INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED MANUFACTURING
https://doi.org/10.1080/0951192X.2019.1690680

Taylor & Francis
Taylor & Francis Group

ARTICLE

Automatic generation of assembly hierarchies for products with complex liaison relations

Zhengqian Jiang and Hui Wang

Department of Industrial & Manufacturing Engineering, Florida A&M University-Florida State University College of Engineering, Tallahassee, FL, USA

ABSTRACT

The assembly hierarchy for a product design determines the subassembly module formation, assembly tasks for the modules and serial-parallel material flow among these tasks. Automatic generation of the candidate assembly hierarchies by computer algorithms is a critical step to exploring potential design space for the assembly system design, and existing research on assembly sequence generation and subassembly identification has limitations in dealing with this challenge. This paper proposes to use assembly hierarchy instead of assembly sequence to generate the design space for assembly system design and optimisation. Based on liaison graphs, this paper first characterises the assembly hierarchy by developing a unique representation model to capture the hierarchical relationship among the assembly operations. A recursive algorithm is then developed to search the candidate design space and facilitate the computer implementation of assembly system configuration design. Two case studies including a real-world laptop assembly demonstrate the effectiveness of the proposed algorithm in the reduction of repetitive exploration of design space and avoidance of missing scenarios for assembly system configuration design by comparing with state-of-the-art assembly sequence generation algorithms. The method can lead to an automated tool to evaluate the manufacturability of product designs and optimise assembly system configuration design.

ARTICLE HISTORY
Received 21 December 2018
Accepted 4 November 2019

KEYWORDS
Assembly system design; assembly planning; assembly hierarchy; automatic generation algorithm

1. Introduction

Most industrial products are assembled from a plurality of basic components. These components can be connected to each other following certain patterns or structural relationship to form different subassembly modules. The subassemblies can be further combined with basic components or other subassemblies to form subassemblies at higher levels of assembly hierarchy. The final product can be created by a combination of multi-level subassemblies defined as an *assembly hierarchy* which determines all assembly operations and the material flow relations among them. Assembly operations can be implemented sequentially or parallelly according to their material flow relations.

Different assembly hierarchies can be adopted to form the same product. Before an assembly system can be generated, it is essential to explore all the possible assembly hierarchies given a product assembly design. For a serially linked product shown in Figure 1, the assembly hierarchy can be enumerated based on a string-parenthesis representation whereby product components are denoted by capital letters in the string, while assembly operations and their hierarchies are characterised by enclosing adjacent components using a set of parentheses (Li et al. 2011). For example, Table 1 shows the enumeration of all possible assembly hierarchies for the four-component product ABCD in Figure 1. The notation (ABCD) represents an assembly operation, by which four components A-D are joined simultaneously (e.g., multi-layer metal sheet joining) while (((AB)C)D) represents a sequential way of assembling AB, C, and D incrementally.

It should be noted that one key difference from the conventional assembly sequence generation problem is whether or not parallel assembly operations are considered. As shown in Figure 2, the notation ((AB)(CD)) depicts two assembly operations, i.e., (AB) and (CD), which can be performed parallelly and the assembly sequence between (AB) and (CD) does not impact assembly system. As such, two enumerations ((AB)(CD)) and ((CD)(AB)) are identical in assembly hierarchy whereas they are different in the assembly sequence.

CONTACT Hui Wang hwang10@fsu.edu Department of Industrial and Manufacturing Engineering, Florida A&M University-Florida State University College of Engineering, 2525 Pottsdamer St., Tallahassee, Florida, 32310, USA

Submitted to **International Journal of Computer Integrated Manufacturing**

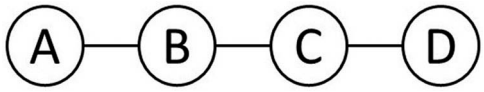

Figure 1. A serially linked assembly design.

Table 1. Possible hierarchy for product ABCD.

Case	Assembly hierarchies
1	(((AB)C)D)
2	((AB)CD)
3	((AB)(CD))
4	(A(BC)D)
5	((A(BC))D)
6	(A((BC)D))
7	(AB(CD))
8	(A(B(CD)))
9	((ABC)D)
10	(A(BCD))
11	(ABCD)

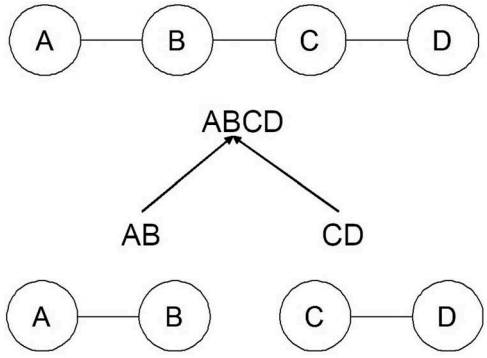

Figure 2. Hierarchy for case 3 of TABLE I, ((AB)(CD)).

Enumeration of the assembly hierarchies is of great significance to assembly system design including determination of assembly operations, assignment of assembly operations to machines, material flow among machines, and machine quantities.

The generation of assembly hierarchies proposed in this paper is motivated by manufacturers' needs for optimising the design of assembly system configuration, which is the topological arrangement of machines or workstations with defined logical material flow among them. The assembly system configuration is significantly affected by (1) the hierarchy of assembly tasks (assembly hierarchies) based on topological designs of products and (2) possible/feasible assembly tasks in such hierarchy that can be performed by the same machine/workstation. The design of the assembly system configuration consists of a two-level decision-making problem (Li et al. 2011). The first-level design determines candidate assembly tasks and material flow relationship among these tasks (i.e., assembly hierarchy selection), and the second-level design assigns the tasks to workstations/machines for workload balancing or cost reduction given the first-level design. Therefore, the generation of all possible assembly hierarchies is the first key step for assembly system configuration design by exploring all feasible solution space (Li et al. 2011; Hu et al. 2011).

Exploration of all the feasible assembly system designs usually includes generation of system configurations, generation of assembly sequences, and matching of sequences with configurations and appropriate operation assignments (Webbink and Hu 2005). By considering the parallel assembly operations, assembly hierarchy determines the assembly sequence (hierarchical structure of assembly tasks), the initial set-up for system configuration (material flows) and also potential assembly sequences, as shown in Figure 3. The assembly hierarchy contains rich information on the system configurations and the assembly sequence. An appropriate assembly hierarchy needs to be selected among all feasible assembly hierarchies at the very early stage of assembly system design. To select the assembly hierarchy, *a computer-implementable algorithm for exploring the design space of assembly hierarchies for given product design is of great importance*.

The process of the assembly sequence generation is usually based on a certain representation of an assembled product. Several representation methods are available based on the existing research work. A common method is bill-of-material (BOM) which has a tree-graph or tabular structure with hierarchical levels (Mather 1987) to list all parts, subassemblies, and materials. Another common representation is the graph or mathematical description of components and their physical connections such as liaison graph (De Fazio and Whitney 1987), adjacency matrix (Dini and Santochi 1992) and ontology-based representation (Kim, Manley, and Yang 2006). A review of assembly sequence representation methods was reviewed by Bahubalendruni, Biswal, and Khanolkar (2015). Based on the assembly sequence representation, several enumeration/generation algorithms were proposed. De Fazio and Whitney (1987) adopted the concept of liaison graph (Bourjault 1984) for generating assembly sequences. Park et al. (2013) developed a new type of parts liaison graph to generate the assembly sequences via analysed information such as the common area between parts, related ratio, and the number

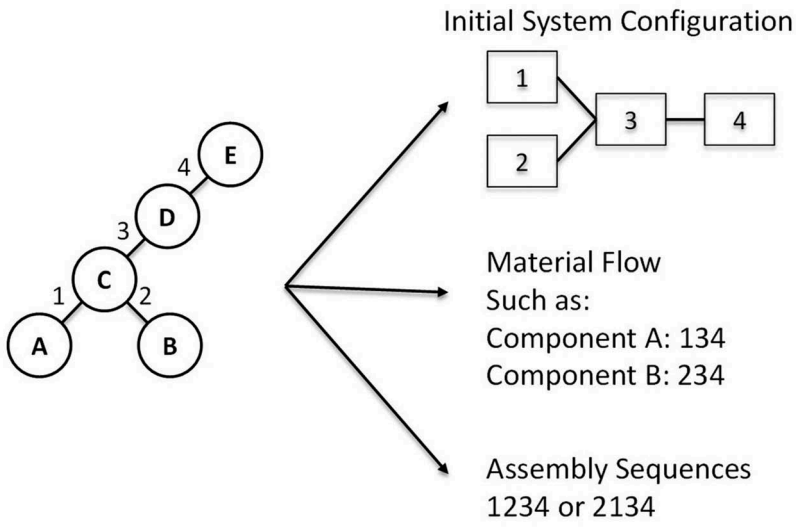

Figure 3. Information included in the assembly hierarchy.

of connected parts. AND/OR graph representation was also used to develop an algorithm which could generate all feasible assembly sequences (Mello and Sanderson 1990, 1991a). Mello and Sanderson (1991b) reviewed five types of representations for assembly sequences and established the mapping of one representation into the others. Bonneville, Henrioud, and Bourjault (1995) proposed a method for the assembly sequence generation considering ternary operations that allow three parts to assemble simultaneously. This method is an extension of the traditional assembly sequences planners. Pan, Smith, and Smith (2006) proposed a fully automated assembly sequence planner which directly extracts geometrical information from a CAD file and finds assembly sequences. Other approaches to the generation of assembly sequences include the computer-aided process planning (Ben-Arieh and Kramer 1994; Suszyński and Żurek 2015) and ant colony algorithm (Wang, Liu, and Zhong 2005). All these generated feasible configurations were used for recommending a good sequence of assembly operations in the second-level design (Choi et al. 1998). An automated initial population generator for genetic assembly planning was proposed by Smith and Smith (2003). Based on hypergraphs and directed graphs, Suszyński and Żurek (2015) proposed a computer programme Msassembly to generate assembly sequences for parts and machinery sets. Recent research of assembly sequence generation utilised heuristic methods and metaheuristic algorithms (Chen and Liu 2001; Akgündüz and Tunalı 2010; Kumar et al. 2011; Xing and Wang 2012; Ibrahim et al. 2015; Huang and Xu 2017). A detailed review on various methods of assembly sequence generation methods, their applications and limitations is presented and well discussed by Bahubalendruni and Biswal (2015).

Huang and Lin (2009) proposed a combinatorial way to calculate the total number of the candidate system configuration. Webbink and Hu (2005) enumerated system configurations by using parentheses to group a string of '1' characters. Similar approaches were employed to create the groups of product components or subassemblies in a study on the supply chain configuration by Wang et al. (2010). Massive research has been conducted on assembly sequence and system configurations. State-of-the-art approaches mostly deal with assembly sequence generations or subassembly module decomposition by studying mechanical interface. These methods have limitations in dealing with the ***research challenges/gaps*** for assembly system configuration design including:

- *A lack of methods exploring the design space without repetition*. Generation of the multiple assembly sequences based on the same assembly hierarchies can lead to repetitive search in the same design space. For example, the assembly sequence generation distinguishes the order of independent operations that can be implemented in parallel. The enumerated assembly sequences are repetitive for the same assembly hierarchy, reducing the search efficiency. There is

an essential need to develop a way of thoroughly exploring the design space without repetition.

- *Design space not sufficiently explored*. Parallel or independent assembly operations have not been well explored to improve assembly sequence generation and system design (Li et al. 2011; Hu et al. 2011). Bonneville, Henrioud, and Bourjault (1995) considered ternary operations in assembly sequence generation. However, simultaneous assembly of multiple components (more than 3) that can be performed by one single assembly operation is not considered and therefore, the design space is not sufficiently explored.
- *A lack of understanding of the relationship between complex product design and assembly hierarchy generation*. The enumeration of assembly hierarchies encounters challenges when the liaisons in a product assembly exhibit complex topology with loops and branches as shown in Figure 4, where the numbers represent liaisons between components. An appropriate logical representation should also be necessary to efficiently characterise the assembly hierarchy for complex product designs.

To address the challenges in assembly hierarchy generation for a product with complex liaisons, this paper proposes an approach to automatically generating assembly hierarchies. A representation using parentheses and numerical coding is adopted to characterise assembly hierarchy. Based on this representation, a tree-structured hierarchy model is proposed to recursively enumerate all assembly hierarchies without redundancy. This algorithm can be used not only for the products with serially linked liaisons but also for complex liaison with loops and branches.

The paper is organised as follows. Section 2 introduces the assembly hierarchy model and representations. The algorithm that realises the automatic assembly hierarchy generation is presented in Section 3. Section 4 presents case studies to verify and demonstrate the algorithm. Section 5 summarises the paper.

2. Assembly hierarchy model

This section proposes a new method of representing assembly hierarchy. Similar to De Fazio and Whitney (1987), the representation in this paper is developed based on liaison graphs. Each connection arc among nodes (components) in Figure 4 represents one basic assembly operation combining the two components. All the liaisons are numerically labelled. The spatial connection patterns among these labelled liaisons and assembly hierarchies can be represented by a - matrix **M** where each entry indicates whether two liaisons connect to the same component.

$$m_{i,j} = \begin{cases} 1, & \text{if liaison } i \text{ and } j \text{ connect to the same component}, \forall i \neq j \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

2.1 Subassembly representation by liaison grouping

An assembly hierarchy is expressed by grouping the numbered liaisons using parentheses. The liaison numbers correspond to different basic assembly operations, and a pair of parentheses represents one

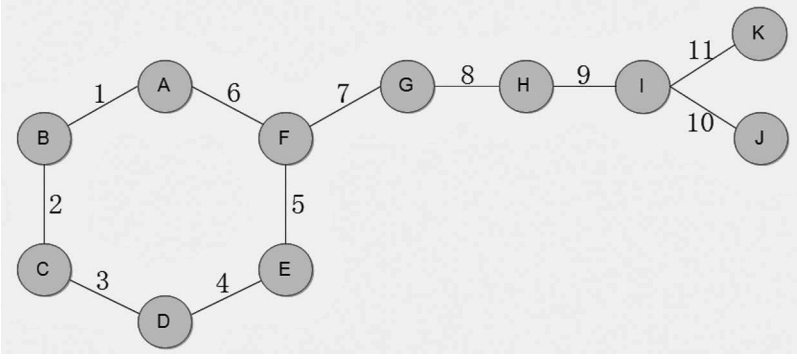

Figure 4. Liaison graph for a general product design.

step of assembly procedures yielding one subassembly. Based on the parenthesis and basic assembly operation numbers, the following representation rules on grouping liaisons are developed to characterise the assembly hierarchy.

Rule 1: One pair of parentheses generates only one subassembly.

Take Figure 4 as an example. The notation (2) represents a subassembly (BC) and (1 2 3) generates a subassembly (ABCD). But (1 3) violates this rule ('illegal') because if basic operations 1 and 3 are finished in the first step of the assembly process, two subassemblies are generated, i.e. (AB) and (CD).

Rule 1 can be represented by the matrix **M** defined above. All the liaisons (basic assembly operations) that are shown to be connected in matrix **M** can be grouped in one pair of parenthesis, representing a subassembly. For instance, consider a subassembly (ABCDE) in Figure 4 which has four basic assembly operations 1–4. The matrix **M** is

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

It is apparent that one basic assembly operation in each level is legal, such as (2). Notation (1 2 3) is also legal because according to matrix **M**, basic operations 1 and 2 connect to component B while basic operations 2 and 3 connect to component C, thus creating one subassembly. Notation (1 3) is not legal because basic operations 1 and 3 do not connect to any common component.

It should be noted that the notation (1 3) could become 'legal' for certain cases. For example, if the operation (2) has already been performed, a new subassembly (BC) is generated. Then basic operations (1) and (3) connect to the same subassembly (BC). Thus, operation (1 3) becomes legal. The reason for this scenario is that every time an assembly operation is performed, the matrix M defined above will change. In the next section, a recursive algorithm is proposed to generate all the assembly hierarchies given a liaison graph. In each recursive level, the connection matrix is different, and the algorithm needs to update matrix **M**. Thus, another changeable basic operation connection matrix **N**(k) should be defined, which represents whether two basic assembly operations connect to the same component or subassembly in the recursive level k. Each entry in this matrix is

$$m_{i,j} = \begin{cases} 1, & \text{if liaisons (tasks) } i \text{ and } j \text{ connect to} \\ & \text{the same component or subassembly} \\ & \text{in current recursive level } k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Different assembly operation selection of the upper level will generate different **N**(k). For the example in Figure 4, by considering basic assembly operations 1 to 4, the corresponding matrices **N**(1) and **N**(2) after assembly operation (2) is performed are

$$\mathbf{N}(1) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \xrightarrow{\text{after task(2) is performed}} \mathbf{N}(2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4}$$

After assembly operation (2) is performed, all the elements in row 2 and column 2 are set to zero. The updated matrix shows the new connection relation among the basic operations other than (2). In the updated matrix **N**(2), basic operations (1) and (3) are connected to the same subassembly. Therefore, (1 3) becomes legal and its completion creates subassembly (ABCD).

Rule 2: All the basic operation numbers in one pair of parentheses must be in ascending order.

Unlike assembly sequence generation in De Fazio and Whitney (1987), the assembly operations such as (1 2 3), (1 3 2), (2 1 3), (2 3 1), (3 1 2) and (3 2 1) in this representation are the same. To avoid the repetitive enumeration, this paper defines that all the basic operation numbers in one pair of parentheses (in the same level) should be sorted in ascending order. Thus, only (1 2 3) is legal among the six different forms above.

2.2 Tree structure of assembly hierarchy

The hierarchical relations among subassemblies can be represented by a multilevel tree structure. Under this tree structure, each node represents one pair of parentheses. Two scenarios are considered for each

node. One is that this node uses all or some of the subassemblies generated by other nodes and the other one is that it only uses the basic components.

Figure 5 shows an example of the tree structure representing the assembly hierarchy for the liaison graph in Figure 4. Operation (2) is defined as the child of (1 3) because the subassembly BC generated by (2) is used in operation (1 3). Similarly, nodes (1 3) and (5) are the children of (4 6), and (4 6), (8) and (10 11) are the children of (7 9).

In such a tree structure, if nodes (8) and (10 11) swap their positions, the subassembly hierarchy remains the same. To avoid such repetitive enumerations, this paper enforces that the children of the same node be arranged in ascending order from left to right according to the basic assembly operation label enclosed. The following representation rule is proposed:

Rule 3: All the children of the same node should be arranged in ascending order from left to right according to the smallest basic assembly operation label of each child node.

For example, assume that (1 4) and (2 3) are two children of one node. Due to Rule 3, (1 4)(2 3) is valid because basic operations (1) and (2) are in ascending order from left to right (only the smallest number in one pair of parentheses is used to arrange the order). An assembly hierarchy can thus be represented by traversing the tree defined above. However, it should also be noticed that assembly hierarchies (2)(1 3)(5)(4 6) and (2)(5)(1 3)(4 6) are the same in Figure 5. To avoid such a repetition, Rule 4 is enforced during the tree traversal.

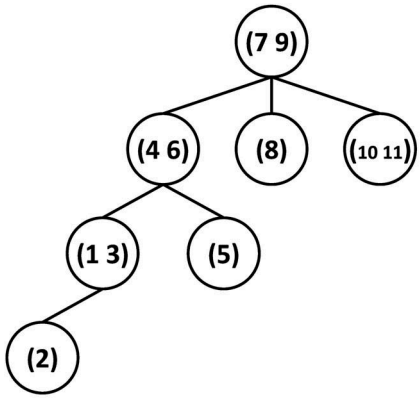

Figure 5. Example of a tree structure of assembly hierarchy.

Rule 4: The only legal expression of any assembly hierarchical tree is post-order traversal.

Post-order traversal of a tree structure starts from the root of the tree following the recursive traversal procedure below:

(1) Traverse all the children of the node from left to right.
(2) Visit the parent node.

Therefore, the only legal expression of the assembly hierarchy in Figure 5 is (2)(1 3)(5)(4 6)(8)(10 11)(7 9).

2.3 Summary

After applying Rules 1 to 4, any assembly hierarchy can be uniquely characterised by the new representation using basic assembly operation numbers and parentheses. Enumeration of all feasible assembly hierarchies without repeating or missing any case can be realised based on this representation. Therefore, the assembly hierarchy generation problem is transformed into an enumeration problem for all the legal arrangement of the basic assembly operation numbers (i.e., grouping these numbers using parentheses as constrained by Rules 1 to 4).

3. Assembly hierarchy generation algorithm

The main idea of the proposed algorithm is to enumerate all the feasible subassemblies recursively. Each recursion generates only one subassembly that should meet the requirements of Rules 1 to 4.

3.1 Algorithm for generating feasible subassembly that satisfies Rules 1 and 2

At each recursive level, it is necessary to first enumerate all the feasible subassemblies according to liaison graph while satisfying Rules 1 and 2. Suppose that in level k (recursive level appeared in Equation (3)), there are n basic operations, i.e. $1, 2 \ldots n$ (where the numbers are sorted in ascending order). A recursive method is used to generate all the combination of $1, 2 \ldots n$ in ascending order so that Rule 2 is met. For example, if there are only three basic operations to be examined, e.g. a, b and c ($a < b < c$). The candidate assembly operations to create subassemblies include (a), (b), (c),

(*ab*), (*bc*), (*ac*), and (*abc*). An algorithm is developed to test whether each candidate assembly operation containing more than one basic operations meets Rule 1. The procedures involved in this algorithm can be described as follows:

(1) Define a set $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$, where $a_1, a_2, \ldots, a_n$ are the basic operation numbers of one candidate assembly operation. Initialise set $\boldsymbol{Q} = \{\boldsymbol{Q}_n\} = \emptyset$, where $n = 1,2,3\ldots\ldots$
(2) Initialise $m = 1$. Record a random basic operation number of the candidate assembly operation to be tested in set $\boldsymbol{Q}_m$.
(3) Randomly pick one $j \in \boldsymbol{A} \cap \overline{\boldsymbol{Q}}$, $\overline{\boldsymbol{Q}}$ denotes the complement set of $\boldsymbol{Q}$, let $i \in \boldsymbol{Q}_m$, according to the matrix $\mathbf{N}(k)$:
    If $n_{i,j,k} = 1$
        Record j in $\boldsymbol{Q}_{m+1}$;
        Find next $j \in \boldsymbol{A} \cap \overline{\boldsymbol{Q}}$,
    Else
        Find next $j \in \boldsymbol{A} \cap \overline{\boldsymbol{Q}}$,
    Until all j is tested
(4) Inspect two scenarios:
    If $\boldsymbol{Q} \neq \boldsymbol{A}$ and $\boldsymbol{Q}_{m+1} = \emptyset$,
        This is an infeasible candidate assembly operation. Terminate.
    Else if $\boldsymbol{Q} = \boldsymbol{A}$
        This is a feasible candidate assembly operation. Terminate.
    Else
        Let $m = m + 1$ and return to Step **3**.
(5) Repeat this procedure until all the candidate assembly operations are tested.

The flowchart for 3.1 is given as follows in Figure 6.

3.2 Elimination of illegal enumerations by exploring tree structure (Rules 3 and 4)

In Section 3.1, the basic operation connection matrix $\mathbf{N}(k)$ is used to generate those feasible subassemblies which satisfy Rules 1 and 2. Next, the hierarchical relationship among these subassemblies as reflected in the tree structure needs to be identified. A string variable **P** is introduced to record the parents-children

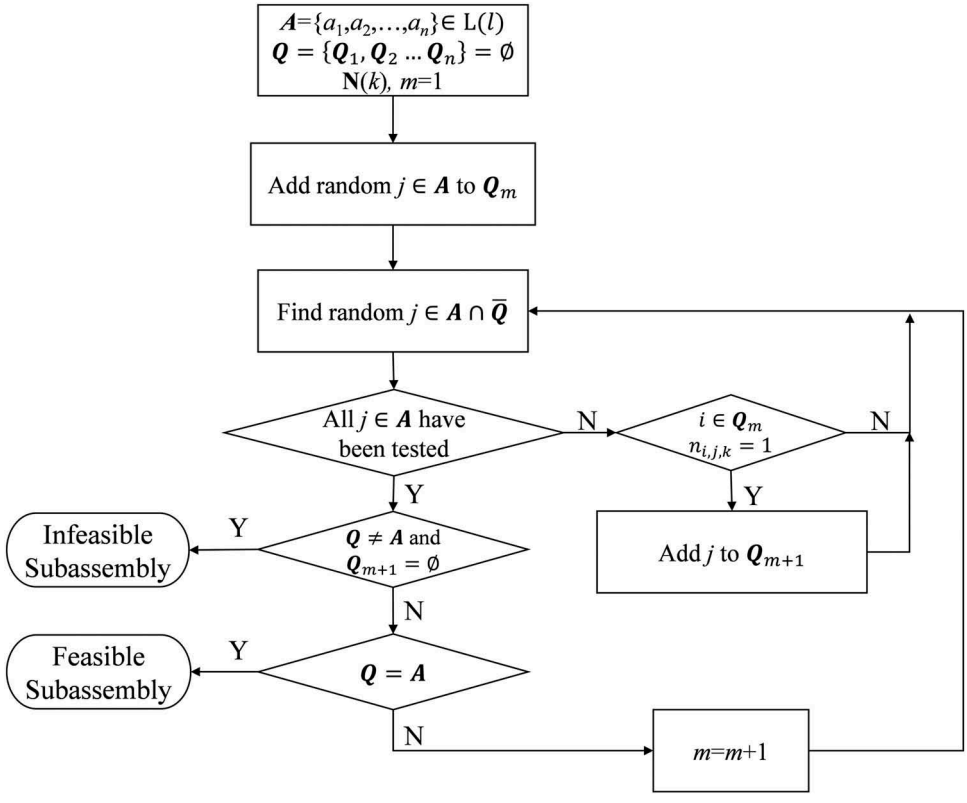

Figure 6. Flowchart of algorithm 3.1.

relationship of the tree structure. The procedures can be stated as follows:

(1) Save one feasible subassembly identified by the algorithm in Section 3.1 into **P**.
(2) Obtain the feasible subassembly in the next recursive level. There are three scenarios, i.e.,
   a. The new subassembly uses all the subassembly/subassemblies saved in **P**. Check Rule 3 to see if the smallest basic operation numbers of all these subassemblies are in ascending order.
      If Check pass
         Delete all the previous subassembly/subassemblies and replace it/them with the new feasible subassembly in **P**. Go to Step 3.
      Else
         Violates Rule 3, and an illegal enumeration is reported. Terminate.
   b. The new subassembly uses a part of the subassembly saved in **P**. Rule 4 requires that any parent node must be appended after its rightmost child while Rule 3 is not violated.
      If Check pass
         Delete the subassembly/subassemblies in **P** which are used by the new subassembly and replace it/them with the new subassembly. Go to Step 3.
      Else
         Violates Rules 3 or 4, and an illegal enumeration is reported. Terminate.
   c. The new subassembly uses none of the subassembly saved in **P**. It is considered to have a parallel hierarchical relationship with all the previous subassemblies. Append the new one after all the previous subassemblies in **P**. Go to Step 3.
(3) If there is no more subassembly and all the basic operations are examined, a feasible assembly hierarchy is obtained that satisfies Rules 3 and 4. Otherwise, Go to Step 2.

The flowchart for 3.2 is given as follows in Figure 7.

Take the hierarchical tree in Figure 8 as an example. Any expression other than (2)(1 3)(5)(4 6)(12)(8)(10 11)(7 9) will be considered illegal for this tree structure.

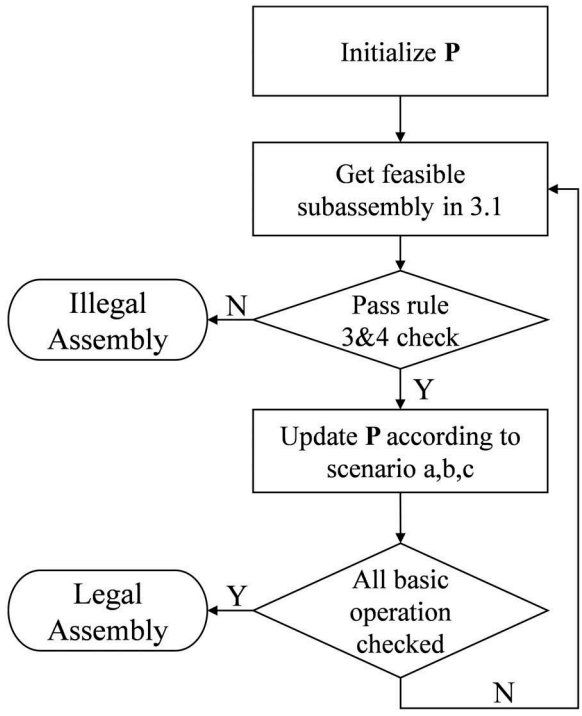

Figure 7. Flowchart of algorithm 3.2.

For real-world products, practical constraints such as geometry and manufacturability should be factored into the generation procedure. This paper considers each of these constraints as a filter. In each recursive level, the filter eliminates all candidate assembly operations that violate the constraint. As such, redundant generation for those impractical operations will not be performed, thereby reducing computational load.

3.3 Recursive algorithms for the enumeration

Based on the algorithms in Sections 3.1–3.2, a recursive algorithm can be developed for enumerating all the legal representations. Two recursive functions CandiOp and AssemHi are developed where AssemHi (Assembly Hierarchy Generation) is a function to generate the lower recursive level, and CandiOp (Candidate Operation Enumeration) is to enumerate all the candidate assembly operations in each recursive level. In recursive level k, follow the five steps below:

(1) Initialise **N**(1) and **P** and call AssemHi for the first recursive level ($k = 1$);
(2) In level k, call CandiOp(k) function to one candidate assembly operations based on the basic operations to be examined;

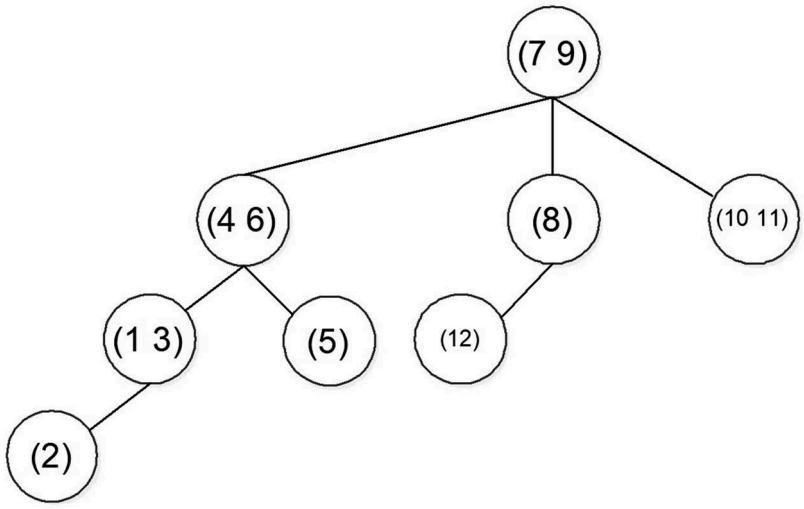

Figure 8. Example of a tree structure.

(3) For the candidate assembly operation generated above, find out whether this operation satisfies (a) Rules 1 and 2 by examining **Q** and **N**(*k*) using the algorithm in Section 3.1. and (b) Rules 3 and 4 by checking string **P** in level *k* using the algorithm in Section 3.2.
   a. If the operation obeys Rules 1 to 4 and practical constraints, proceed to Step 4.
   b. Otherwise, go to Step 2 and call CandiOp(*k*) function to examine the next candidate assembly operation in level *k*.
(4) Record the identified legal assembly hierarchy in level *k* in string variables **S**(*k*), *k* = 1,2 … *n*, where *n* is the total the number of basic operations;
(5) Judge whether all the basic operations are recorded in **S**(1), **S**(2), …, **S**(*k*).
   a. If Yes: Print **S**(1), **S**(2), …, **S**(*k*) as an assembly hierarchy. Go to Step 2 and Call CandiOp(*k*) function to examine the next candidate assembly operation in level *k*.
   b. If No: Update matrix **N**(*k* + 1) and string variable **P**. Call AssemHi(*k* + 1) function at recursive level *k* + 1 which call CandiOp(*k* + 1) function at Step 2 to generate the candidate assembly operations for level *k* + 1.

The flowchart of this programme is shown in Figure 9, where *L* represents the total number of the combinations of candidate assembly operations in the same recursive level generated by the algorithm in Section 3.1.

4. Examples and validation

This section shows several examples to verify and demonstrate this proposed algorithm. When the liaison relations of the products become increasingly complex, the design space grows larger, and computation becomes more expensive. Comparison with the traditional assembly sequence generation, this research addresses the following challenges in the assembly hierarchy generation, i.e.,

- The information based on hierarchy analysis, such as assembly task formation and hierarchical material flow relationship among the tasks, can be used to filter unnecessary candidate solutions that the assembly sequence generation cannot eliminate from the design space.
- Assembly sequence generation does not consider the case when certain basic operations may be performed simultaneously via an assembly operation that encloses/groups multiple components at a time. For example, multiple metal plates can be welded simultaneously via a multi-layer joining process.

4.1 Case study for the comparison with assembly sequence generation

To compare the difference between assembly sequence and assembly hierarchy, this paper uses the ballpen assembly example in De Fazio and Whitney

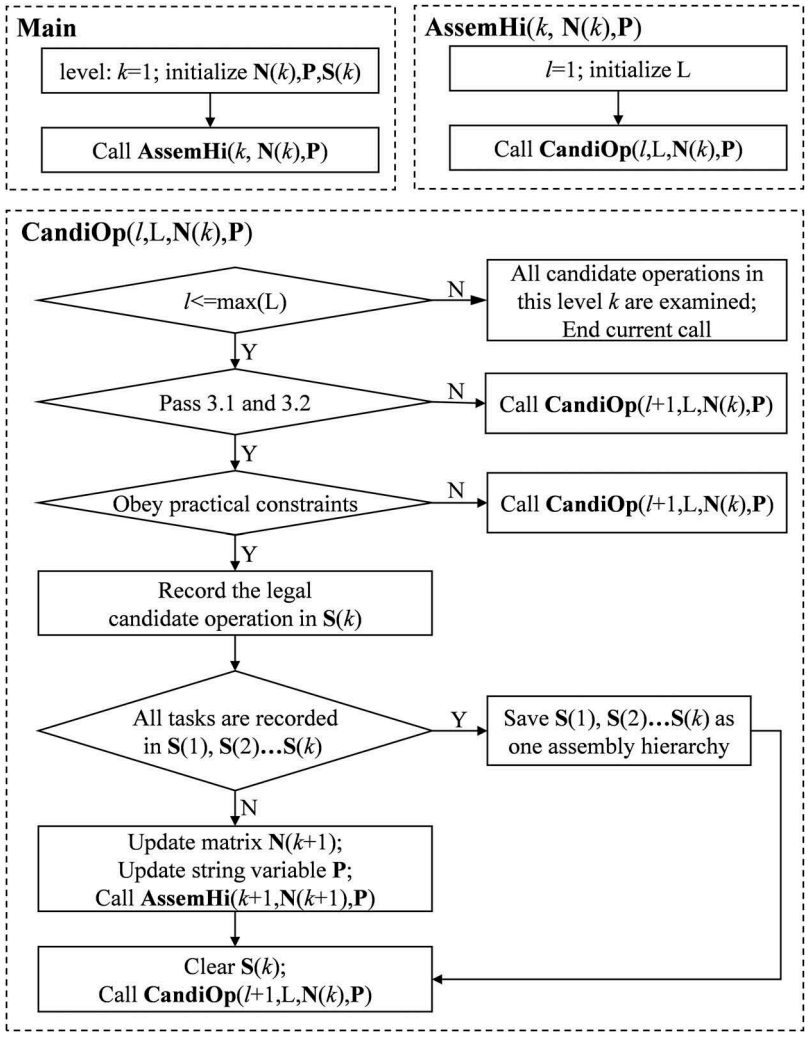

Figure 9. Flowchart of enumeration algorithms using 3.1 and 3.2.

(1987). The liaison graph of the ballpen is given in Figure 10. A description of precedence requires the following liaison precedence relations: 3→4, 1→5 and 4→(1 and 2), where $a \rightarrow b$ represents that basic operation a must have been performed before basic operation b. The proposed algorithm for assembly hierarchy generation is applied, and the results are presented in Table 2.

Compared with De Fazio's method, using assembly hierarchy generation for assembly system design and optimisation can avoid repetitive considerations of the assembly sequences that have the same hierarchical

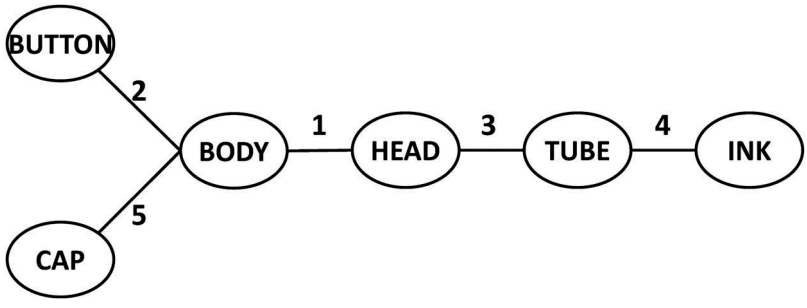

Figure 10. Liaison graph of a ballpen.

structures. The assembly sequence generation algorithm in (De Fazio and Whitney 1987) yields 12 assembly cases. Ten of these cases are the results labelled with stars in Table 2. The remaining two results are (3)(2)(4)(1)(5) and (3)(4)(2)(1)(5), which are considered equivalent to Result 13 = (2)(3)(4)(1)(5) from the perspective of assembly hierarchy. The design space without simultaneous assembly is simplified from 12 to 10. In addition, the proposed algorithm generates 15 more assembly hierarchies considering the case when certain basic operations can be performed simultaneously while satisfying the requirements of the liaison precedence relations. These 15 scenarios are ignored by state-of-the-art assembly sequence generation algorithms. Furthermore, by analysing the tree structure of the assembly hierarchy, more constraints can be added during the generation procedure. For example, the head, tube, and ink may be enclosed in a refill module and should be produced in a subassembly and on the tree structure; these assembly tasks will be placed on a subassembly branch. By implementing these constraints during generation, the resultant design space is reduced to 5 as shown in Figure 11.

Table 2. Assembly hierarchy generation result for Figure 8.

	Result 1 = (1 3)(4 5)(2)		Result 14 = (3)(1 4)(2 5)
	Result 2 = (1 3)(4)(2 5)		Result 15 = (3)(1 4)(2)(5)
	Result 3 = (1 3)(4)(2)(5)		Result 16 = (3)(1 4)(5)(2)
	Result 4 = (1 3)(4)(5)(2)		Result 17 = (3)(1)(4 5)(2)
	Result 5 = (1 3)(5)(4)(2)		Result 18 = (3)(1)(4)(2 5)
	Result 6 = (1)(3 5)(4)(2)	★	Result 19 = (3)(1)(4)(2)(5)
	Result 7 = (1)(3)(4 5)(2)	★	Result 20 = (3)(1)(4)(5)(2)
	Result 8 = (1)(3)(4)(2 5)	★	Result 21 = (3)(1)(5)(4)(2)
★	Result 9 = (1)(3)(4)(2)(5)		Result 22 = (3)(4)(1 2)(5)
★	Result 10 = (1)(3)(4)(5)(2)		Result 23 = (3)(4)(1)(2 5)
★	Result 11 = (1)(3)(5)(4)(2)	★	Result 24 = (3)(4)(1)(2)(5)
★	Result 12 = (1)(5)(3)(4)(2)	★	Result 25 = (3)(4)(1)(5)(2)
★	Result 13 = (2)(3)(4)(1)(5)		

★ Same results in (De Fazio and Whitney 1987)

4.2 A real-world example: laptop computer assembly

Figure 12 shows the components for a simplified laptop computer example as used by Hu et al. (2011). In the graph, the replaceable components, such as the hard drive and the main battery, are excluded from the computer assembly process. The corresponding liaison graph is given in Figure 13.

The practical constraints for assembling this product are discussed as follows. Assume that from the manufacturability point of view, the following relations must be maintained, i.e., 1→2, 1→3, 1→4, ((8 or 6) and 7 and 9 and 10)→5, 10→7, 10→9, 7→9. Another constraint considers the completion of certain basic operations that result in the simultaneous completion of other basic operations. This scenario typically occurs when several liaisons form a loop. The liaison graph in Figure 11 shows that basic operations 2, 3 and 4 form a circle. When basic operation 2 is completed, basic operations 3 and 4 must be performed simultaneously to complete the assembly for the liaison loop because liaisons 3 and 4 both represent the contact between subassembly (BC) and a component D. The processing time of assembly tasks is given in Table 3.

If the algorithm is implemented without any practical constraints, it becomes a complete enumeration of all possible assembly hierarchies. The algorithm may take hours to generate 19,224,300 different assembly hierarchies. However, in reality, there are different practical constraints mentioned above. With these practical constraints, the algorithm results in 2156 legal assembly hierarchies within 10 seconds, as summarised in Figure 14. Among these feasible assembly hierarchies, 376 of them are the assembly hierarchies without the

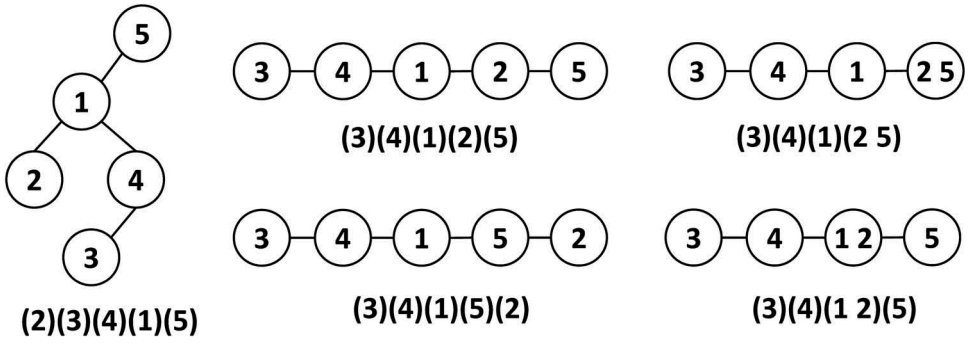

Figure 11. Enumeration results with additional constraints for ballpen assembly.

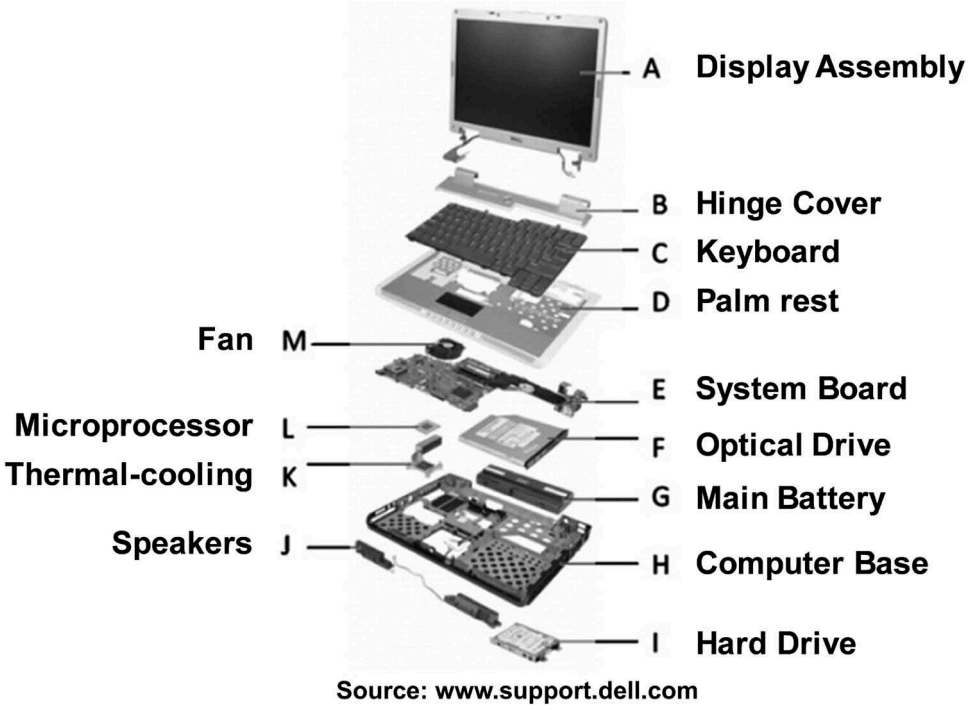

Figure 12. Components of a laptop computer.

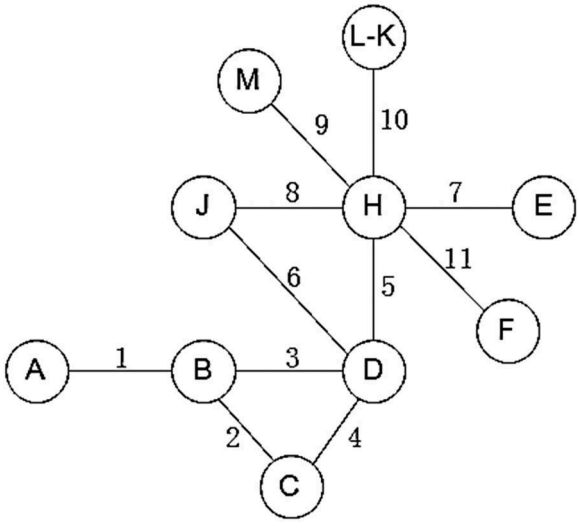

Figure 13. Liaison graph for a laptop computer.

Table 3. Processing time of assembly tasks (min).

Tasks	1	2	3	4	5	6	7	8	9	10	11
Time	2.2	1.3	1.7	1.1	3.2	3.7	4.2	3.1	2.0	5.0	2.5

consideration of simultaneous assembly (unless required by the constraints). If the assembly sequence generation is adopted, the size of the candidate design space will become 12,096 while many sequences are representing the same hierarchies. By using assembly hierarchy, the size of the design space is reduced from 12,096 to 376. Scenarios with simultaneous assembly are also considered without the limitation to three parts assembly proposed by Bonneville, Henrioud, and Bourjault (1995). When the simultaneous assembly (not limited to ternary operation) is considered, the design space extended to 2156 with 5, 6, 7, 8, and 9 operations. Thus, the proposed algorithm recovers 2156–376 = 1780 scenarios that were ignored by the traditional assembly sequence generation problem. Some examples are illustrated in Figure 15. The nodes represent the assembly tasks or groups of assembly tasks that create/co-create one subassembly, and the arcs represent their precedence relations. By assuming that each node is assigned to an individual machine, the Gantt charts and makespans are listed on the right side of the examples, where the makespan represents the time difference between the start and finish of a sequence of assembly tasks. It can be seen that the 8-stage assembly planning leads to the minimal makespan among the examples as listed and can be potentially chosen for assembly system configuration design. The makespans in Figure 15 can be further optimised by assigning multiple machines to one node or grouping several nodes into one machine.

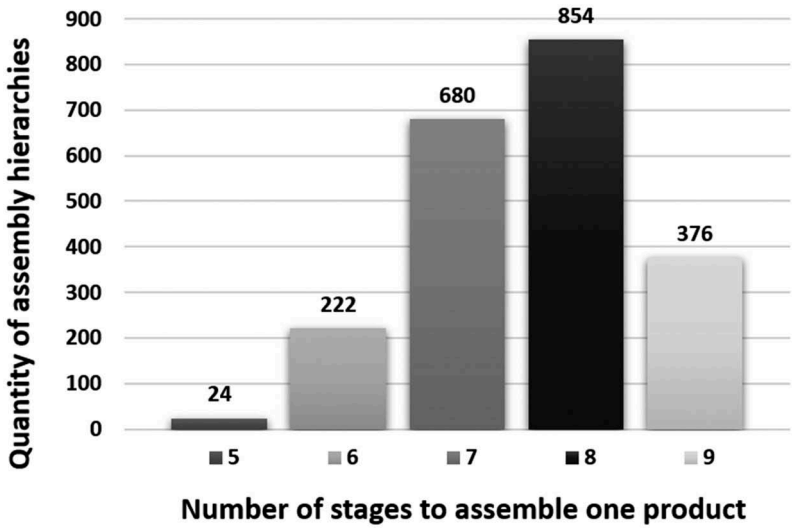

Figure 14. Summary of the assembly hierarchy generation for the laptop.

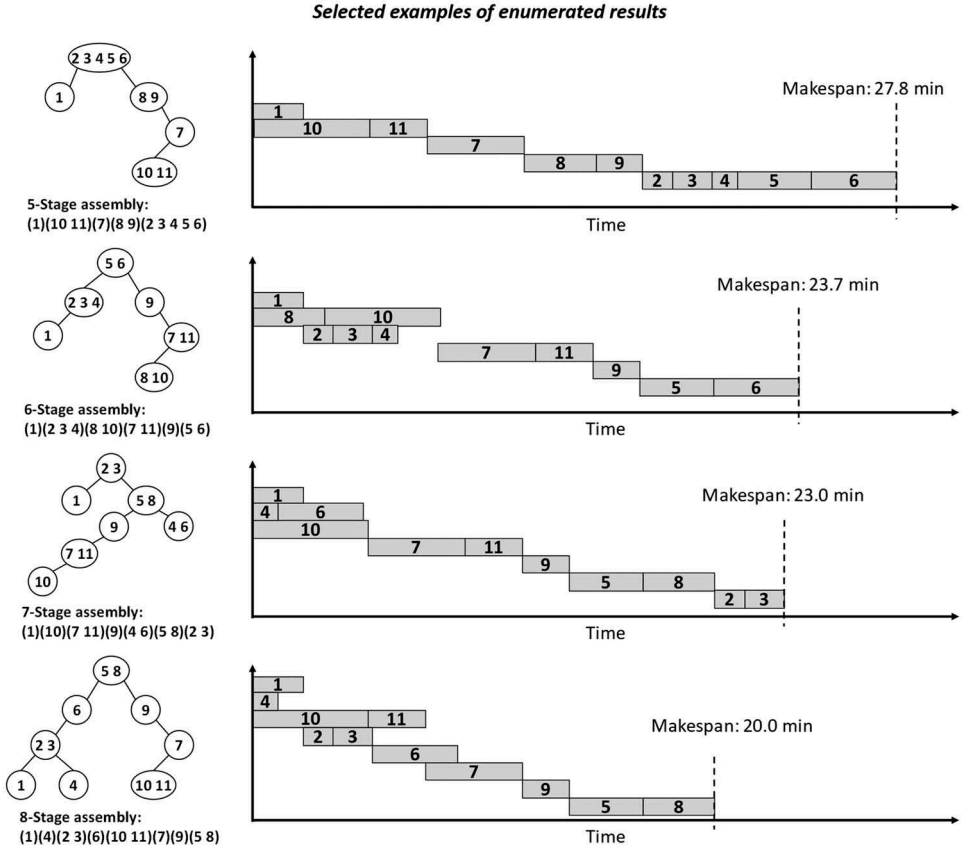

Figure 15. Examples of enumerated results including hierarchical configurations, mathematical representations, and corresponding Gantt charts showing the makespans of completing the assembly.

The assembly sequence must distinguish the strict sequence between assembly operations of different assembly liaisons and as such, the sequence for those parallel/independent assembly operations must be identified. By contrast, the assembly hierarchy only focuses on the hierarchical structure/relationship among the material flows for different assembly operations and does not distinguish between the assembly sequence for those parallel/independent operations. When dealing with the problem of

assembly system configuration design, which concerns with the topological arrangement of machines or workstations with defined logical material flow among them, the assembly hierarchy has a clear advantage in exploring the hierarchical relations in the assembly operations without repetitive search in the same design space.A comparison of the design space between the proposed algorithm using assembly hierarchy and the traditional assembly sequence generation is given in Table 4. The numerical results demonstrate that the proposed algorithm can avoid repetitive design space search while not missing scenarios when simultaneous assembly operations are involved in comparison with the assembly sequence generation .

4.3 Discussion

This research provides a maths-based tool implementable by computers for engineers to evaluate the product design manufacturability. Given a product design, the algorithm can generate the entire feasible design space considering all kinds of constraints including the designers' preferences. The algorithm output can be further used as the input for assembly system configuration optimisation. For example, Li et al. (2011) proposed an optimisation framework for designing assembly systems with complex configurations by jointly considering product design hierarchy, line balancing, and equipment selection. Such a configuration of assembly systems reflects the topological arrangement of workstations/machines and material flows among them. A two-stage optimisation algorithm has been developed to explore all the possible solutions to the assembly system configuration based on an initial configuration as shown in Figure 16, where a circle represents the assembly tasks, the dashed boxes represent machines. The initial configuration is directly derived from each assembly hierarchy by assigning one task to each machine, and it is evolved/updated by exploring feasible ways of task-machine assignments to evaluate various serial, parallel, and hybrid configurations. This research provides a computer-aided generation algorithm for the assembly hierarchy, which can be directly fed to the optimisation outlined by Li et al. (2011) as the initial configurations. The first version of this tool is being used by the Research and Development Centre of one major automotive manufacturer in the USA to evaluate their electric vehicle battery designs. If the initial configuration in Li et al. (2011) were created based on the traditional assembly sequence method, the computation would be less efficient for a large-sized problem since the optimisation has to explore a large number of unnecessary assembly sequences that, however, correspond to the same assembly hierarchy.

Conclusion

Automatic generation of assembly hierarchies plays an essential role in assembly system design as it determines the assembly operations and the hierarchical material flows. The assembly hierarchy contains the information useful for optimising the assembly configuration and the assembly sequence. State-of-the-art research mostly dealt with assembly sequence generation and subassembly module formation considering mechanical interface designs. ***Research gaps*** still exist in the development of an assembly hierarchy generation algorithm including (1) an efficient representation of assembly hierarchy for complex product designs and (2) the consideration of parallel assembly tasks and simultaneous completion of multiple assembly tasks.

This paper proposed a computer-aided algorithm to generate assembly hierarchies for product designs with complex liaison relations. A new assembly hierarchy representation with four rules was proposed. Based on the new representation, a recursive algorithm was developed to automatically generate all the feasible assembly hierarchies. Using string representation and the connection matrix of liaison graph, this algorithm can generate assembly hierarchies not only for serially linked assemblies but also for

Table 4. Comparison between assembly hierarchy and assembly sequence in case studies.

	Design space without considering simultaneous operations			
Scenarios	Assembly sequence method	Assembly hierarchy method	Design space reduction %	Additional design space considering simultaneous operations
Case study 1	12	10	16.67%	15
Case study 2	12,096	376	96.89%	1780

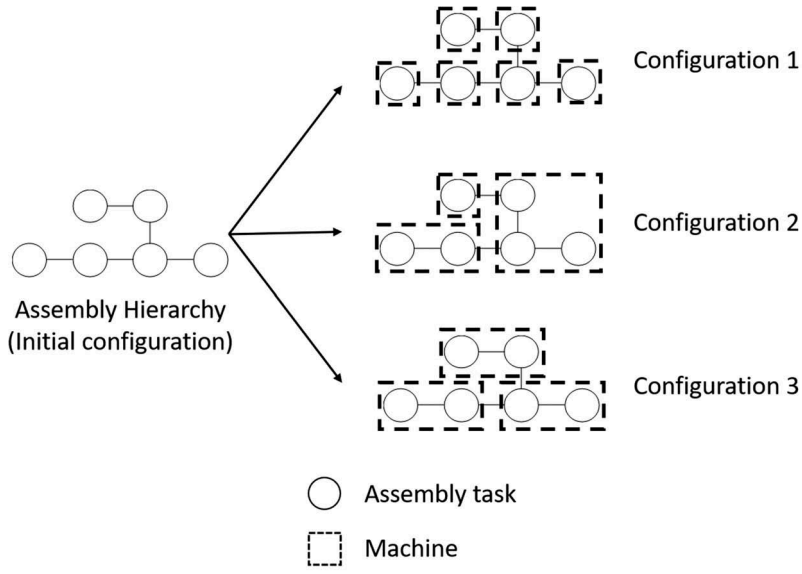

Figure 16. Assignment of tasks to machines based on one initial configuration.

assemblies with branches and loops in their liaison graphs. This generation algorithm is a recursive procedure, and in each recursive level, all 'illegal' results that violate the four rules and/or some practical constraints are not explored, thus greatly improving the computational efficiency. Case studies were conducted for a ballpen liaison graph and a real-world product (laptop) to demonstrate the procedure. A comparison was made to illustrate the difference between assembly hierarchy generation and conventional assembly sequence generation problem. The advantages of the proposed method (innovations) are twofold including (1) reduction of the assembly sequences corresponding to the same assembly hierarchy, thus reducing the repetitive exploration/search of the same design space for solving the assembly system configuration design problems and (2) consideration of simultaneous assembly operations, which are usually ignored by assembly sequence generation algorithm. Therefore, the proposed algorithm has its advantage in improving the search efficiency of design space for assembly hierarchy selection in the assembly system configuration design problem.

The realisation of the automatic assembly hierarchy generation provides a computer-aided tool for optimisation algorithms to select appropriate system configurations, thereby leading to a computer-aided tool for engineers to evaluate the manufacturability of product designs.

Acknowledgments

This research is partially supported by an NSF grant HRD-1646897 and has been conducted at the FAMU-FSU College of Engineering. The authors also thank Prof. S. Jack Hu at the University of Michigan and Dr. Yhu-tin Lin from the GM Technical Center for providing industry backgrounds that motivate this research.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the National Science Foundation Grants CMMI-1901109 and HRD-1646897.

References

Akgündüz, O. S., and S. Tunalı. 2010. "An Adaptive Genetic Algorithm Approach for the Mixed-model Assembly Line Sequencing Problem." *International Journal of Production Research* 48: 5157–5179. doi:10.1080/00207540903117857.

Bahubalendruni, M., B. B. Biswal, and G. R. Khanolkar. 2015. "A Review on Graphical Assembly Sequence Representation Methods and Their Advancements." *Journal of Mechatronics and Automation* 1: 16–26.

Bahubalendruni, M. R., and B. B. Biswal. 2015. "A Review on Assembly Sequence Generation and Its Automation." Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 230: 824–838. doi:10.1177/0954406215584633.

Ben-Arieh, D., and B. Kramer. 1994. "Computer-aided Process Planning for Assembly: Generation of Assembly Operations Sequence." *The International Journal of Production Research* 32: 643–656. doi:10.1080/00207549408956957.

Bonneville, F., J. M. Henrioud, and A. Bourjault. 1995. "Generation of Assembly Sequences with Ternary Operations." In *Proceedings. IEEE International Symposium on Assembly and Task Planning*, 245–249. doi:10.1002/bip.360360211

Bourjault, A. 1984. "Contribution to a Methodological Approach of Automated Assembly: Automatic Generation of Assembly Sequence". University de Franche-Comte.

Chen, S.-F., and Y.-J. Liu. 2001. "An Adaptive Genetic Assembly-sequence Planner." *International Journal of Computer Integrated Manufacturing* 14: 489–500. doi:10.1080/09511920110034987.

Choi, C. K., X. F. Zha, T. L. Ng, and W. S. Lau. 1998. "On the Automatic Generation of Product Assembly Sequences." *International Journal of Production Research* 36: 617–633. doi:10.1080/002075498193606.

De Fazio, T. L., and D. E. Whitney. 1987. "Simplified Generation of All Mechanical Assembly Sequences." *Robotics and Automation, IEEE Journal Of* 3: 640–658. doi:10.1109/JRA.1987.1087132.

Dini, G., and M. Santochi. 1992. "Automated Sequencing and Subassembly Detection in Assembly Planning." *CIRP Annals - Manufacturing Technology* 41: 1–4. doi:10.1016/S0007-8506(07)61140-8.

Hu, S. J., J. Ko, L. Weyand, H. ElMaraghy, T. Lien, Y. Koren, H. Bley, G. Chryssolouris, N. Nasr, and M. Shpitalni. 2011. "Assembly System Design and Operations for Product Variety." *CIRP Annals-Manufacturing Technology* 60: 715–733. doi:10.1016/j.cirp.2011.05.004.

Huang, N., and Y.-T. Lin. 2009. "Chaining Set Partitions with Applications in Manufacturing System Configuration Planning." *International Journal of Operational Research* 6: 380–404. doi:10.1504/IJOR.2009.026939.

Huang, W., and Q. Xu. 2017. "Automatic Generation and Optimization of Stable Assembly Sequence Based on ACO Algorithm." In *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2057–2062. Takamatsu, Japan, August.

Ibrahim, I., Z. Ibrahim, H. Ahmad, M. F. M. Jusof, Z. M. Yusof, S. W. Nawawi, and M. Mubin. 2015. "An Assembly Sequence Planning Approach with a Rule-based Multi-state Gravitational Search Algorithm." *The International Journal of Advanced Manufacturing Technology* 79: 1363–1376. doi:10.1007/s00170-015-6857-0.

Kim, K.-Y., D. G. Manley, and H. Yang. 2006. "Ontology-based Assembly Design and Information Sharing for Collaborative Product Development." *Computer-Aided Design* 38: 1233–1250. doi:10.1016/j.cad.2006.08.004.

Kumar, M. S., M. N. Islam, N. Lenin, D. Vignesh Kumar, and D. Ravindran. 2011. "A Simple Heuristic for Linear Sequencing of Machines in Layout Design." *International Journal of Production Research* 49: 6749–6768. doi:10.1080/00207543.2010.535860.

Li, S., H. Wang, S. J. Hu, Y.-T. Lin, and J. A. Abell. 2011. "Automatic Generation of Assembly System Configuration with Equipment Selection for Automotive Battery Manufacturing." *Journal of Manufacturing Systems* 30: 188–195. doi:10.1016/j.jmsy.2011.07.009.

Mather, H. 1987. *Bills of Materials*. Burr Ridge, Illinois: Irwin Professional Pub.

Mello, L. S. H. D., and A. C. Sanderson. 1990. "AND/OR Graph Representation of Assembly Plans." *Robotics and Automation, IEEE Transactions On* 6: 188–199. doi:10.1109/70.54734.

Mello, L. S. H. D., and A. C. Sanderson. 1991a. "A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences." *IEEE Transactions on Robotics and Automation* 7: 228–240. doi:10.1109/70.75905.

Mello, L. S. H. D., and A. C. Sanderson. 1991b. "Representations of Mechanical Assembly Sequences." *IEEE Transactions on Robotics and Automation* 7: 211–227. doi:10.1109/70.75904.

Pan, C., S. S. Smith, and G. C. Smith. 2006. "Automatic Assembly Sequence Planning from STEP CAD Files." *International Journal of Computer Integrated Manufacturing* 19: 775–783. doi:10.1080/09511920500399425.

Park, H.-S., J.-W. Park, M.-W. Park, and J.-K. Kim. 2013. "Development of Automatic Assembly Sequence Generating System Based on the New Type of Parts Liaison Graph." In *Product Lifecycle Management for Society. PLM 2013. IFIP Advances in Information and Communication Technolog*, edited by A. Bernard, L. Rivest, D. Dutt. vol 409. Berlin, Heidelberg: Springer..

Smith, G., and S. Smith. 2003. "Automated Initial Population Generation for Genetic Assembly Planning." *International Journal of Computer Integrated Manufacturing* 16: 219–228. doi:10.1080/0951192021000039602.

Suszyński, M., and J. Żurek. 2015. "Computer Aided Assembly Sequence Generation." *Management and Production Engineering Review* 6: 83–87. doi: 10.1515/mper-2015-0030.

Wang, H., J. Ko, X. Zhu, and S. J. Hu. 2010. "A Complexity Model for Assembly Supply Chains and Its Application to Configuration Design." *Journal of Manufacturing Science and Engineering* 132: 021005. doi:10.1115/1.4001082.

Wang, J., J. Liu, and Y. Zhong. 2005. "A Novel Ant Colony Algorithm for Assembly Sequence Planning." *The International Journal of Advanced Manufacturing Technology* 25: 1137–1143. doi:10.1007/s00170-003-1952-z.

Webbink, R. F., and S. J. Hu. 2005. "Automated Generation of Assembly System-design Solutions." *Automation Science and Engineering, IEEE Transactions On* 2: 32–39. doi:10.1109/TASE.2004.840072.

Xing, Y., and Y. Wang. 2012. "Assembly Sequence Planning Based on a Hybrid Particle Swarm Optimisation and Genetic Algorithm." *International Journal of Production Research* 50: 7303–7312. doi:10.1080/00207543.2011.648276.