CONCURRENT ENGINEERING: Research and Applications

# Formulation and Search of Assembly Sequence Design Spaces for Efficient Use of Assembly Plant Resources for New Products

**Shilpitha Bomi Reddy and Zahed Siddique***

*School of AME, University of Oklahoma, Norman, OK 73019, USA*

**Abstract:** Efficient procedures for generation of feasible assembly sequences and effective utilization of available assembly plant resources can greatly reduce the development time and cost of platforms for new product family members. This article presents a method to generate feasible assembly sequences and an approach to select an assembly process that reduces the existing plant modification cost. Assembly sequence design space is combinatorial in nature. Mathematical models to solve the effects of constraints on these spaces and algorithms to efficiently enumerate feasible spaces are explored in this research. Algorithms to search the feasible space to identify assembly process that can reduce the modification cost of the existing assembly plant can help increase utilization of existing resources. A software application that implements the method and algorithms has been developed. The algorithms use the concept of recursive partitioning of set of components to generate assembly sequence space. The assembly processes are then evaluated to determine the process that maximizes resource utilization for new platforms. The application of the proposed approach is demonstrated using automotive underbody front structure family.

**Key Words:** assembly reasoning, assembly sequence design space, assembly resource utilization.

## 1. Introduction

In order to gain advantage over their competitors or to survive in the current turbulent market, companies need to introduce newer models more frequently. One approach is to organize similar products into a product family that typically a share common platform. Companies can further reduce product development time and cost through efficient management and utilization of existing resources [1–5]. The increasing demand for product variety force companies to make frequent changes to their product design, manufacturing process, assembly plants, etc. In order to shorten the product design time, companies need to develop decision tools for assembly process planning [6–9]. The present work is focused on developing a decision tool that automates generation of feasible assembly sequences from a set of constraints and selection of an assembly sequence to minimize changes to existing plant for new platforms.

Utilization of available assembly resources can greatly reduce development time and cost for platforms and new product family members [4,5]. All members of a family have similar assembly processes in general and have the potential to be assembled using the same assembly line [10]. Combinations of assembly processes for the similar member products and platforms make up the assembly line for the entire product family. The design of the assembly process and selection of assembly plants are usually performed after the product is designed. This article focuses on the development of a method to determine an assembly sequence for new product family members that utilize existing assembly plant resources. The article also focuses on developing a software application to automatically generate feasible assembly sequences for a new product family member. The objectives of this research are to: (1) develop a method for enumerating assembly sequence design spaces (Section 3); (2) solve the effects of constraints and determine feasible assembly sequence space (Section 4); (3) select an assembly sequence by utilizing existing assembly plant resources that requires minimum modification to the existing plant. (Section 5); (4) develop a software application to automatically generate an assembly process for new product family members using existing resources.

## 2. Background

### 2.1 Assembly Sequence Representation

In order to generate feasible assembly space, first assembly sequences need to be represented. The assembly sequence diagram representation was developed by

*Author to whom correspondence should be addressed.
E-mail: zsiddique@ou.edu
Figures 1, 2, 4 and 5 appear in color online: http://cer.sagepub.com


1063-293X/10/02 0129–12 $10.00/0 DOI: 10.1177/1063293X10370798
© The Author(s), 2010. Reprints and permissions:
Downloaded from SHAREOK. Persistent Handle URL : https://hdl.handle.net/11244/25241
Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 26, 2016

De Fazio and Whitney [11], which corrects some of the drawbacks in Bourjault [12] tree representation. Delchambre [13] proposed assembly plans, which consists of precedence graphs. Haung and Lee [14] addressed the representation and acquisition of precedence knowledge of an assembly, which plays an important role in the generation of assembly sequence and planning.

Homem de Mello and Sanderson [15] used AND/OR graph to represent all mechanical assembly sequences. Directed graphs are used to represent a set of all assembly sequences [16]. The nodes in this graph correspond to stable state partition of set of parts. The edges in this graph are ordered pairs of nodes and each edge corresponds to an assembly task. HAP can overcome difficulties associated with computational complexities in case of large assemblies.

### 2.2 Generation of Assembly Sequences

Several methods have been proposed to solve for generation of assembly sequences. Forward assembly sequence search or backward assembly sequence search are based on traditional tree search, graph theory, artificial intelligence, and disassembly approach. Bourjault [12] developed means for generating all assembly sequences from a series of rules using algorithms based on precedence relations. De Fazio and Whitney [11] used a similar concept by simplifying his technique, which includes a smaller set of questions. Homem de Mello and Sanderson [17] developed a program that takes input as a representation of a product and generates all feasible assembly sequences presenting them as AND/OR graph representation of assembly sequences.

Several authors proposed heuristics to generate a set of feasible assembly plans. Generative assembly process planner [18] takes input as solid model from which a suitable assembly representation is formed as a graph model. Lee and Shin [19] presented an assembly planning system COPLANNER. The XAP/1 system [20] is oriented towards plan optimization rather that generating all feasible plans. Lin et al. [21] uses contact relation matrix (CRM) approach to generate assembly sequences for product design.

Assemble planning problem presented by Bonneville et al. [22] uses genetic algorithm (GA) with the initial population is composed of few assembly plans examined by an expert. The approach of assembly sequence planning problem [23] is a modified version of GA with the search space clustered as families of sequences having similar genetic characteristics.

The disadvantages of the nonheuristics methods are the combinatorial explosion as the number of components in products increases. These methods also require the user to validate all the sequences before selection. In addition, the rules derived from the answers to the questions become difficult if the product complexity increases. For the heuristics methods, the drawback is the complexity involved in selecting an initial feasible assembly plan.

## 3. Enumeration of Assembly Sequence Space

An assembly sequence design space is defined as a set of all feasible sequences for a given product or family of products. The feasible sequences are obtained by applying constraints, such as precedence relations, imposing subassemblies, etc., on assembly sequence spaces. In this section, an algorithm and associated mathematical tools, to determine the unconstrained assembly sequence design space is presented. The problem is defined as: Given a list of components $V=\{v_1, v_2, \ldots, v_n\}$ for a product or new product family member, determine an assembly sequence space that models all possible assembly sequences for the component set V.

Development of a model to represent assembly process and assembly plant is discussed in Section 3.1. Individual assembly sequences are represented using series-reduced planted tree (SRPT). Mathematical models and associated algorithm to enumerate the assembly sequence design space are discussed in Sections 3.2 and 3.3.

### 3.1 Series-reduced Planted Tree

To enumerate the assembly sequence space a new representation of assembly process called series-reduced planted tree was introduced by Siddique [24]. If $C$ is taken to be collection of components in the product and $|C|=n$, then assembly sequence is a SRPT with $n+1$ labeled leaves. Vertices of degree one is called leaves or endpoints and vertices more than degree one are called assembly workstations. If two or more components are assembled together they are considered as a single component or a sub-assembly, which can be assembled to other components or subassemblies. The unassembled components are considered as leaves of the tree. The finished product is represented by point adjacent to the root. The advantage of this representation is that each SRPT refers to an assembly process. The SRPT representation is extended to capture product family assembly plant information. The leaf node labels of the tree are underlined to distinguish between platform and optional components. If a product family member does not have an optional component then we assume that no activity will be performed at that workstation for the product. For new products with options not included in the current assembly plant, we try to apply the concept of late point differentiation to assemble the optional components in the assembly process [25].

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

### 3.2 Enumeration of Assembly Sequence Space

The assembly sequence space is combinatorial in nature and can be enumerated using recursion. If $T$ is an assembly sequence represented as SRPT. The branches of the tree can consist of individual components and subassemblies. Each subassembly is an assembly space. Consider an assembly sequence of a product with a set of components $V=\{a, b, c, d\}$ divided into two branches such that one branch has component $a$ and the other branch has components $b$, $c$, and $d$ (Figure 1(a)). Each branch is a subassembly and branch with components $a$ has only one sequence. The branch with components $b$, $c$, and $d$ is a subassembly with more than one sequence. The branch with three components has four possible sequences (Figure 1(b)).

This recursive nature leads to enumeration of assembly sequence space. To solve the enumeration of assembly sequence space mathematically set partitions are applied. Therefore, enumeration of assembly sequence space is described as partitioning of set of components recursively. The set of components of a product are partitioned into subsets such that, each subset of two or more components is considered as a branch and individual components are considered as leaves. The subset with more than two components is further partitioned. Consider $V=\{v_1,v_2,\ldots,v_n\}$ be a set of $n$ components then the unconstraint assembly sequence space is formally represented as: $Seq_V = Seq\{v_1,v_2,\ldots,v_n\}$.

### 3.3 Algorithms to Generate Assembly Sequence Space

Before developing the algorithm to generate the unconstrained assembly sequence space, several definitions need to be presented. A set of distinct objects that can be divided into a number of ways is called a set partition. The number of ways, which a set of distinct objects can be partitioned, is called Bell number. Bell number for a set with $n$ objects, $B(n)$, is the sum of the number of ways that it can be partitioned into 1,2,3, . . . ,$n$ subsets. The number of ways that a set of size $n$ can be partitioned into $k$ subsets is called Stirling number of the second kind and is denoted as $S_2(n, k)$. They satisfy the following recurrence relation, which forms the basis of recursive algorithms for generating them.

$$n_k = k\binom{n-1}{k} + \binom{n-1}{k-1} \tag{1}$$

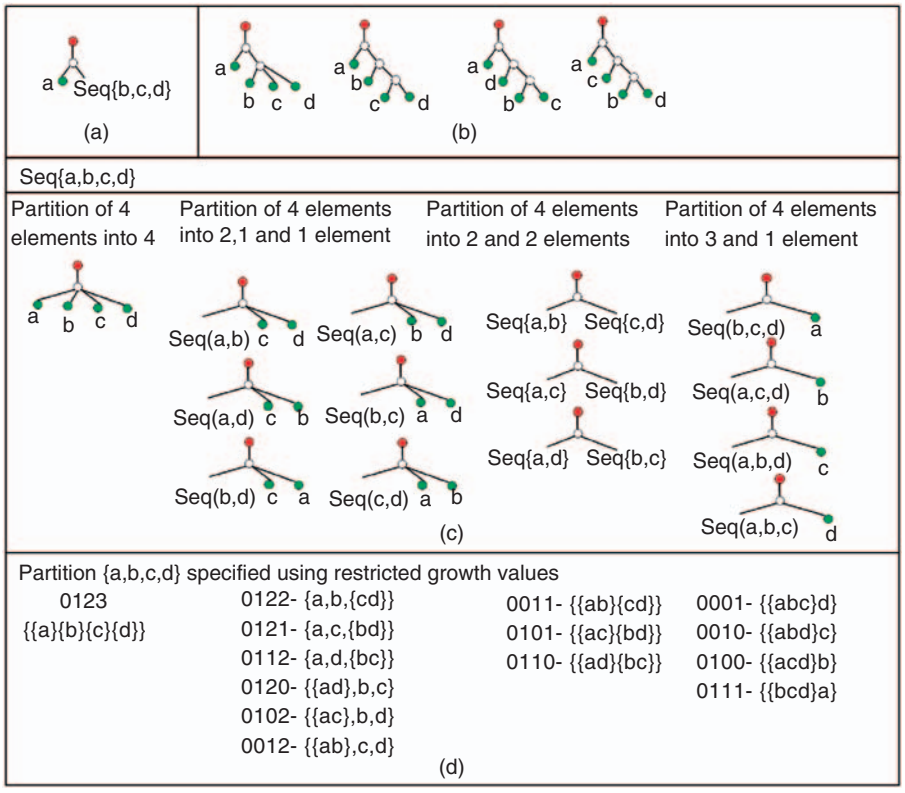



**Figure 1.** Recursive nature of assembly sequence space: (a) components divided into two branches; (b) four feasible assembly sequence for four components divided into two branches of one and three components; (c-d) set partitions obtained from restricted growth strings for four components.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

In general Bell number $B(n)$ is the sum of Stirling numbers from 1 to $n$. The $n$-th Bell number can be computed using the formula:

$$B(n) = \sum_{k=1}^{n} S_2(n,k) \tag{2}$$

For example the set {1, 2, 3} can be partitioned into three subsets in one way: {{1},{2},{3}}; into two subsets in three ways {{1,2},{3}},{{1,3},{2}}, and {{1},{2,3}}; and into one subset in one way: {{1,2,3}}. The Bell number is $B(3) = S_2(3,1) + S_2(3,2) + S_2(3,3) = 1 + 3 + 1 = 5$. Restricted growth function is used to generate the partitions themselves. The restricted growth array is a set of zero-based indices specifying to which partition each element of the set belongs.

For a set of size 4, for example $RG = [0,0,0,0]$, defines all elements belonging to one partition, which is {1,2,3,4}, $RG = [0,1,1,0]$ represents {1,4}, {2,3}, which specifies 1 and 4 belonging to the first partition and 2 and 3 belonging to second partition. The $RG$ array values have the characteristic that for each $i$, $RG[i] <= 1 + \max(RG[0], RG[1], \ldots, RG[i-1])$. A set partition of the set $[n] = \{1, 2, \ldots, n\}$ is a collection $B_0$, $B_1$, $B_2, \ldots, B_j$ of disjoint subsets of $[n]$ where $B_0 \cup B_1 \cup B_2 \cdots \cup B_j = [n]$ and $B_0 \cap B_1 \cap B_2 \cdots B_j = \varnothing$. Each $B_i$ is considered as a block. A restricted growth string (or $RG$ string) is a string $a[1, \ldots, n]$ where $a[i]$ is the block in which element $i$ occurs. Restricted growth strings are often called restricted growth functions.

The restrictions on assembly sequence space are

- The assembly plan can be nonlinear, that is more than one component can be moved at a time.
- Assembly plan is monotone that every operation makes all the moved parts reach goal positions and no operation separates the parts already assembled.
- Assembly plan can be sequential, such that each operation involves moving a set of parts along a common trajectory.

Algorithm 1, used to generate restricted growth values, was developed by Orlov [26]:

Given: Set of components of size $n$.
Step 1: create two array variables $k$ and $m$ of size $n$ and initialize them to zero.
Step 2: Each value of array $m$ is calculated from equation $m_i = \max\{k_0, \ldots, k_i\}$. Each value of array $m$ is outcome of above equation throughout the algorithm.
Step 3: Calculate the bell number from Stirling number of second kind.
Step 4: The restricted growth values are generated and stored in array $k$. Repeat this step from $i = 1$ to (Bell number $-1$)

```
for i = n−1 to 1 do
if k_i ≤ m_{i−1} then k_i ← k_i + 1and m_i ←max(m_i, k_i)
if ((i + 1)≤n)
   for j = i + 1 to n−1 do
      k_j ←k_0 and m_j ←m_i
Return
```

Consider a set of four components as $V = \{a, b, c, d\}$, the restricted growth is shown in Figure 1(d).

Algorithm 1 generates partitions for set of components. However, for enumerating assembly sequence space the partitioned subsets need to be further partitioned until all the components are divided into individual elements. Algorithm 2, used for recursive partitioning of sets is:

Given: A set of components of size $n$.
Step 1: If number of components = 2, then there is only one sequence. If number of components = 1 then it is an individual component.
Step 2: If number of components > 2, then,
Step 3: Generate the restricted growth values using Algorithm 1.
Step 4: After obtaining, the restricted growth values collect the elements, which fall into the respective subsets. Again, repeat the steps 1–4. Each time when the sequence is determined for the subsets add it as a subset to main set.
Step 5: Print sequence according to values stored in subsets.

Consider a set of four components $V = \{a, b, c, d\}$, all set partitions and restricted growth function are shown in Figure 1(c) and (d).

## 4. Constraints on Assembly Space

Since the assembly space is combinatorial in nature, enumeration of the design space becomes difficult, if not impossible, as numbers of components increase. In order to address this issue, constraints are applied on the assembly space to ensure that only feasible assembly sequences are generated, which is different than generating and then checking for feasibility. Constraints on the assembly space are based on work done by Nugen [27] and expanded in Wilmes and Siddique [25].

### 4.1 Type I Constraints

Type I constraints on an assembly sequence space is a situation where the basic components of a subset are completely specified along with the sequence among the components. The effect of Type I constraint can be imagined as specifying a sub-tree of a SRPT.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

#### 4.1.1 MATHEMATICAL MODEL FOR TYPE I CONSTRAINT

Formally, consider a set of $n$ components $V = \{v_1, v_2, \ldots, v_n\}$ then the unconstrained assembly sequence space can be represented as $Seq_V$. If $k$ is the number of components satisfying Type I constraint and $t$ denotes the super-node representing Type I constraint, then the assembly sequence space satisfying Type I constraint is formally represented as:

$$Seq_V\left(C_T^I\right) = Seq_{V-K+t} \tag{3}$$

*Multiple Type I Constraints*: Suppose that there are $m$ Type I constraints $T_1, T_2, \ldots, T_m$ with $K_1, K_2, \ldots, K_k$ different set of components satisfying each Type I constraints, that is $T_1 \cap T_2 = T_1 \cap T_3 = \cdots = T_{m-1} \cap T_m = \emptyset$, then formal representation of multiple constraints is:

$$Seq_V\left(C_{T_1}^I + C_{T_2}^I + \cdots + C_{T_m}^I\right) = Seq_{V-K_1+t_1-K_2+t_2-\cdots K_k+t_k} \tag{4}$$

*Nested Type I Constraints*: Another way of arranging Type I constraints is by nesting. With Type I constraint, nested constraints are redundant. For example if $T_1 \subseteq T_2$ are both Type I, then $T_1$ is a redundant constraint. We can solve Type I constraint by just applying $T_2$:

$$Seq_V\left(C_{T_2}^I\right) = Seq_{V-K_2+t_2} \tag{5}$$

#### 4.1.2 ALGORITHM 3 TO APPLY TYPE I CONSTRAINTS

Algorithm 3 identifies the sequence among the components in Type I constraints specifying them as a sub-tree. The Type I constraints are represented as super-nodes (components) to generate the assembly sequence space. Applying Type I constraints reduce the size of the assembly space.

Given: A product with $n$ components stored in array $V = \{v_1, v_2, \ldots, v_n\}$. Type I constraints $T_1, T_2, \ldots, T_m$ having $k_1, k_2, \ldots, k_m$ elements showing the sequence among the components. When Type I constraints are defined each Type I constraint is given a new component number as $e_{i(i=1,\ldots,m)}$.

Step 1: Check for multiple constraints and nested constraints. If there are multiple constraints then check for $T_1 \cap T_2 = T_1 \cap T_3 = \cdots = T_{m-1} \cap T_m = \emptyset$. If the condition is not equal to NULL (empty set), then the constraints are invalid constraints. If there are any nested constraints, for example if $T_1 \subseteq T_2$, then eliminate $T_1$ and just apply $T_2$.

Step 2: Solve for Type I constraint ($T_i$) where $i = 1,2,\ldots,m$ having $k_i$ number of elements. Define all the elements of $T_i$ constraint by specifying the sequence among the components. Give name of new component number as $e_i$.

Step 3: Check whether $T_i$ is subset of array $V$. Check whether each element of $T_i$ is contained in array $V$.

Step 4: Next check for the first element of $T_1$ in the array $V$, if it is found at position '$j$' in main array $V$, then replace the subset $T_1$, with $V[i]$ at its respective position. Now change the positions of all the components in $V[\,]$ from $j+1$ to $(n-k_i)$ or depending upon the positions of the elements eliminated from the array $V$, which are contained in $T_1$. Delete the positions from $n-k_i$ to $n-1$. The total number of components in array $V$ are $n-k+1$.

Step 5: Check for the next Type I constraint and repeat all the steps from 1 to 4.

Step 6: After all the Type I constraints are defined and specified as super nodes the assembly sequence space is determined by using the Algorithm 2.

Step 7: After enumerating the assembly space the new component numbers given for each Type I constraint are replaced with their respective components and completely defined sequence among the components of each Type I constraint.

### 4.2 Type II Constraints

The effect of Type II constraints can be considered as specifying a set of subassembly sequences for a given set of components on an assembly sequence space. The effect of Type II constraint can be imagined as specifying different sub-trees for a same set of components of a SRPT.

#### 4.2.1 MATHEMATICAL MODEL FOR TYPE II CONSTRAINT

If $v$ is the entire set of components for a given product and $Q$ is the set of components representing Type II constraint. The sequence space among the components is denoted by $Seq_Q$. Then each sequence of $Q$ is considered as Type I constraint, which can be represented as a super-node to determine the assembly sequence space. The assembly sequence satisfying Type II constraint is:

$$Seq_V\left(C_Q^{II}\right) = \left( Seq_V\left(C_{q_i}^I\right) \middle| (q_1, \ldots, q_i) \in Seq_Q \text{ and } i = 1,2,\ldots,R_{|Q|} \right) \tag{6}$$

*Multiple Type II Constraints:* Two types of situation emerge when more than 1 Type II constraints are applied. These cases are defined as multiple and nested. In the multiple case, the constrained subsets are disjoint. In the second case constrained subsets are nested, one subset is contained in another subset.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

If there are $m$ many multiple Type II constraints then they have to satisfy $Q_1 \cap Q_2 = Q_1 \cap Q_3 = \cdots = Q_{m-1} \cap Q_m = \emptyset$, and the formal representation is:

$$Seq_V\left(C^{II}_{Q_1} + C^{II}_{Q_2} + \cdots + C^{II}_{Q_m}\right) = \left( Seq_V\left(C^{I}_{q_1 i} + C^{I}_{q_2 j} + \cdots + C^{I}_{q_m k}\right) \middle| \begin{array}{l} q_{1i} \in Seq_{Q1} \text{ and } i = 1,2,\ldots,R_{|Q_1|} \\ q_{2j} \in Seq_{Q2} \text{ and } j = 1,2,\ldots,R_{|Q_2|} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ q_{mk} \in Seq_{Qm} \text{ and } k = 1,2,\ldots,R_{|Q_m|} \end{array} \right) \quad (7)$$

If there are $m$ many nested Type II constraints then,

$$Seq_V\left(C^{II}_{Q_1} C^{II}_{Q_2} \ldots C^{II}_{Q_m}\right) = \left( Seq_V\left(C^{I}_{q_1 i}\right) \middle| \begin{array}{l} q_{1i} \in Seq_{V-Q_1+q_1}(C^{I}_{q_2 j}) \text{ and } i = 1,2,\ldots,R_{n-|Q_1|+1} \\ q_{2j} \in Seq_{Q_1-Q_2+q_2} \text{ and } j = 1,2,\ldots,R_{|Q_1|-|Q_2|+1} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ q_{m_k} \in Seq_{Qm} \text{ and } k = 1,2,\ldots,R_{|Q_m|} \end{array} \right) \quad (8)$$

where: $V \supset Q_1 \supset Q_2 \supset \cdots \supset Q_m$.

#### 4.2.2 ALGORITHM 4 TO APPLY TYPE II CONSTRAINTS

In Algorithm 4, Type II constraint is represented as super-node, which is used as a component in generating assembly sequence space. Each super-node corresponding with Type II constraint represents an assembly space, with components contained in the constraint. Once the initial assembly space with super-node is generated, each super-node is replaced by its corresponding sub-assembly space.

Given: Given $n$ components of a product stored in an array $V = \{v_1, v_2, \ldots, v_n\}$. Given all Type II constraints $Q_1, Q_2, \ldots, Q_m$ having $x_1, x_2, \ldots, x_m$ elements with out showing any specified sequence among the components. When Type II constraints are defined each Type II constraint is given a new component number as $q_{i(i=1,\ldots,m)}$.

Step 1: Check for multiple constraints and nested constraints. If the constraints are multiple constraints then check for $Q_1 \cap Q_2 = Q_1 \cap Q_3 = \cdots = Q_{m-1} \cap Q_m = \emptyset$. If the condition is not equal to NULL then the constraints are invalid constraints. Then ask the user to give a valid constraint. If there are any nested constraints, for example if $Q_1 \subseteq Q_2$, then $Q_1$ is a subset of $Q_2$. Therefore the elements of subsets are enumerated first and the sub-assembly space is represented as super-node.

Step 2: Solve for each Type II constraint ($Q_i$) where $i = 1, 2, \ldots, m$ having $x_i$ number of elements. Define all the elements of $Q_i$ constraint giving name of each Type II constraint as super node $q_i$.

Step 3: Check whether $Q_i$ is subset of array $V$. Check whether each element of $Q_i$ is contained in array $V$.

Step 4: Next check for the first element of $Q_1$ in the main array $V$, if it is found at Position '$j$' then replace the subset $Q_1$, in $V$ at its respective position. Now change the positions of all the components in $V$ from $j+1$ to $(n - y_i)$ or depending upon the positions of the elements eliminated from the main array, which are contained in $Q_1$. The total number of components in array $V$ is $n - y_i + 1$.

Step 5: Check if $Q_i$ has any subsets, which are already defined under Type I or Type II constraints. If it has then replace them by following the procedure in Step 1, 2, 3, & 4 assuming $Q_i$ as main array.

Step 6: Check for the next Type II constraint and repeat from Step 1 to 5.

Step 7: The sequence space among the components of each Type II constraint is generated using Algorithm 2 and replaced by new components numbers (super-nodes) in array $V$.

Step 8: The final assembly sequence space is generated by using the new component numbers of Type II constraints. The component number of Type II constraint in each sequence of final space is replaced with corresponding sub-assembly space.

### 4.3 Type III Constraints

Type III constraints are used to specify precedence relationships among components in the assembly. Consider a product with four components $a$, $b$, $c$, and $d$. If component $a$ has to be assembled after $d$, then we use the notation $a > d$. The precedence relationship does not mean that component $a$ is immediately after $d$, an assembly sequence where $c$ and $d$ is first assembled, then $b$ is assembled, with $a$ added at the last workstation will be a valid assembly sequence. To enumerate the assembly sequence space with Type III constraints we use a concept of 'growth retardation.' Since the assembly sequence space is recursive in nature, if a partition is infeasible then the growth of that partition is

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

stopped. So, in the four component example with a Type III constraint, $a>d$, the recursion is not applied to partitions that do not meet the constraint. To solve the Type III constraints mathematically we need to calculate the level numbers. A level number for a given component in a sequence is defined as number of vertices between the component and the root. Algorithm 5 to check Type III constraints is:

Given: Given an input strings of the form $\{v_1, v_2, \{v_{k-1}, v_k\}, \ldots, v_n\}$ and array Temp given a Type III constraint of type $x_1\ x_2 > x_3, x_4$.
Step 1: Each component on both sides of precedence symbol has to be checked.
Step 2: Calculate the level numbers for the given input string. Store the level numbers with their respective positions corresponding to component numbers in array Temp.
Step 3: Solve for the level numbers of given Type III constraint on both sides of the precedence symbol. Obtain the level numbers of components from array obtained in Step 3 by matching the component number with the component numbers in Type III constraint string. If a component number in Type III constraint in not contained in the string or vice versa do not check for that component number.
Step 4: Check if level number on right-hand side of precedence symbol is greater than the level number on left-hand side of the precedence symbol. If this condition is satisfied then the Type III constraint is satisfied for that given input string. If not the growth of partition is stopped or the given string is not considered as a valid string.
Step 5: For other precedence constraints repeat from Steps 2 to 4.

To generate feasible sequences, first the Type I constraints are applied by collapsing the subsets into super-nodes. Then partially apply Type II constraints by collapsing the subsets into super-nodes. The associated sub-spaces are considered later. Organize the Type III constraints. Enumerate the subspaces associated with each Type II constraint based on Type III constraints. Enumerate the sequence space for the new set of components with super-nodes obtained from Type I and Type II constraints. The final assembly sequence space is determined by considering combinations of each Type II constraint.

## 5. Assembly Process Selection by Measuring Process and Plant Similarity

One of the focuses of this article is to select an assembly process that will increase the utilization of current assembly plant resources. The similarity between the assembly process and a given assembly plant is measured by the difference in the path length among the common components in assembly process and plant. The path length difference indicates the assembly sequence deviation from assembly plant sequence. The path length for a given component $q$ in assembly process or assembly plant is calculated as number of vertices denoted by $m$ in between root and component $q$. To calculate the change in assembly sequence between the assembly process and plant, the difference of path lengths for each component in the assembly process and assembly plant is calculated. Let $x$ be the path length of a given component $q$ in assembly process and $y$ is the path length of the component $q$ in assembly plant sequence then the difference in path length is absolute value $|x-y|$. The path lengths of optional components are not considered when measuring the change in assembly sequence of process and plant:

Given: All feasible assembly sequences and assembly plant sequence.
Step 1: Calculate the path lengths from the root for each component in assembly plant sequence.
Step 2: Calculate the path lengths from the root for each component in assembly process for the first feasible assembly sequence.
Step 3: Measure the difference of path lengths (absolute value) for each components in assembly process and plant. Do not measure for optional components.
Step 4: Calculate the total difference of this assembly process from assembly plant by adding all the differences for each component.
Step 5: Repeat Steps 2, 3, and 4 for each assembly process.
Step 6: Compare the total difference of all the assembly sequences and select the assembly process(es) with minimum total difference.

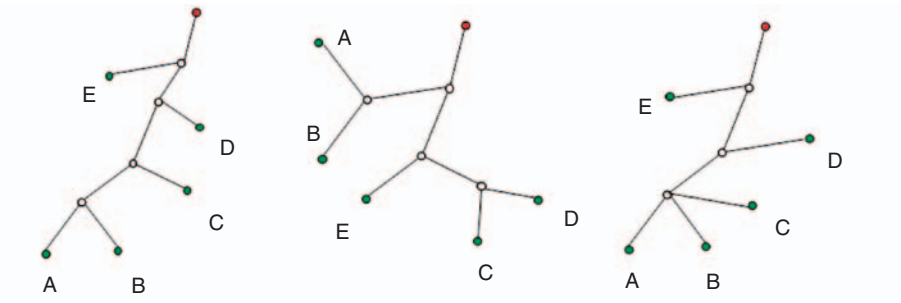
Consider the example with five components, as shown in Figure 2. The total path length difference is 2 for assembly process 1 (Figure 2(a)). The path lengths calculated for assembly process 2 (Figure 2(b)) are 2, 2, 3, 3, 3 and differences are 1, 1, 0, 1, 2. The total difference for assembly process 2 is 5. Hence we select assembly process 1. A detailed comparison is shown in Figure 2(d).

## 6. Implementation

### 6.1 Generation of Assembly Space

A software program coded in Visual C++ has been developed to automatically generate assembly sequence space. A flow chart gives overview of the generation of feasible assembly sequences and selection of assembly plant that minimizes existing plant modification cost is shown in Figure 3.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016




| Component | Assembly | | Assembly plant | Absolute difference | |
|---|---|---|---|---|---|
| | Process 1 | Process 2 | | Process 1 | Process 2 |
| A | 4 | 2 | 3 | 1 | 1 |
| B | 4 | 2 | 3 | 1 | 1 |
| C | 3 | 3 | 3 | 0 | 0 |
| D | 2 | 3 | 2 | 0 | 1 |
| E | 1 | 2 | 1 | 0 | 1 |
| Total difference | | | | 2 | 4 |

(d)

**Figure 2.** (a) Assembly process 1, (b) assembly process 2, (c) existing assembly plant sequence, (d) comparison of feasible assembly process and plant.

The user input is given in text file. Generation of the space starts with reading the input file. The first step is to declare all the variables, pointers, elements for Type I, Type II, and Type III constraints. Next, the number of components is initialized by reading the first line from the file. The names of the components are also added to the corresponding component numbers. After reading the component names an element of Type II constraint is created, which represents the main space element. The main space element contains list of all components, number of components and a component number given to the main space element as an increment of the number of components. Next step is to read the constraints.

When the first Type I constraint is read, an element of Type I constraint is created and the associated component numbers are added to component list by reading each component number and the component name is given as the user specified one. Each Type I constraint is given a new component number. Each time a new element is created at the start of new hierarchy. After all hierarchies are specified the sequence string representing the Type I constraint is written to a file. For Type II constraints are read, the associated components are stored in the component list of Type II constraint. When Type III constraints are read for each Type III constraint an object of type supercon3 is created. The components on left side of the precedence symbol are stored in 'before' and on the right side are stored in 'after.' The next step is to organize the Type I and Type II constraints. First all the Type II constraints are ordered according to the number of components in each constraint. After that Type II constraints are organized according to the hierarchy.

A function then organizes the constraints according to the component number of the constraint. The parent component list and child component list are compared. If all the components are found in parent component list the child element component number is replaced with the components present in the parent component list. Therefore, if any Type I and Type II constraints are subsets of other Type II constraint or main space. Then they are replaced with their component numbers.

Type III constraints are organized by finding the appropriate element with each individual Type III constraints. Each component on both sides of precedence symbol is compared. Each component is checked for the appropriate element and if it is found in the appropriate element then the component is added to the list of Type III constraint of the element.

The assembly sequence with all three types of constraints by first checking whether the element is a Type II constraint then only it generates the space. Next it checks whether any child elements exist for the current element. If it has any children it calls the same function to generate the space for children. It also checks whether the current element is a Type I constraint, if it is a Type I constraint then it checks for the next element. A function is then invoked to generate the space for the current element. This method generates the sequence space and writes them into a file. For each Type II constraint the associated space is written to a separate file. The generation of the space starts from the bottom for the hierarchy by generating sub spaces.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

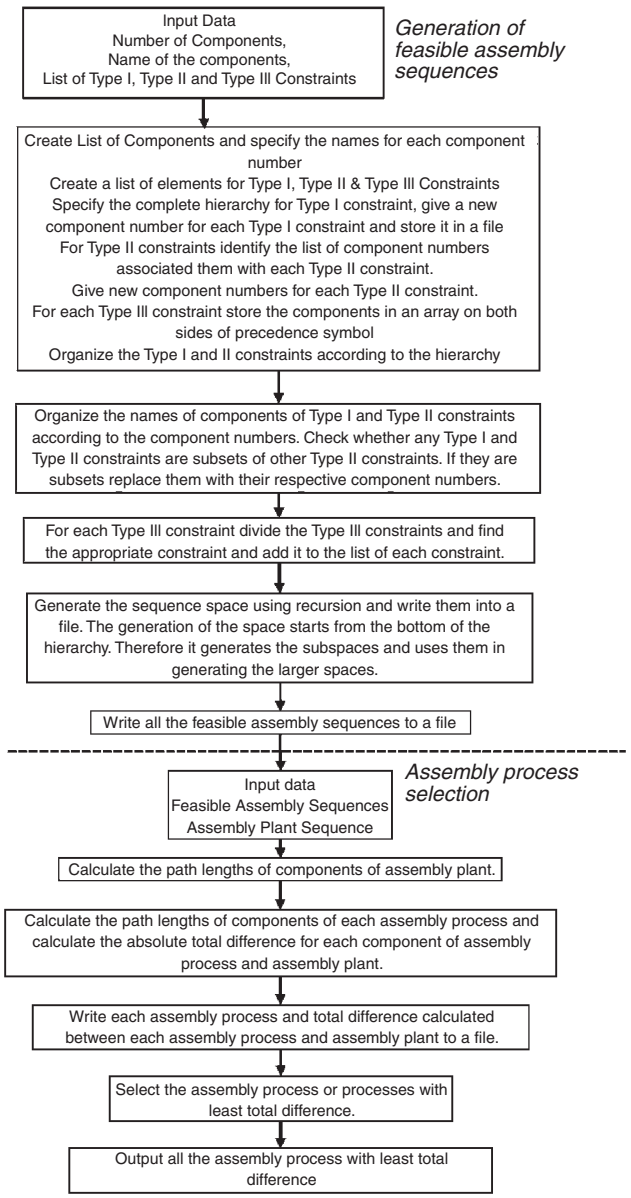



**Figure 3.** Flowchart of overall approach.

### 6.2 Selecting Assembly Process that Minimizes Existing Plant Modification Cost

A class called component is created with members functions that reads the feasible assembly sequences and then determines the total absolute difference of path lengths for all components in assembly process and assembly plant. The main function counts the number of components for any given assembly process or assembly plant. Another function calculates the path lengths of component numbers of assembly plant and each assembly process. By calculating the path lengths the total path length difference is calculated for all components of assembly process and plant.

## 7. Automotive Underbody Front Structure Product Family

Automotive underbody front structure is an important part of automotive underbody called chassis. The front structure of an automotive underbody has a number of varieties, to meet the requirements of different automotive models. Different functions are used to satisfy needs of different market segments. In this case study the automotive underbody front structure product family is used to demonstrate the generation of assembly sequence and to perform trade-off studies in order to utilize existing assembly plant resources.

Imagine a manufacturing company already has an assembly plant that manufactures several automotive underbody front structure varieties. A new automotive underbody front structure has been designed with few added options and the manufacturer wants to determine if the current assembly plant can be used to manufacture the new underbody front structure without any change. If not, then determine the underbody front structure assembly sequence that will require minimum modification to the existing assembly plant. The steps involved are presented next.

*Step 1: Identify the assembly process constraints for the new automotive underbody front structure product family member*: The new automotive underbody front structure model being designed by the manufacturer with a new option added, which requires assembling of five new components: Support Battery Tray Front, Bracket Fan Shrout Upper, Plate Assembly Engine Front Mounting, Rein. Floor Side Member Inner RR(LH), and Rein. Floor Side Member Inner RR(RH) to the underbody front structure.

The constraints on the assembly process for the new automotive underbody front structure are shown in Figure 4 (optional components are underlined). The constraints use the component numbers shown in component list. For example, $T_2 = \{29, 30, 31, 32\}$, which means that Radiator Support Upper Reinforcement, Radiator Support Upper, Bracket Rad Mounting, Hood Lock Reinforcement are assembled first at same workstation. This is a Type I constraint.

*Step 2: Generate the feasible assembly sequences using the software tool*. Using the developed application, the feasible assembly sequences are generated. The component names and the list of all three types of constraints are given as input in a text file. The user's input file contains: number of components given as size, names of components, and the three types of constraints. The feasible assembly sequences are generated and saved in a text file. There are 624 feasible assembly sequences for the new automotive underbody front structure.

*Step 3: Assembly process selection by comparing feasible sequences with the existing assembly plant*.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

Component list:

1. Front Fender Apron Upper Brace (RH)
2. Front Suspension Mounting Housing (RH)
3. Bracket Radiator Overflow Cont
4. Front Fender Apron Upper Inner Reinf (RH)
5. Front Fender Apron Extension (RH)
6. Engine Mtg Plate Front (RH)
7. Floor Side Inner Member Reinf (RH)
8. Floor Side Inner Front Member Assembly (RH)
9. Bracket Engine Mounting Rear
10. Brake Line Bracket Front (RH)
11. Filler Front Floor Side Member Front (RH)
12. Reinf Engine Rear Support Bracket
13. Front Suspension Housing Upper Reinf (RH)
14. Front Suspension Mounting Housing (LH)
15. A/C L Mounting Brkt Center (LH)
16. Front Fender Apron Upper Brace (LH)
17. Floor Side Inner Member Reinf(LH)
18. Floor Side Inner Front Member Assembly (LH)
19. Engine Mtg Plate Front (LH)
20. A/C L Mounting Brkt RR
21. Battery Support Bracket Inner
22. Front Fender Apron Extension (LH)
23. Front Fender Apron Upper Inner Reinf(LH)
24. Brake Line Bracket Front
25. Filler Front Floor Side Member Front (LH)
26. A/C L Mounting Brkt Center (LH)
27. Front Suspension Housing Upper Reinf (LH)
28. Front Side Member Extension
29. Radiator Support Upper Reinf
30. Radiator Support Upper
31. Bracket Radiator Mounting
32. Hood Lock Reinf
33. Front Cross Member Lower Reinf
34. Front Cross Member Lowerfill
35. Front Tow Hook Bracket Assembly
36. Head Lamp Mounting Panel Reinf
37. Front Fender Apron Reinf (RH)
38. Front Fender Apron Reinf (LH)
39. Reinf Front Side Member Extension
40. Bracket Accel Padel Stop
41. Dash Panel
42. Reinf Dash Panel Brake Master Cyl
43. Body Front Cross Lower Member
44. Support Battery Tray Front
45. Brkt Fan Shrout Upper
46. Plate Assembly Engine Front Mounting
47. Reinf Floor Side Member Inner RR (LH)
48. Reinf Floor Side Member Inner RR (RH)

*Type I*

$T_1$ = {,13,{,{,4,5,},{,10,11,12,},{,9,{,3,{,1,2,},},{,6,7,8,} ,},},}
$T_2$ = {,29,30,31,32,}
$T_3$ = {,33,34,35,}
$T_4$ = {,28,46,47,48,}
$T_5$= {,36,37,38,45,}
$T_6$= {,27,{,{,22,23,},{,{,24,25,},26,},{,{,17,18,19,}, {,{,21,20,}, {,{,14,44,},15,16,},},},},}

*Type III*

$Q_1$ = {,40,41,42,28,39,46,47,48,}
$Q_2$ = {,29,30,31,32,33,34,35,36,37,38,45,}
$Q_3$ = {,33,34,35,}
$Q_4$ = {,14,15,16,17,18,19,20,21,22,23,24,25,26,27,44,}
$Q_5$ = {,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22, 23,24,25,26,27,29,30,31,32,33,34,35,36,37,38,44,45,}

*Type III*

40,41,42,>,28,39;
43,>,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16, 17,18,19,20,21,22, 23,24,25,26,27,29,30,31,32,33,34,35,36,37,38,44,45;

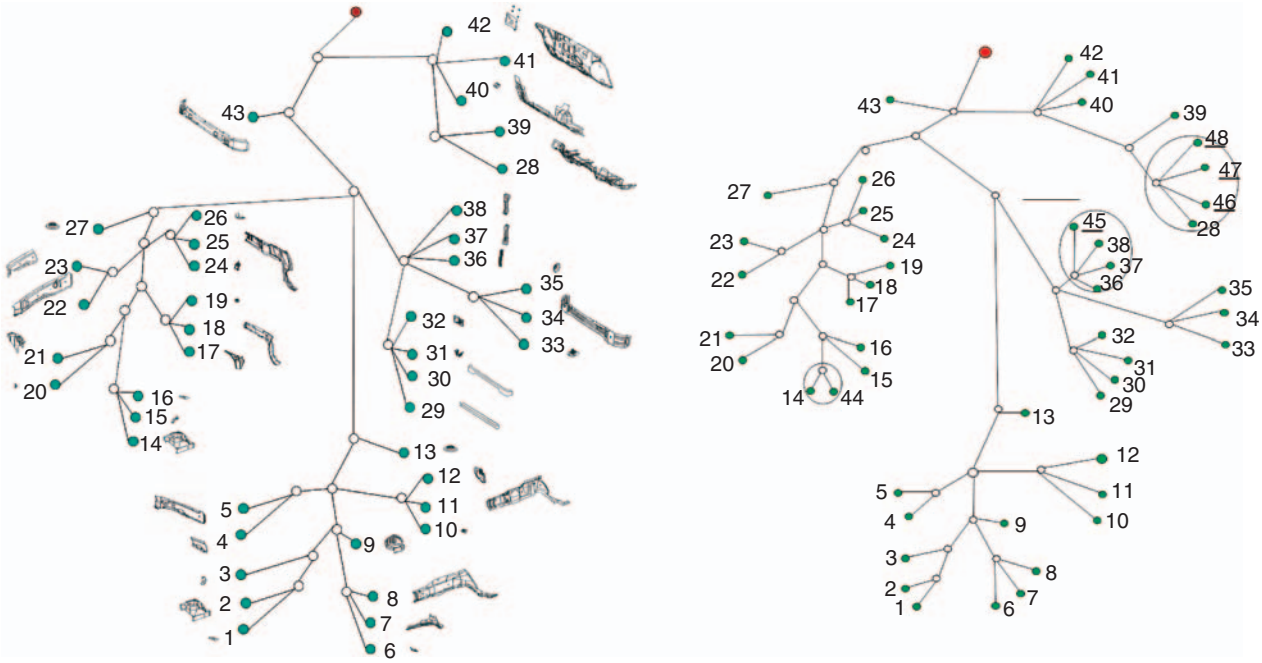



**Figure 4.** Name of components and assembly constraints for new automotive underbody front structure family member: (a) existing product family assembly plant model, (b) modified product family assembly plant sequence to incorporate the new automotive underbody front structure product family member.

The feasible assembly sequences are compared with the existing assembly plant sequence. The assembly plant representation for the existing automotive underbody front structure is modeled using SRPT Representation (Figure 4(a)). The assembly plant sequence can be written as:

{{43,{{27,{{22,23,}{24,25,26,}{{17,18,29,}{{20,21,}{14,15,16,}}}}}}{13,{{4,5,}{10,11,12,}{9,{3,{1,2,}}{6,7,8,}}}}{36,37,38,{29,30,31,32,}{33,34,35,}}}}{40,41,42,{39,28,}}}. The numbers in the string corresponds to automotive underbody front structure components, as shown in Figure 4(a).

The feasible sequences are compared with the existing assembly plant sequence and the absolute difference in path lengths for each component of each assembly process is calculated. All the differences are added to calculate the total difference for each assembly process. The assembly process with the least total difference is selected and displayed in the output text file (Figure 5).

For the automotive front structure, there is only one assembly process which has the least total difference of '8.' There are no assembly processes with total difference '9' and '10.' There are three assembly processes with

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016




**Figure 5.** Output file showing the final assembly process.

total difference '11.' The assembly process with total difference of '8' will be selected to make changes in the existing assembly plant sequence, because it represents the least modification to the existing assembly plant.

## 8. Concluding Remarks

In this article an approach to generate assembly sequence design spaces has been presented. The SRPT was developed to represent assembly plant and assembly process sequence. The hierarchical nature of this representation leads to recursion, which consists of components assembled as subassemblies. This recursive nature is used to recursively partitioning the set of components to enumerate entire assembly space. The algorithm to generate assembly sequence space was presented.

A methodology to solve the effects of three types of constraints to generate only feasible assembly sequence spaces was presented. The Type III constraint uses the concept of growth retardation such that only feasible assembly sequences are retained. An algorithm was developed to automatically solve the effect of all constraints and to generate only feasible sequences by eliminating the infeasible ones.

A unique approach to compare each feasible assembly sequences with assembly plant sequence to determine a product family assembly process that minimizes the existing assembly plant modification cost has been developed. Each component path distances from the root were measured and compared for assembly process and plant sequence. A computer application, which utilizes the above concepts to determine the feasible assembly sequences and to select an assembly process that increase utilization of existing assembly plant resources, was presented. The method was applied to automotive underbody front structure product family to estimate the change for existing assembly plant for a new product family member. Some of the limitations of this research include:

- This research assumes that information provided by the user about components and assembly constraints are valid and correct.
- The software tool developed selects assembly process based on change in sequence of assembly process and assembly plant. A tool which considers information about moving cost, workstation attributes and time etc., is necessary.
- The method developed to generate assembly sequence space does not consider information about geometric checks, mating relationships and other aspects.
- The assembly process generated in this article is presented in the form of strings using curly braces. The user has to determine assembly tree from this sequence. A visual representation of the tree can help users understand the assembly process.

## References

1. Bower, J.L. and Hout, T. (1988). Fast Cycle Capability for Competitive Power, *Harvard Business Review*, **66**(Nov–Dec): 110–118.
2. McDermott, C.M. and Stock, G.N. (1994). The Use of Common Parts and Designs in High-tech Industries: A Strategic Approach, *Production and Inventory Management Journal*, **35**(3): 65–68.
3. Wheelwright, S.C. and Clark, K.B. (1992). *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency and Quality*, New York: The Free Press.
4. Martinez, M.T., Favrel, J. and Ghodous, P. (2000). Product Family Manufacturing Plan Generation and Classification, *Concurrent Engineering: Research and Applications*, **8**: 12–23
5. Liverani, A., Amati, G. and Caligiana, G. (2004). A CAD-augmented Reality Integrated Environment for Assembly Sequence Check and Interactive Validation, *Concurrent Engineering: Research and Applications*, **12**: 67–77.
6. Giudice, F., Ballisteri, F. and Risitano, G. (2009) Concurrent Design Method Based on DFMA–FEA Integrated Approach, *Concurrent Engineering: Research and Applications*, **17**: 183–202.
7. Gottipolu, R.B. and Ghosh, K. (1997). Representation and Selection of Assembly Sequences in Computer-Aided Assembly Process Planning, *International Journal of Product Research*, **35**(12): 3447–3465.
8. Grewal, S. and Choi, C.K. (2005). An Integrated Approach to Manufacturing Process Design and Costing, *Concurrent Engineering: Research and Applications*, **13**: 199–207.
9. Li, Y., Xue, D. and Gu, P. (2008). Design for Product Adaptability, *Concurrent Engineering: Research and Applications*, **16**: 221–232.
10. Chan, S.C.F., Soo, S.M.K. and Yu, K.M. (2006). Customer-driven Collaborative Product Assembler for Internet-based Commerce, *Concurrent Engineering: Research and Applications*, **14**: 99–109.
11. De-Fazio, T.L. and Whitney, D.E. (1987). Simplified Generation of all Mechanical Assembly Sequences, *IEEE Journal of Robotics and Automation*, RA-**3**(6): 640–658.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016

12. Bourjault, A. (1984). Contribution à une Approche Méthodologique de l'Assemblage Automatisé: Élaboration Automatique des Séquences Opératoires, PhD Thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté.
13. Delchambre, A. (1992). *Computer-aided Assembly Planning*, London, UK: Chapman & Hall.
14. Huang, Y.F. and Lee, C.S.G. (1991). A Framework of Knowledge-based Assembly Planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, CA, pp. 599–604.
15. Homem De-Mello, L.S. and Sanderson, A.C. (1990). AND/OR Graph Representation of Assembly Plans, *IEEE Transactions on Robotics and Automation*, **6**(2):188–199.
16. Homem De-Mello, L.S. and Sanderson, A.C. (1991). Representation of Mechanical Assembly Sequences, *IEEE Transactions on Robotics and Automation*, **7**(2): 228–240.
17. Homem De-Mello, L.S. and Sanderson, A.C. (1989). A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences, *IEEE Transactions on Robotics and Automation*, **1**: 56–61.
18. Laperriere, L. and Eimaraghy, H. (1996). GAPP: A Generative Assembly Process Planner, *Journal of Manufacturing Systems*, **15**(4): 282–293.
19. Lee, S. and Shin, Y.G. (1990). A Cooperative Planning System for Flexible Assembly, In: *Proceedings of the 2nd International Conference on Computer Integrated Manufacturing*, Troy, NY.
20. Wolter, J.D. (1989). On the Automatic Generation Assembly Plans, In: *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, pp. 62–68.
21. Lin, M.C., Tai, Y.Y., Chen, M.S. and Alec Chang, C. (2007). A Rule Based Assembly Sequence Generation Method for Product Design, *Concurrent Engineering: Research and Applications*, **15**: 291–308
22. Bonneville, F., Perrard, C. and Henrioud, J.M. (1995). A Genetic Algorithm to Generate and Evaluate Assembly Plans, In: *IEEE Symposium on Emerging Technology and Factory Automation*, Paris, France, pp. 231–239.
23. Sebaaly, M.F. and Fujimoto, H. (1996). A Genetic Planner for Assembly Automation, In: *Proceedings of the IEEE Conference on Evolutionary Computation*, Nagoya, Japan, pp. 401–406.
24. Siddique, Z. (2000). Common Platform Development: Designing for Product Variety, Doctoral Dissertation, Georgia Institute of Technology.
25. Wilmes, L. and Siddique, Z. (2004). Applicability of Design Spaces to Utilize Current Assembly Plant Resources to Produce New Product Family Members, In: *Proceedings of ASME 2004 Design Engineering Conference*, Paper No. DETC2004-52528, Salt Lake City, Utah.
26. Orlov, M. (2002). Efficient Generation of Set Partitions. Available at: http://www.informatik.uni-ulm.de/ni/Lehre/WS04/DMM/Software/partitions.pdf (accessed April 20, 2010).
27. Nugen, F. (1999). Enumerating with Constraints on the Hierarchy Space, Unpublished Manuscript, Georgia Institute of Technology.

## Shilpitha Bomi Reddy



Shilpitha Bomi Reddy is currently working as a Net Developer for The MitGroup. Ms. Reddy's research interests include: product platform design, assembly process generation, and development of computer applications to support designers. She completed her Masters from School of Aerospace and Mechanical Engineering at University of Oklahoma.

## Zahed Siddique



Prof. Zahed Siddique is currently working as an Associate Professor at the School of Aerospace and Mechanical Engineering of University of Oklahoma. His research interests include product family design, collaborative design, design education, and reverse engineering. Dr Siddique received his PhD in Mechanical Engineering from Georgia Institute of Technology.

Downloaded from cer.sagepub.com at UNIV OF OKLAHOMA on January 20, 2016