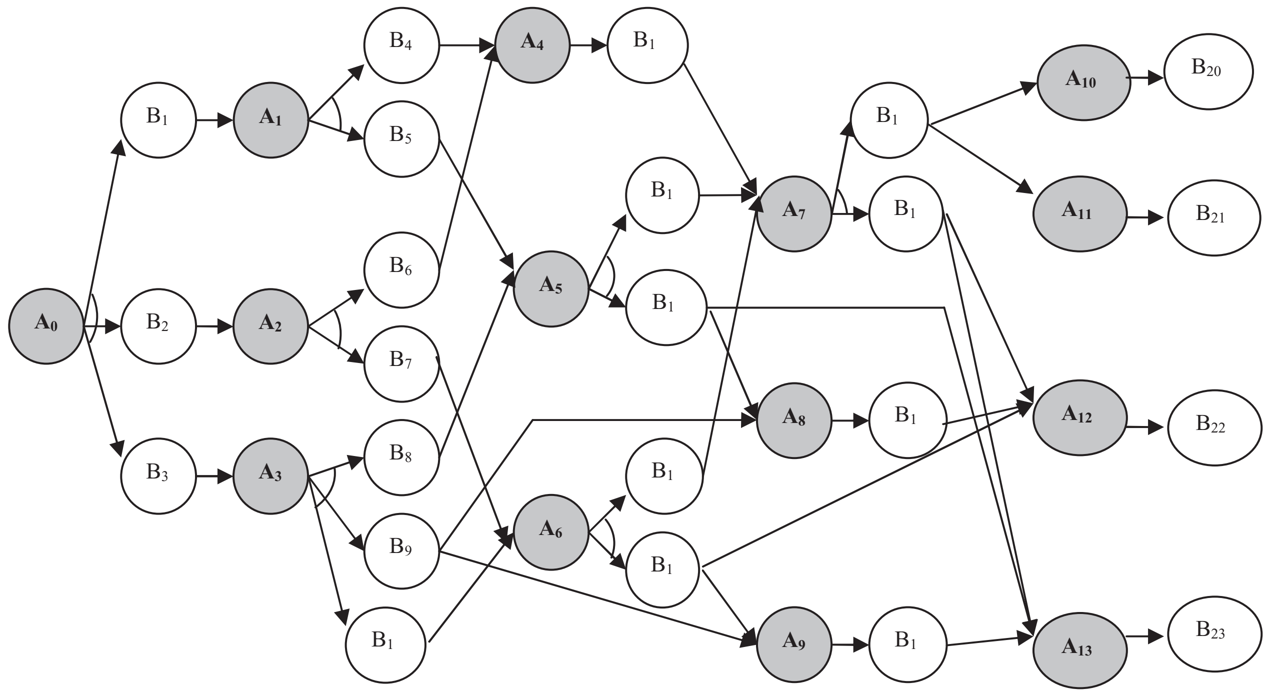


Fig. 4. The TAOG of the sample product.

for the same problem which turns out to be very efficient in solving large problems. Fifth, using the suboptimality theorem in Section 4, we show the derivation of well-known TPDs from AOG that link the analyses of the ALB and DALB problems.

From another perspective, our idea of using an AOG instead of a TPD aims to improve the information about the precedence relations and task times contained in the TPD. There are a few attempts in the ALB literature in the same vein, although without using an AOG. Specifically, Pinto *et al.* (1983) consider a TPD with tasks, each of which can be performed in a number of ways. Their only similarity to our study is due to the fact that there are alternative ways of doing tasks. However, they do not consider explicitly or implicitly an AOG. They solve the line balancing problem together with the equipment selection problem. Capacho and Pastor (2006) state that the solution of the line balancing problem with a given TPD can be improved if sequence-independent tasks can be changed into sequence-dependent tasks by incorporating appropriate OR-type relations. This statement is intuitive and is considered in similar studies, such as the disassembly sequencing problem. On the other hand, our idea of using an AOG is merely to differentiate between subassembly independent tasks as opposed to sequence-dependent tasks (Section 4). The authors do not propose any methodology to implement their idea. Similarly, Scholl *et al.* (2006) consider sequence-dependent tasks by relaxing some AND type relations in a given precedence diagram. They develop an IP formulation by using a TPD rather than an AOG.

3. IP formulation of the problem

Notation

A_k	=	artificial nodes in AOG $k = 0, 1, 2, \dots, h;$
B_i	=	normal nodes in AOG $i = 1, 2, \dots, l;$
d_{B_i}	=	task time (normal node) of B_i;
$P(A_k)$, $P(B_i)$	=	immediate predecessor set of A_k, B_i, respectively;
$S(A_k)$, $S(B_i)$	=	immediate successor set of artificial node A_k, B_i, respectively;
T	=	cycle time;
j	=	station index $j = 1, 2, \dots, M.$

Decision variables

$$x_{ij} = \begin{cases} 1 & \text{if task } B_i \text{ is assigned to station } j, \\ 0 & \text{otherwise.} \end{cases}$$

Auxiliary variables

$$f_j = \begin{cases} 1 & \text{if station } j \text{ is opened,} \\ 0 & \text{otherwise.} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if task } B_i \text{ is performed,} \\ 0 & \text{otherwise.} \end{cases}$$

The formulation of the problem can be given as

$$\min \sum_{j=1}^{M} j \times f_j$$

subject to

$$\sum_{i:B_i\in S(A_k)} z_i = 1 \qquad \text{for } k=0, \tag{1}$$

$$\sum_{i:B_i\in S(A_k)} z_i = \sum_{i:B_i\in P(A_k)} z_i \qquad \text{for } k=1,2,\ldots,h, \tag{2}$$

$$\sum_{j=1}^{M} x_{ij} = z_i \qquad \text{for } i=1,2,\ldots,l, \tag{3}$$

$$\sum_{i:B_i\in P(A_k)} \sum_{j=1}^{v} x_{ij} \geq \sum_{i:B_i\in S(A_k)} x_{iv} \qquad \text{for } k=1,2,\ldots,h, \quad v=1,2,\ldots,M, \tag{4}$$

$$\sum_{i=1}^{l} x_{ij} \times d_{B_i} \leq T \times f_j \qquad \text{for } j=1,2,\ldots,M, \tag{5}$$

$$x_{ij} \in \{0,1\}, \quad f_j \in \{0,1\}, \quad z_i \in \{0,1\}. \tag{6}$$

Constraints (1) and (2) ensure that exactly one of the OR-successors is selected. Hence, these two constraints force the solution to be a set of tasks that constitute a DT. Constraint (3) ensures that a selected task is assigned to one of the stations. This constraint resembles the *occurrence* constraints in the ALB literature. Constraint (4) defines the precedence relations. Since exactly one of the OR-predecessors and one of the OR-successors of an artificial node is selected, constraint (4) ensures that the selected successor is not assigned to a lower-indexed station than the one to which the selected predecessor is assigned. Constraint (5) forces the total workload of a station to be less than the cycle time, if that station is opened. Constraint (6) indicates the 0–1 integer variables.

Note that this formulation is a general case of the traditional ALB problem in the sense that when the auxiliary variable z_i and the constraints (2) and (3) are eliminated, and the constraints (4) and (5) are modified appropriately, the IP formulation of the ALB problem is obtained.

Our formulation differs from the previous model of Altekin *et al.* (2007) in the following respects. In their study the authors solve the disassembly planning problem of determining the disassembly level that maximizes profits. They consider an operational problem in which the decisions are made frequently as new products arrive to be disassembled or cost/profit parameters change. Hence, the authors use costs, profits and inventory-related parameters as well as task durations and an AOG. Their model essentially generates partial disassembly decisions. Since their model involves several decision variables, parameters and constraints, the resulting model can be optimally solved only for very restrictive problem sizes.

In this study, however, we consider a strategic design problem for highly standard product structures. We assume that the level of disassembly (full disassembly or partial disassembly) is already set for a family of products after economic and technical analyses. In other words, we consider a system that will be operational for the foreseeable future barring major changes in the environmental factors.

There are also modeling differences between these two studies. Cycle time is a decision variable in Altekin *et al.* (2008) whereas it is a fixed parameter in our study. We use a TAOG which enables us to incorporate a large number of precedence relations in a very compact form, whereas Altekin *et al.* (2008) uses the original AOG with its additional decision variables (our artificial nodes are represented by the dummy decision variables in their study). Their model can be cast to handle only the complete disassembly case by adjusting the cost/profit parameters appropriately, but it still has to consider all possible disassembly levels, which diminishes the effectiveness of their model in the complete disassembly case. Another distinction is that Altekin *et al.* (2008) works with both AOG and a PPD, but we use the TAOG. This enables us to develop a more efficient model for a specific problem with fewer variables and constraints. For that reason, in our computational experiments we manage to solve reasonably large-size problems. In summary, our study differs from Altekin *et al.* (2008) in terms of objective, scope and the modeling approach.

4. Theorem of suboptimality

Both the AOG and TPD include information about the task precedence relations that should be considered in the assembly/disassembly process. The tasks in the AOG are defined based on the contacts broken and the current subassembly, whereas the tasks in the TPD are defined only in terms of the contacts. This leads to some tasks being labeled differently in the AOG; however, in the TPD a task that breaks a specific contact is labeled uniquely, independent of the current subassembly. For instance, tasks τ_6 and τ_8 in Fig. 5 break the same contacts (c_4, c_5), but in the AOG, they are labeled as different tasks. Hence, we call the tasks in the AOG Subassembly-Dependent tasks (SD tasks) and the tasks in TPD Subassembly-Independent tasks (SI tasks).

Theorem of suboptimality. *The optimal solution to the DALB (ALB) problem for a given TPD of a product (z^*_{TPD}) constitutes an upper bound on the optimal solution to the DALB (ALB) problem for the AOG (z^*_{AOG}) of the same product.*

We can give the proof for the ALB problem. The same proof also holds for the DALB problem since it is the reverse of the ALB problem.

Proof. Since the main intuition of the theorem is discussed in length in Section 2.1, we now provide a simple argument to support it as follows.

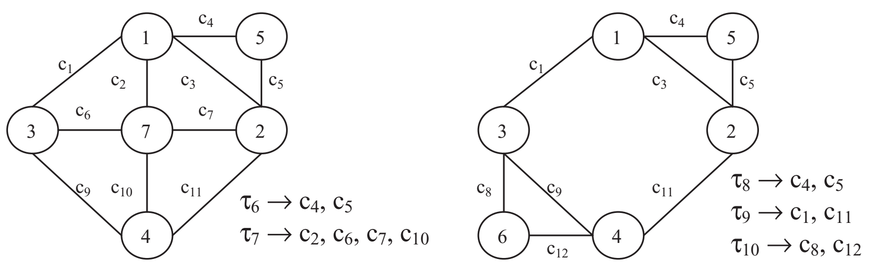


Fig. 5. Two subassemblies of the AOG in Fig. 2 and their GOC.

The ALB problem using a TPD (ALB-TPD) is actually a problem of sequencing the SI tasks, defined many decades ago (Held and Karp, 1962). Each SI task sequence generated from a given TPD corresponds to a specific design, one of which characterizes the optimal solution. An AOG contains all feasible ways to assemble/disassemble the product. Hence, any SI task sequence obtained from the TPD of a product has a corresponding SD task sequence in the AOG of the same product. Then, it only remains to see that the solution to the ALB problem characterized by an SI task sequence uses an equal or larger number of stations than the solution to the ALB problem characterized by the corresponding SD task sequence. However, when unifying a few SD tasks and labeling them as the same SI task we need to take the maximum of the durations of the SD tasks as the duration of the corresponding SI task (since we lose the information on which subassembly the relevant SI task applies). Hence, the duration of the SI task is equal or larger than the corresponding SD task, yielding a line design with equal or more number of stations. ■

4.1. *The derivation of a TPD from the AOG*

In this section we use an example to demonstrate the derivation of TPDs from the AOG of a product. The discussion here also clarifies the impact of the theorem of suboptimality.

Denote the set of DTs in the AOG as S_{DT}. Re-label the tasks in each DT such that the tasks that break the same contacts are labeled as the same tasks. Denote the resulting trees as DT^{m} and the set of DT^{m}s as $S_{\text{DT}^{\text{m}}}$. Note that DT^{m}s include SI task sequences. The two DT^{m}s that have exactly the same tasks are called equitask DT^{m}s. To get a TPD from $S_{\text{DT}^{\text{m}}}$, we first form groups of equitask DT^{m}s and then combine them into a single TPD for each group (let the set of TPDs be S_{TPD}). For instance, suppose that DT_1^{m} and DT_2^{m} consist of only two tasks, τ_1 and τ_2. Suppose further that, τ_1 is the predecessor of τ_2 in DT_1^{m}, and the successor of τ_2 in DT_2^{m}. Then, DT_1^{m} and DT_2^{m} are equitask DT^{m}s. We combine DT_1^{m} and DT_2^{m} so that the resulting TPD has two tasks, τ_1 and τ_2, without any precedence relation between each other. Note that if there are no equitask DT^{m}s, each DT^{m} corresponds to a TPD. For our sample product, by examining the contacts, re-labeling the tasks and combining the DT^{m}s, we obtain two TPDs as illustrated in Fig. 6.

The theorem of suboptimality and the derivation of TPDs from the AOG can also be seen in Fig. 7. We emphasize the fact that in the ALB studies, researchers usually solve the problem for a given TPD without actually knowing the product structure. Typically a process engineer provides this information (a specific TPD) to the analyst who solves the line balancing problem. However, this selected TPD may not be optimum considering all possible DTs that can be generated from the product structure. Hence, the resulting design proposed by the analyst (even though the problem is solved by an exact algorithm) may not be optimal for the actual problem (i.e., $z^*_{\text{TPD}} \geq z^*_{\text{AOG}}$). In contrast, in this study we use the full information about all possible TPDs for a given product structure (embedded in the AOG). Hence, the proposed exact methods in this study guarantee the optimality.

4.2. *Numerical examples to compare the TPD and AOG approaches*

To illustrate the discussion above, we now take the AOG and the two TPDs in Figs 2 and 6, respectively. Table 1 depicts task durations (for the SI tasks of TPD_1 and TPD_2) which are set as the maximum of the durations of the corresponding SD tasks. We solve the ALB-AOG and ALB-TPD problems for cycle time (T) values varied from 22 to 90. This range is determined between the maximum task duration (22) and 90 over which the ALB-AOG and ALB-TPD

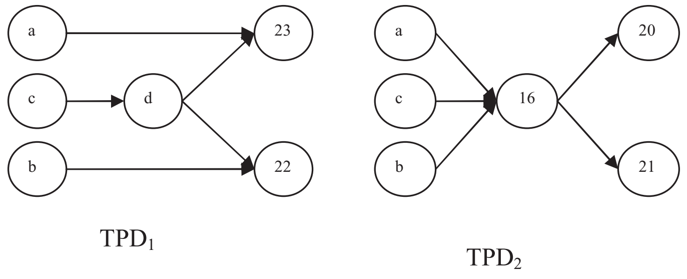


Fig. 6. S_{TPD} obtained by combining equitask DT^{m}s of the sample product.

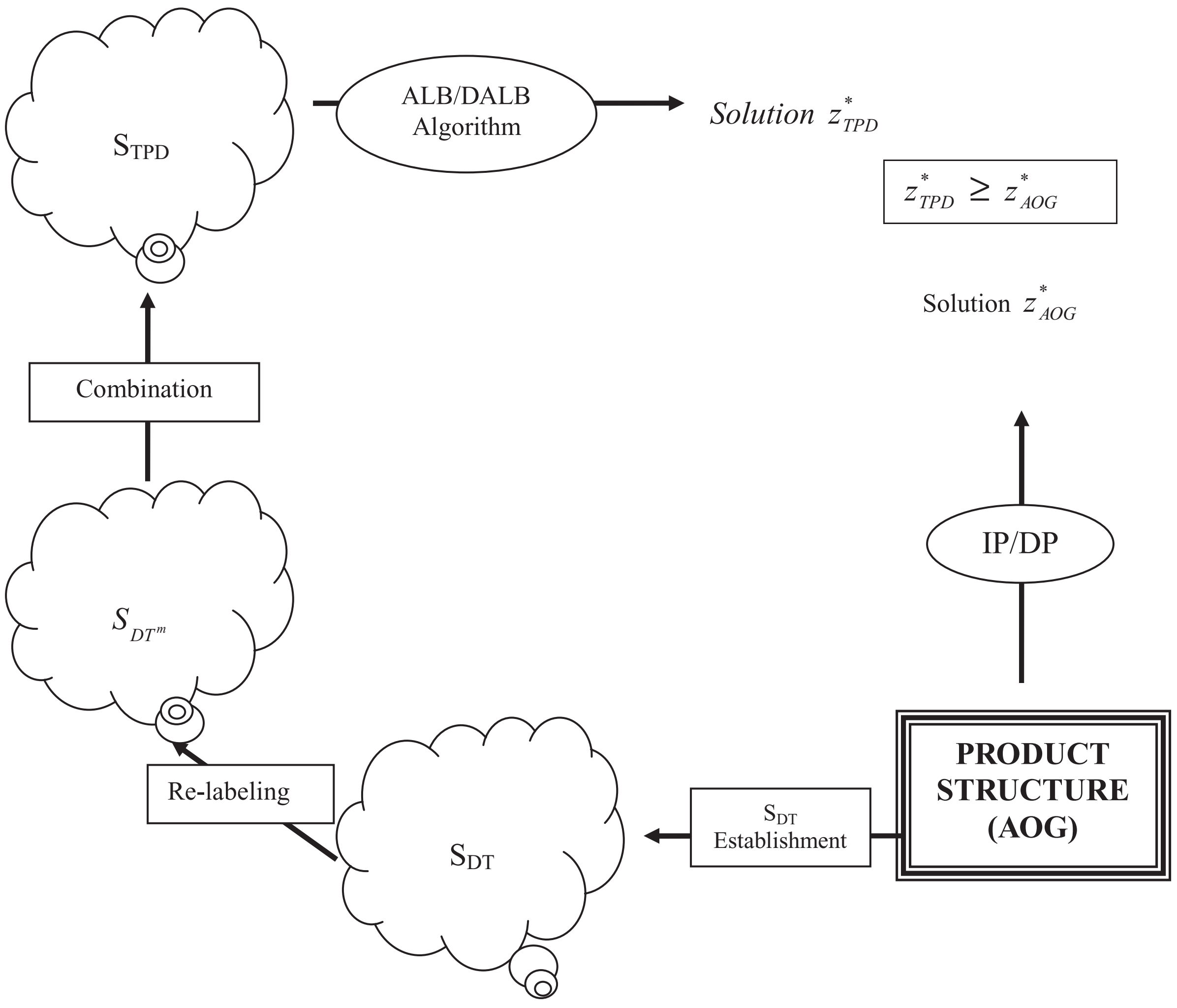


Fig. 7. A schematic view of the theorem of suboptimality.

problems yield the same designs. The solutions of the problems (numbers of stations) using the AOG, TPD_1 and TPD_2 for the ranged cycle time values are listed in Table 2. Note that the number of stations found by using the AOG are the optimal solutions to the problems.

The solutions of the 69 problems in Table 2 show that using TPD_1 and TPD_2 as the precedence diagram to the ALB-TPD problem results in optimal solutions for 28 and 25 problems, respectively. When both TPD_1 and TPD_2 are used, the number of problems with optimal solutions increases to 30. It is interesting to note that even if all TPDs are considered, the AOG still performs better in 56% of the instances. This is mainly due to the increase in the task times of the SI tasks when they are unified. This example clearly shows the superiority of using AOG instead of one or a set of TPDs.

Furthermore, we also compare the AOG and TPD on a real-life example from the literature (Lambert, 1997). This is a ballpoint pen disassembly example used by several researchers. The CAD drawing, GOC and AOG of the ballpoint pen are given in Figs 2 to 4 of Lambert's paper on pages 2514 and 2515. Following the analysis of the GOC, we obtain the SI tasks and their durations from the original SD tasks (see Table 3). These durations are proportional to the task costs of Lambert (1997) (fourth column of Table 1). Using the idea given in Section 4.1, we obtain the TPD in Fig. 8 from the AOG of the product (Fig. 4 in Lambert 1997). We then solve the ALB-AOG and ALB-TPD problems for cycle time (T) values varying from 80 to 525. This range is determined between the maximum task duration (80) and 525 over which the ALB-AOG and ALB-TPD problems yield the same number of stations. The solutions of the problems (i.e., number of stations) using the AOG and TPD for the ranged cycle time values are listed in Table 4. The results indicate that the AOG performs better than the TPD in 410 out of 445 cases (92.14% of the time the AOG is superior to the TPD). Even in some cases, the TPD leads to designs with 60% more stations.

Table 1. Task duration times of the SD and SI tasks

	AOG task																						
	1	*6*	*8*	*14*	*19*	*2*	*4*	*10*	*12*	*18*	*3*	*5*	*7*	*11*	*9*	*15*	*13*	*17*	*16*	*20*	*21*	*22*	*23*
SD task duration	22	21	21	20	18	22	21	21	20	18	14	13	13	12	16	15	15	14	14	7	7	7	7
SI task duration	22	22	22	22	22	22	22	22	22	22	14	14	14	14	16	16	16	16	14	7	7	7	7
TPD task	a	a	a	a	a	B	B	B	B	B	c	c	c	c	D	D	D	D	16	20	21	22	23

5. The proposed DP approach

5.1. *Partial AOG*

In the DP formulation, we use a partial AOG denoted as AOG ($\{A_i\}$). The partial AOG is a graph that includes a subset of DTs with one of their final nodes being A_i. With this partial AOG (or using a subset of DTs), we reduce the permutation-size solution space to combination size. The proposed DP approach is the extension of the well-known formulation of Held and Karp (1962) with the partial AOG.

AOG ($\{A_i\}$) is obtained in two steps: (i) delete all the normal and artificial nodes that precede node A_i; and (ii) delete all the other normal and artificial nodes that precede the normal nodes deleted in step 1 (i.e., we delete all the nodes that do not succeed node A_i, either directly or indirectly). This process results in the AOG ($\{A_i\}$) being a graph such that one of the final nodes of any generated DT is A_i. This is used in the DP recursion process.

Next, we extend the definition of a partial AOG as follows. Let *AOG* ($\{A_1, A_2, \ldots, A_k\}$) be a graph obtained in k steps: First, find AOG ($\{A_{i_1}\}$) from the AOG, then find AOG ($\{A_{i_1}, A_{i_2}\}$) from AOG ($\{A_{i_1}\}$), and then repeat this procedure until finding the AOG ($\{A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}}, A_{i_k}\}$) from the AOG ($\{A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}}\}$). Note that the sequence of artificial nodes is arbitrary in this k-step procedure. By convention, AOG ($\{\emptyset\}$) = AOG, and AOG ($\{A_0\}$) = $\emptyset$. In Fig. 9, some examples of partial AOGs are displayed for the sample product. The dashed lines and dashed nodes in the figure are merely intended to help the reader follow the construction of partial AOGs, but otherwise do not belong to the partial AOG.

Furthermore, let $S = \{A_1, A_2, \ldots, A_k\}$ be a set of artificial nodes. Final nodes of an AOG (S), denoted by $F(AOG(S))$, are defined as the set of normal nodes that do not precede any other normal node in AOG (S). For instance, the nodes B_{20}, B_{21}, B_{22}, B_{23} are the final nodes of the AOG in Fig. 4.

5.2. *Assembly task sequences*

We define assembly task sequences $\sigma = (B_1, B_2, \ldots, B_t)$ that are obtained from the normal nodes (tasks) of the AOG. This sequence is feasible if:

(i) $P(B_i) \neq P(B_j) \quad$ for $i \neq j$
(ii) $|\{B_1, B_2, \ldots, B_{i-1}\} \cap P(P(B_i))| = 1 \quad$ for $i = 2, 3, \ldots, t$,

where $P(B_i)$ is the artificial predecessor of the normal node B_i, $P(P(B_i))$ is the normal predecessor of the artificial node $P(B_i)$ and $|\,.\,|$ is the cardinality operator defined on the sets.

The first condition above prevents a sequence with an artificial node from having two OR-successors simultaneously. For instance, the sequence (B_1, B_4, B_5) in Fig. 4 is prohibited by this condition. Without the second condition, two normal nodes that are not OR-successors of the same artificial node but belong to the different DTs may exist in a given sequence. Note that the sequence (B_1, B_4, B_{11}, B_{13}) violates this condition. The second condition also guarantees that the normal nodes follow the precedence relations. For example, the sequence (B_4, B_{11}, B_{16}, B_{20}, B_1, B_{21}) is not allowed because the nodes (tasks) are not in the correct order, although they belong to the same DT.

Final nodes of a sequence, denoted by $F(\sigma)$, are defined to be the nodes of the sequence such that the sequence still remains feasible when they are removed from the sequence. The tasks B_{20} and B_{21} in Fig. 4 are the final nodes of the sequence (B_1, B_4, B_{11}, B_{16}, B_{20}, B_{21}). Associated with each feasible sequence σ is a particular assignment of tasks to the stations, (called the induced assignment for σ). This is obtained as follows. Assign as many tasks as possible from the beginning of the sequence to the first station, as many as

Table 2. Number of stations in the solution using AOG and TPDs for different cycle time (T) values

	Number of stations				*Number of stations*				*Number of stations*		
T	*AOG*	*TPD₁*	*TPD₂*	*T*	*AOG*	*TPD₁*	*TPD₂*	*T*	*AOG*	*TPD₁*	*TPD₂*
22–27	4	5	5	30–34	3	3	4	44–63	2	2	2
28	3	5	4	35	2	3	4	64–85	1	2	2
29	3	4	4	36–43	2	3	3	86–87	1	2	1
								88–90	1	1	1

Table 3. Task duration times of the SD and SI tasks

	AOG task																			
	a	*c*	*B*	*j*	*s*	*d*	*g*	*L*	*Q*	*e*	*m*	*p*	*F*	*h*	*I*	*k*	*o*	*N*	*r*	*t*
SD task duration	10	20	15	55	55	25	40	65	45	30	70	40	35	45	50	60	80	75	50	60
SI task duration	20	20	55	55	55	65	65	65	45	70	70	70	50	50	50	80		75	50	60
TPD task	1	1	2	2	2	3	3	3	3	4	4	4	5	5	5	6		7	8	9

possible from the beginning of the remaining subsequence to the second station, and so on, while not violating the cycle time (T) constraint. Intuitively, the induced assignment for a sequence is the optimal assignment. If the induced assignment for σ requires r stations and $w^{(r)}$ is the sum of durations of the tasks assigned to the last station, the quantity $c_\sigma = r - 1 + w^{(r)}/T$ is a measure of the "cost" of executing σ.

If a feasible sequence σ^* is formed by appending a task B_{t+1} to the end of σ, then:

$$c_{\sigma*} = c_\sigma + \Gamma(c_\sigma, d_{B_{t+1}}),$$

where

$$\Gamma(x, y) = \begin{cases} \lfloor x + y/T \rfloor - x + y/T & \text{if } \lfloor x \rfloor < \lfloor x + y/T \rfloor \\ \qquad < x + y/T, & \\ y/T \quad \text{if } \lfloor x + y/T \rfloor = \lfloor x \rfloor \textit{ or } \lfloor x + y/T \rfloor & \\ \qquad = x + y/T, & \end{cases} \tag{7}$$

where $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x.

The above equation can be interpreted as follows; if the unused idle time in the last station of the induced assignment σ is greater than or equal to $d_{B_{t+1}}$, then $\Gamma = d_{B_{t+1}}/T$; otherwise, a new station is opened. Hence, the unused idle time is added to $d_{B_{t+1}}/T$ in the computation of Γ.

We should further point out the fact that there is a natural correspondence between partial AOGs and feasible sequences defined by the below two mappings:

$$G(\sigma) = \{AOG(S) | F(\sigma) = F(AOG(S))\},$$
$$G^{-1}(AOG(S)) = \{\sigma | G(\sigma) = AOG(S)\}.$$

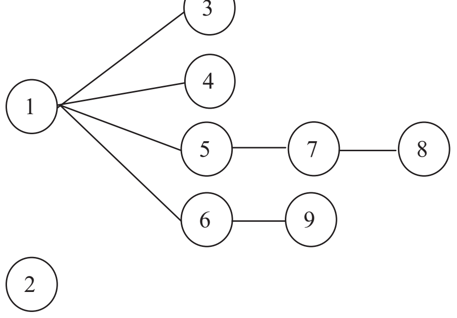


Fig. 8. TPD for the product in Lambert (1997).

Note that G is a one-to-many mapping. In other words, for an AOG(S) the number of feasible sequences is greater than or equal to one, while for each feasible sequence there is only one AOG(S). We define the following cost of each partial AOG(S) as the cost of the sequence that has the minimum cost over all the sequences to be obtained from that partial graph:

$$C(AOG\ (S)) = \min_{\sigma \in G^{-1}(AOG(S))} c_\sigma. \tag{8}$$

5.3. *The proposed DP approach*

From the discussions in the previous sections, it follows that solving the DALB-AOG problem is equivalent to finding $C(AOG\ (\emptyset))$. Furthermore, the minimum number of stations required for the disassembly line to perform the complete disassembly of the product is $\lceil C(AOG(\emptyset) \rceil$, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

Let B_i be the last task of the final node of $AOG(\emptyset)$. The optimum solution of the problem is obtained by choosing the DT with the minimum cost over all the sequences in which the last task is B_i. A graph is used to generate the rest of the solution. This graph called $AOG(\emptyset \cup \mathrm{P}(\mathrm{B}_i))$ is developed in such a way that all its tasks belong to the same DT of the $AOG(\emptyset)$ with B_i. This procedure is repeated until $AOG\{A_0\}$ is obtained. $C(AOG(S))$ can be calculated by the following recursion using Equations (7) and (8):

$$C(AOG\ (S)) \tag{9}$$
$$= \begin{cases} 0 & \text{for } S = \{A_0\} \\ \min\limits_{B_i \in F(AOG(S))} \{C(AOG(S \cup P(B_i))) + \Gamma(C(AOG(S \cup & \\ \qquad P(B_i))), d_{B_i})\} & \text{for } S \neq \{A_0\}. \end{cases}$$

Table 4. Number of stations in the solution using AOG and TPDs for different cycle time (T) values

	Number of stations			*Number of stations*			*Number of stations*	
T	*AOG*	*TPD*	*T*	*AOG*	*TPD*	*T*	*AOG*	*TPD*
80–89	7	8	115–119	4	6	215–264	2	3
90–99	6	8	120–134	4	5	265–414	2	2
100–104	5	8	135–144	4	4	415–524	1	2
105–109	5	7	145–179	3	4	>525	1	1
110–114	5	6	180–214	3	3			

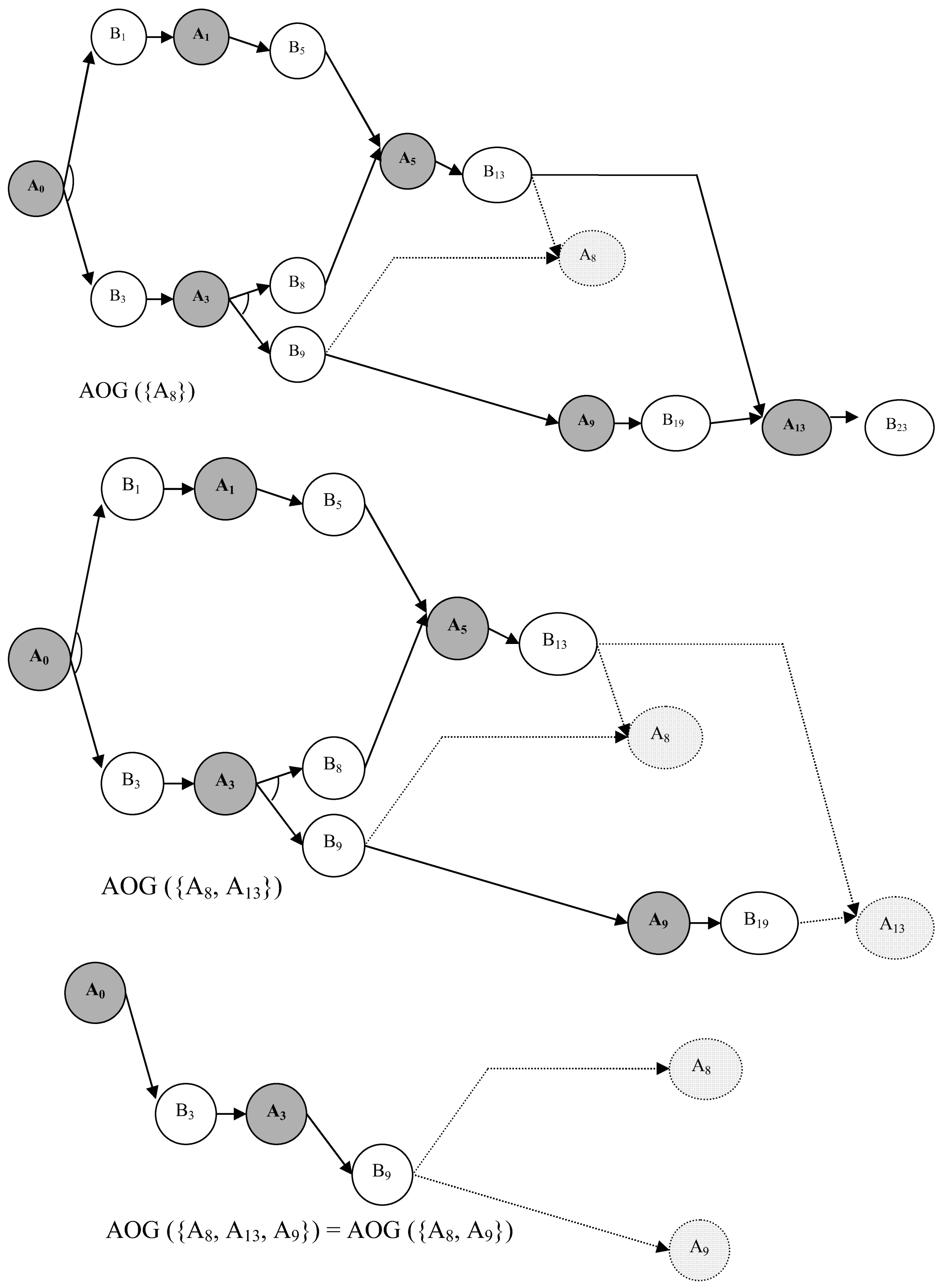


Fig. 9. Partial AOGs obtained from the AOG of the sample product in Fig. 4.

The recurrence relation in Equation (9) implies that if the solution of the ALB-AOG problem for an $AOG(S)$ yields a sequence of t tasks, the solution of the $AOG(S \cup P(B_i))$ yields a sequence of t–1 tasks, where $B_i \in F(AOG(S))$. In other words, the kth stage of the DP formulation is the set of $AOG(S)$s that are solved to generate the sequences with $(N-1-k)$ tasks, where N is the number of components in the product. Hence, there are a total of N stages, together with the stage 0, in the solution process of the problem. Stage 0 has only one state, which is $AOG(\emptyset)$. Similarly, the final stage (stage N–1) has only one state, which is $AOG(\{A_0\})$. The number of states in the other stages depends on both the number of components and the geometry of the product.

6. Computational results

In this section, we compare the two exact methods (IP and DP) by solving several instances of the DALB-AOG problem. Unlike the ALB problem, there are no test problems in the literature for the DALB problem that can be used for benchmarking purposes. Hence, we have to generate the problem sets for DALB.

6.1. *Benchmark problem generation scheme*

There are two types of tasks in the AOG: sequential and parallel tasks. Sequential tasks disassemble only one part from the subassembly whereas parallel tasks take apart a subassembly into two, each of which has at least two parts. Hence, a sequential task precedes at most one task whereas a parallel task precedes exactly two tasks. For example, the nodes B_9, B_{13}, B_{15}, B_{16} and B_{17} in Fig. 4 are the parallel tasks while the others are the sequential tasks. Parallel tasks are common in AOGs and this makes the DALB-AOG problem more difficult to solve due to the larger number of AOG alternatives. On the other hand, having only sequential tasks in the AOG makes the problem more manageable to handle. In this study, we use only sequential tasks to keep the computational times as low as possible.

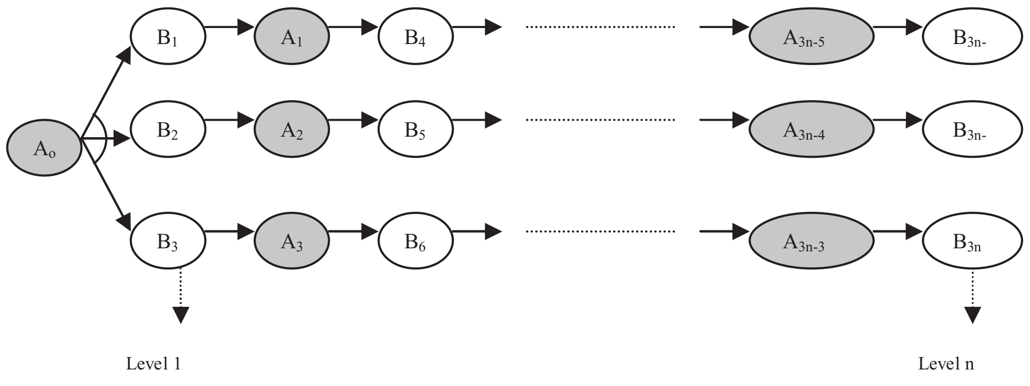
Another factor that affects solvability (or difficulty level) of the DALB problem is the size of the AOG. Recall that the size of AOG depends on both the geometry and number of components of the product. The number of components defines the number of tasks (normal nodes) in the solution of the problem. Since the main assumption of the AOG is that each task disassembles a subassembly into exactly two subassemblies or components, the number of tasks in the solution is one less than the number of components (or parts) in the product. For example, all of the AOGs in Fig. 10 belong to the products having $N+1$ components.

Even though the number of parts can be easily incorporated as a problem parameter, the part geometry is difficult to quantify to determine the size of the AOG. In this study, we define two parameters: the number of artificial nodes at each level in the AOG, denoted by a, and the number of tasks (normal nodes) for each artificial node, denoted by t, where, to make the terminology understandable, the *level* of an artificial node is defined as the number of artificial nodes preceding it (Fig. 10). The parameter a can vary from one level to another. The parameter t can vary even within a single level as well as between levels. In our experiments, we take them to be equal to each other both at each level and in each single level in order to standardize the AOGs so that researchers can easily compare their findings in future studies. The only exception is that the parameter t does not define the number of normal nodes for the first and the last a artificial nodes (i.e., A_0, $A_{an-(a-1)}$, $A_{an-(a-2)}$, …, A_{an-1}, A_{an}). In Figs 10(a) and (c), the parameter $a = 3$, whereas in Fig. 10(b), $a = 5$. In Figs 10(a) and (b), the parameter $t = 1$, whereas in Fig. 10(c), $t = 2$. When $t = 1$, the generation of AOGs is straightforward (see Figs 10(a) and (b)). However, when $t > 1$, say x, we generate AOGs as follows: assign the successors of the artificial nodes at level y to the artificial nodes at level $y+1$ one by one. That is, the first successor of the first artificial node at level y precedes the first artificial node at level $y+1$, the second successor precedes the second artificial node, etc. After the first artificial node at level y is connected to the artificial node at $y+1$, assign the first successor of the second artificial node to the $(x+1)$st artificial node, the second successor to the $(x+2)$nd artificial node, and so on. Whenever no unassigned artificial nodes remain at level $y+1$, we again start from the first artificial node. For example, as displayed in Fig. 10(c), $t = 2$. The first successor (B_4) of the first artificial node (A_1) at level 1 precedes the first artificial node (A_4) at level 2; the second successor (B_5) precedes the second artificial node (A_5); the first successor (B_6) of the second artificial node (A_2) precedes the third artificial node (A_6). Then, the artificial nodes at level 2 are finished. Hence, the second successor (B_7) precedes the first artificial node (A_4), and so on.

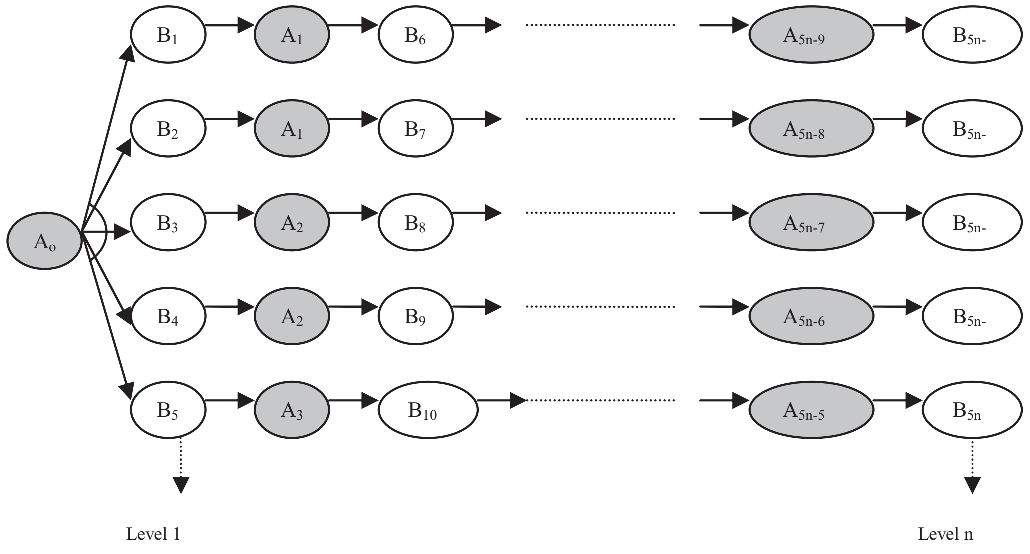
Using these three parameters (i.e., a, t and N), the total number of artificial nodes in the AOG of an N-part product, together with the node A_0, becomes $a \times (N-2) + 1$. The total number of normal nodes is $a \times [t \times (N-3) + 2]$. In our computational experiments we conclude that the problem is solvable if it can be solved without exhausting the memory and within 10 minutes of CPU time on a Pentium 4 Processor with 2.66 GHz using 512 MB RAM.

6.2. *The DP solution*

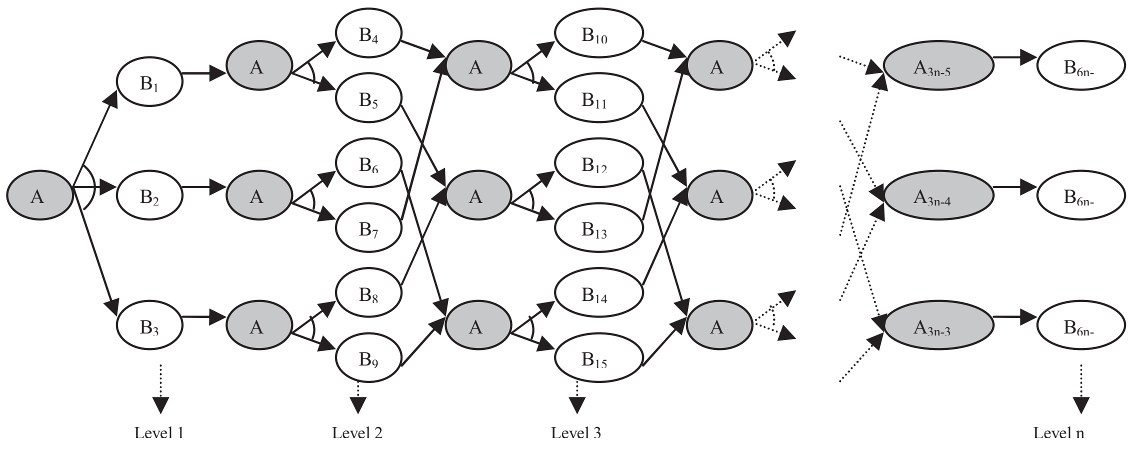
First, we discuss the use of the parameters N, a and t in the DP formulation. Note that the number of states in each stage is equal to a. This is because the (normal) nodes that are successors of the same artificial node yield the same partial AOG in stage $k+1$ when selected among the final nodes in stage k. However, the number of states is one for the first and the last stage. Parameter t determines the number of connections between the states of the successive



(a)



(b)



(c)

Fig. 10. Sample AOGs to illustrate the experiment: (a) $a = 3$, $t = 1$; (b) $a = 5$, $t = 1$; and (c) $a = 3$, $t = 2$.

Table 5. Solvable size of the AOGs without parallelism by the DP approach

Number of artificial nodes at each level (a)	*Number of tasks for each artificial node (t)*	*Maximum solvable number of parts (N)*	*Total number of tasks (normal nodes in AOG)*	*Stopping criteria*
3	1	249	744	OM
	2	98	576	OM
	3	87	762	TR
	5	67	966	TR
	10	38	1056	TR
4	1	225	896	OM
	2	74	576	OM
	3	64	740	TR
	5	48	908	TR
	10	27	968	TR
5	1	206	1025	OM
	2	60	580	OM
	3	53	760	OM
	5	35	810	TR
	10	21	910	TR
10	1	159	1580	OM
	2	32	600	OM
	3	24	650	TR
	5	17	720	TR
	10	12	920	TR

stages. That is, each state in stage k is connected with t states in stage $k+1$. The above discussion implies that if a unit of memory space is assigned to each state, a total of $(N-2) \times a + 2$ memory spaces are required. The computational requirements consist of $a \times [(N-3) \times t + 2]$ operations. It is $a + (N-3) \times t + 1$ for the second phase to find the optimal path. This is valid under the assumption that a and t are independent of N. However, this does not actually hold true in practice, leading to an excessive number of computational requirements (non-polynomial) of the DP. In our case, we assume independence to keep the computational times manageable.

In practice, since it is always desirable to solve large-size products, we first investigate the solvable sizes of the problem (N) for given values of a and t. As depicted in Table 5, the parameter $t = 1$, 2, 3, 5 and 10. The parameter $a =$ 3, 4, 5 and 10. For each combination of these two parameters, we run the DP algorithm as many times as possible until the problem instance cannot be solved to optimality. The stopping criteria used in the experiments are given in the right-hand column in the table; OM stands for out-of-memory and TR denotes the failure of time requirements. The following observations are made in the experiments.

1. When a increases, the solvable size of the problem decreases due to the fact that as a increases, the number of states increases. This also implies that when the number of feasible subassemblies increases, i.e., when the product geometry is highly compact, it takes more time to solve the problem.
2. When t increases, the solvable size of the problem decreases due to the increased number of computations.
3. All the parameters (a, t and N) have linear (polynomial) effects on the DP solution except for when $t = 1$.
4. The proposed DP approach can solve problems with N ranging from 30 to 60. This is a remarkable performance considering the "curse of dimensionality" property of DP approaches.

The above results are valid for any cycle time and task duration values, since time complexity of the proposed DP approach does not depend on these parameters.

6.3. *The IP approach*

We used CPLEX 7.5 to solve the IP model. CPLEX employs some fathoming techniques to expedite the solution process. Since the fathoming process is highly dependent on task durations and cycle time, the solvable size of the problem by CPLEX is affected by the data set. This means that, unlike the DP approach, for a given AOG, the problem solution time changes as a function of task durations and cycle time. Hence, in the IP case we have to generate a number of problem instances (minimum five independent replications) for a given AOG to assess the performance. Again, a 600 seconds CPU time is taken as the stopping criterion. Table 6 shows the results.

The first observation from Table 6 is that the IP formulation is not as successful as the DP formulation. The main reason that the solution time increases exponentially is due

Table 6. Solvable problem sizes with the IP approach

Number of artificial nodes at each level (a)	*Number of tasks for each artificial node (t)*	*Maximum solvable number of parts*
3	1	30
	2	30
	3	30
	5	30
	10	30
4	1	25
	2	25
	3	25
	5	25
	10	25
5	1	25
	2	20
	3	20
	5	20
	10	20
10	1	25
	2	20
	3	20
	5	20
	10	20

to the increase in the number of constraints of the IP formulation. This is true even if the number of variables in the IP formulation is a polynomial function of N, in the order of $O(N^2)$. In contrast, however, the number of stages and the computation time in the DP formulation increases in the order of $O(N)$.

Unlike the DP case, t and a do not affect the efficiency of the IP formulation as much as the parameter N. Our analysis indicates that the robustness of CPU time in these two parameters is due to two reasons. First, N increases the number of variables by $O(N^2)$, but a and t increase the number of variables by $O(a)$ and $O(t)$. Second, while N affects the constraints given in Equations (3), (4) and (5), the parameter t does not affect any constraint, and parameter a only affects the constraints in Equations (2) and (4).

7. Conclusion and future research

In this paper, we study the DALB problem and develop IP and DP-based formulations. We also prove the theorem of suboptimality by which we show that the well-known TPDs in the line balancing literature can be obtained from an AOG and the resulting solution using AOG is (potentially) better than the traditional ALB solution using a TPD in the literature. Our extensive computational experiments indicate that the DP approach performs better than the IP approach in terms of the solvable sizes of the problem.

We propose five further research directions. First, one of the main assumptions in the DALB literature is that each disassembly task generates two subassemblies. Even though this assumption reduces the size of the AOG, it increases the effort needed to solve the DALB problem. This assumption can be relaxed to represent disassembly processes in a realistic way. Second, the OR-type of precedence relations can be considered in the derivation of TPDs from the AOG. This is a non-trivial extension that may require some significant work. In our study we used AOGs having only sequential tasks and no parallel tasks. To generalize from the conclusions, it would be important to consider AOGs with parallel tasks. Hence, as the third research direction, it might be interesting to test the DP and IP formulations on general AOGs. Fourth, in the disassembly process, task times when parts are in a poor state may be longer than for standard parts. This aspect should be explicitly modeled with stochastic task times as a separate study. Finally, it is important to implement the proposed methodologies in practice to assess the actual performance of the models.

References

Altekin, F.T., Kandiller, L. and Ozdemirel, N.E. (2008) Profit-oriented disassembly-line balancing. *International Journal of Production Research*, **46**(10), 2675–2693.

Becker, C. and Scholl, A. (2006) A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, **168**, 694–715.

Brennan, L., Gupta. S.M. and Taleb, K.N. (1994) Operations planning issues in assembly/disassembly environment. *International Journal of Operations & Production Management*, **14**(9), 57–67.

Capacho, L. and Pastor, R. (2006) The ASALB problem with processing alternatives involving different tasks: definition, formalization and resolution. *In Proceedings of the International Conference on Computational Science and Its Applications* (3) 2006, pp. 554–563.

De Mello, L.S.H. and Sanderson. A.C. (1990) And/or graph representation of assembly plans. *IEEE Transactions on Robotics and Automation*, **6**(2), 188–199.

De Mello, L.S.H. and Sanderson, A.C. (1991a) A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, **6**(2), 228–240.

De Mello, L.S.H. and Sanderson, A.C. (1991b) Representation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, **7**(2), 211–227.

Gungor, A. and Gupta, S.M. (2001a) A solution approach to disassembly line balancing problem in the presence of task failures. *International Journal of Production Research*, **39**(7), 1427–1467.

Gungor, A. and Gupta, S.M. (2001b), Disassembly sequence plan generation using a branch-and-bound algorithm. *International Journal of Production Research*, **39**(3), 481–509.

Gupta, S.M. and Taleb, K.N. (1994) Scheduling disassembly. *International Journal of Production Research*, **8**, 1857–1866.

Held, M. and Karp, R.M. (1962) A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, **10**(1), 196–210.

Inman, R.A. (2002) Implications of environmental management for operations management, *Production Planning & Control*, **13**(1), 47–55.

Johnson, M.R. and Wang, M.H. (1995) Planning product disassembly for material recovery opportunities. *International Journal of Production Research*, **33**(11), 3119–3142.

Johnson, M.R. and Wang, M.H. (1998) Economical evaluation of disassembly operations for recycling, remanufacturing and reuse. *International Journal of Production Research*, **36**(12), 3227–3252.

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

Krikke, H.R., van Harten, A. and Schuur, P.C. (1998) On a medium term product recovery and disposal strategy for durable assembly products. *International Journal of Production Research*, **36**(1), 111–139.

Lambert, A.J.D. (1997) Optimal disassembly of complex products. *International Journal of Production Research*, **35**, 2509–2523.

Lambert, A.J.D. (2003) Disassembly sequencing: a survey. *International Journal of Production Research*, **41**(6), 3721–3759.

Pinto, P.A., Dannenbring, D.G. and Khumawala, B.M. (1983) Assembly line balancing with processing alternatives: An application. *Management Science*, **29**, 817–830.

Scholl, A. and Becker, C. (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, **168**, 666–693.

Scholl, A., Boysen, N. and Fliedner, M. (2006) The sequence-dependent assembly line balancing problem. *Working Paper*, FSU Jena, Germany.

Biographies

Ali Koc is a Ph.D. student in the Operations Research and Industrial Engineering program at the University of Texas at Austin. He received his B.S. and M.S. degrees from Bilkent University. His research interests are in the areas of stochastic modeling and programming, algorithms for high-performance computing, capital budgeting and project scheduling, and production planning and control. He is a member of INFORMS.

Ihsan Sabuncuoglu is the Chair of Industrial Engineering at Bilkent University. He received B.S. and M.S. degrees in Industrial Engineering from Middle East Technical University and a Ph.D. degree in Industrial Engineering from Wichita State University. He teaches and conducts research in the areas of simulation, scheduling and manufacturing systems. He has published papers in the *International Journal of Production Research*, *International Journal of Flexible Manufacturing Systems*, *International Journal of Computer Integrated Manufacturing*, *Computers and Operations Research*, *European Journal of Operational Research*, *Journal of Operational Research Society* and *Computers and Industrial Engineering*. He is on the Editorial Board of *International Journal of Operations and Quantitative Management*. He is an associate member of the Institute of Industrial Engineering and Institute for Operations Research and the Management Sciences.

Erdal Erel is the Dean of the Faculty of Business Administration at Bilkent University, Ankara, Turkey. He received his B.S. in Industrial Engineering from the Technical University of Istanbul, Turkey and M.S. from Stanford University. He received his Ph.D. in Industrial Engineering and Operations Research from Virginia Polytechnic Institute and State University. His research interests are in the areas of manufacturing systems analysis, and production planning and control. He has published papers in the *International Journal of Production Research*, *International Journal of Production Economics*, *Omega*, *European Journal of Operational Research*, *Journal of Operational Research Society*, *Applied Mathematics and Computation*, and *Annals of Operations Research*. He is a member of INFORMS.