Prod. Eng. Res. Devel. (2014) 8:233–242
DOI 10.1007/s11740-013-0499-6

ASSEMBLY

Consensus tree method for generating master assembly sequence

Mohamed Kashkoush · Hoda ElMaraghy

Received: 30 July 2013 / Accepted: 13 September 2013 / Published online: 22 September 2013

Abstract An assembly process plan for a given product provides the sequence of assembly operations, their times as well as the required tools and fixtures for each operation. Much research has been done on automating and optimizing assembly sequence generation as the most important part of an assembly process plan. A novel method for generating the assembly sequence of a given product based on available assembly sequence data of similar products is presented. The proposed method uses a binary tree form to represent the assembly sequences of an existing family of products. A Genetic Algorithm is employed to find the consensus tree that represents the set of all assembly sequence trees with minimum total dissimilarity distance. This is similar to defining Generic Bill-of-Material. The generated consensus tree serves as a master assembly sequence for the product family. The assembly sequence for a new product variant that falls within, or significantly overlaps with, the scope of the considered family of products can be directly extracted from the derived master assembly sequence tree. The developed method is demonstrated using a family of three control valves. This novel method greatly simplifies and enhances automatic assembly sequence generation and minimizes subsequent modifications, hence, reduces assembly planning cost and improves productivity.

Keywords Assembly sequence generation · Consensus trees · Dissimilarity distance · Genetic algorithm

M. Kashkoush · H. ElMaraghy (✉)
Intelligent Manufacturing Systems (IMS) Centre, University of Windsor, 401 Sunset Avenue, Windsor, ON N9B 3P4, Canada
e-mail: hae@uwindsor.ca

1 Introduction

The sequence by which assembly operations are carried out for a given product has a great effect on many aspects of the assembly process such as the cost of assembly per unit, level of assembly difficulty, the need for fixtures, the likelihood of components damage during assembly, the ability to do in-process testing, and the likelihood of rework [1].

Finding a good assembly sequence automatically is not a simple task as the number of feasible assembly sequences could be very large even for a small number of components. The proliferation of product variety and frequent design changes demands responsive and cost-effective process planning. Significant research has been done on the automatic generation of assembly sequences [2–8].

Existing assembly sequence generation algorithms do not make full use of available legacy assembly sequence data. A novel method that addresses this problem from a different non-traditional perspective is proposed. It seeks to determine the master assembly sequence which facilitates subsequent variant sequence planning. A master assembly sequence is a generic assembly sequence for a given set of products which share a significant number of components and common product structure. Upon constructing the master assembly sequence for a given family of products, any new variant that falls within, or considerably overlaps with, the scope of the considered family of products could be directly extracted. Forming a master assembly sequence from individual assembly sequences is somehow similar to forming a Generic Bill of Material (GBOM) from a set of individual Bills-of-Material of a given product family to improve supply chain and production management [9, 10].

The concept of master assembly sequencing has been limitedly addressed before using different names [11–13].

For example, Martinez et al. [11] generated master assembly sequence, called parent plan, for a hypothetical product which includes all components of a product family (called meta-product). However, they employed a traditional assembly sequence planning approach (reviewed in Sect. 2) to generate the master assembly sequence for the meta-product.

In this research, a binary tree is used to represent any assembly sequence including the master assembly sequence. The concept by which the master assembly sequence is constructed is inspired by the well-known approach used in biology to find the consensus tree for a set of conflicting evolutionary trees [14, 15]. A new consensus tree method based on Genetic Algorithm has been developed to deal with the specific characteristics of assembly sequences. The method is demonstrated using a case study of three back-flushing control valves where the master assembly sequence for the three valve variants is derived and used to form the assembly sequence of a new valve variant.

The outline of paper is as follows; Sect. 2 reviews research related to assembly sequence generation. Section 3 defines the research scope and Sect. 4 describes the proposed method. An illustrative example is detailed in Sect. 5 and a case study is used for further demonstration and justification of the method in Sect. 6. Summary and conclusions are included in Sect. 7.

2 Related literature

Assembly sequencing is a mature research topic [16]. Research on automating or semi-automating assembly sequence generation goes back to the eighties and earlier [17]. However, the main principle adopted by most algorithms, with few exceptions (e.g. Reconfigurable Process Planning (RPP) [18]), is more or less the same.

A traditional assembly sequence generation algorithm has two phases. First, the set of all feasible or valid assembly sequences are generated according to a predefined set of feasibility or validity constraints. Geometrical and precedence feasibility of an assembly operation represent the major feasibility constraints. Second, the set of feasible sequences generated are searched for the best sequence according to specified optimization criteria such as minimum total number of changes of parts orientation and assembly tools during assembly [17].

Instead of carrying out the assembly sequence generation in two consecutive phases, the Generative Assembly Process Planner (GAPP) developed by Laperriere and ElMaraghy [5] used an A* algorithm [19] which simultaneously applies both feasibility constraints (geometric and accessibility constraints) and optimization criteria (number of parts reorientations, concurrent execution of assembly tasks, grouping of similar tasks and assembly stability). It finds the optimal assembly sequence without exhaustively generating all feasible sequences.

Recent algorithms combine the two phases by employing a fitness function that considers both assembly constraints and optimization criteria through a meta-heuristic optimization algorithm. The assembly sequence generation algorithm by Wang and Liu [8] follows this trend and is used next to explain how this type of assembly sequence generation algorithms works.

Wang and Liu considered an assembly sequence to be linear when all product components are assembled one at a time and sub-assemblies are never formed (i.e. it was assumed that no parallel assembly operations take place) [17]. A linear assembly sequences is typically represented by a permutation (sequence) vector the length of which equals the number of assembly components. The product design is analyzed and five component-to-component relational matrices are built. The first three matrices represent geometrical, local assembly precedence and assembly stability feasibility constraints that should be satisfied in the final assembly sequence. The remaining two matrices represent two assembly optimization criteria that may favour one sequence over another. These are the number of assembly tool changes and number of assembly connection changes in a generated assembly sequence. The number of assembly direction changes is also considered as a third optimization criterion and is applied through the same matrix used to represent the geometrical constraint. A discrete Particle Swarm Optimization (PSO) algorithm is then used to search for the best assembly sequence using a weighted fitness function of the mentioned constraints and optimization criteria. A schematic description for this assembly sequence generation algorithm [8] is shown in Fig. 1.

The work of Laperriere and ElMaraghy and Wang and Liu are just few examples of how assembly sequencing has been tackled for decades. Many variants of these algorithms exist in literature. Among the major elements that would distinguish an algorithm is for example the assembly sequences representation. Other common forms of assembly sequence representation include assembly states [5], partial assembly trees [20] and And/Or graphs [21]. Many different optimization techniques/algorithms have been employed such as Genetic Algorithm (GA) [22], Simulated Annealing (SA) [23], Ant Colony Optimization (ACO) [24], and Artificial Neural Networks (ANN) [7]. Recently, some algorithms used CAD models to automatically generate and apply assembly constraints which requires sophisticated geometry handling algorithms [25]. Assembly constraints may also be user-defined interactively by pre-analyzing the product [8] or using interactive

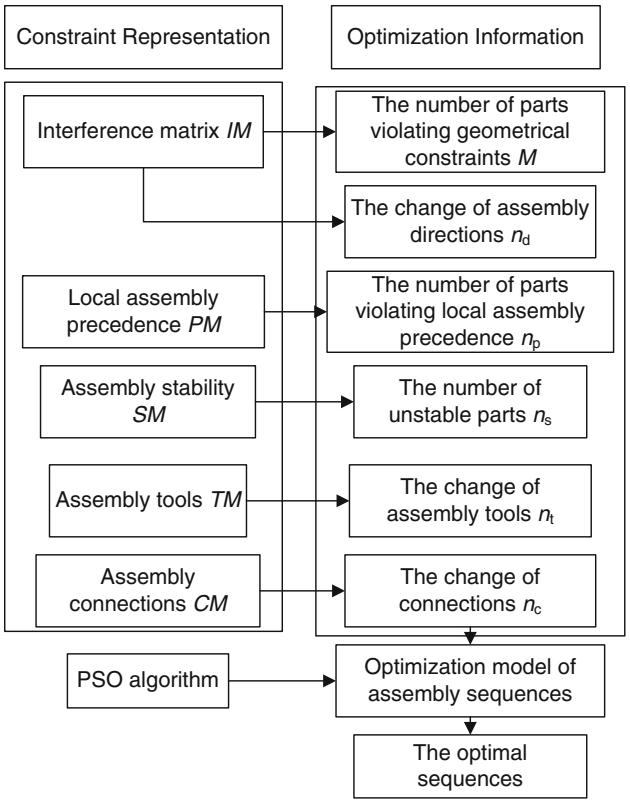

Fig. 1 Assembly sequence planning algorithm [8]

questions-and-answers [1]. Interactive methods would be practical with simpler and less complicated products. The type and number of assembly constraints and optimization criteria that an assembly sequencing generation algorithm considers vary depending on product type, assembly facility, assembly volume, and other factors. A comprehensive survey for dealing with the assembly constraints and optimization criteria is found in [26].

3 Scope

A non-traditional approach is proposed for assembly sequence generation. Existing individual assembly sequences for a given product family or a set of similar product variants which share a significant number of components and have a common product structure are merged together into one generic or master assembly sequence. The master sequence is then used to extract the assembly sequence for new product variants belonging to the same product family.

This research presents a simpler alternative to traditional assembly sequence generation by benefiting from the knowledge embedded in available assembly sequences for existing products/product families. Hence, the problem is to find the consensus (or generic) assembly sequence that best represents the set of available individual sequences even if some conflicts among them exist (e.g. different assembly sequence exits for the same combination of components).

More specifically, given a set of N assembly sequences, for N products with a total of n different components, it is required to find a single assembly sequence of all the n components with the minimum conflict with existing sequences. Measuring conflict will be detailed in Sect. 4.4. The following assumptions are made:

- Non-linear assembly with parallel operations is allowed.
- Assembly operations are sequential with one component added at a time.
- Assembly sequence data for existing product variants are available.
- The same name or part number is used for various versions or variants of the same component.

If the assembly sequences of available product variants are not optimal sequences, the proposed method still works well; however, the resulting master assembly sequence will be feasible but not necessarily optimal. Non-linear assembly is the general case of linear assembly which is also allowable. Assembly sequence for a non-linear assembly problem could be perfectly represented as an unordered rooted binary tree or what is known in the assembly planning literature as partial assembly trees [27]. The root of a partial assembly tree represents the final product (complete assembly) and the leaves represent individual components. Every other intermediate node represents the subassembly resulting from adding its two sub-nodes.

Figure 2 shows the partial assembly tree representing the assembly sequence of a product consisting of five components. According to this plan, components 1 and 3 may be assembled before, after, or at the same time as components 2 and 4. Component 5 is then added to sub-assembly (1, 3, 2, 4) to obtain the final product (1, 3, 2, 4, 5).

According to this representation, it is required to find the generic or *consensus* tree that best represents a given set of partial assembly trees or *cladograms*. The terms consensus and cladograms are both adopted from the biology and phylogenetics literature where these tools are used. A

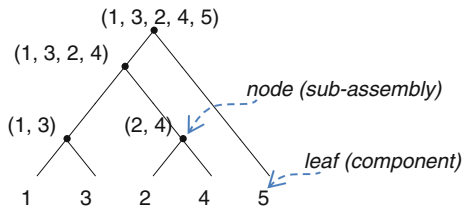

Fig. 2 An example for a partial assembly tree

cladogram is a phylogenetic tree which basically has the same structure and properties as a partial assembly tree and is used in biological studies to classify species and analyze their evolutionary histories [28]. It is not unusual in biology to have different classifications for the same set of species, where a consensus tree [14] for these classifications is sought. Existing methods for constructing the consensus tree deal with trees that have the exact same set of leaves, which is not always the case in assembly sequences.

4 Finding master assembly sequence using genetic algorithms

This section presents the Genetic Algorithm-based method developed for sequential and non-linear assembly applications to construct the consensus tree for a given set of partial assembly trees (assembly sequence trees or assembly trees for short) that do not necessarily have the same number of components. For a given set of individual assembly sequence trees, the master assembly sequence tree is equivalent to their consensus tree. Many algorithms and methods have been developed in the biology and phylogenetics literature to construct consensus tree. However, as mentioned before, none of them handle trees with different number of leaves. Thus, a new consensus tree building method is developed to deal with this general case. MATLAB® programming and numerical computational software was utilized to implement the proposed method.

4.1 Methodology

Any of the available assembly sequence trees is first encoded into m x m square matrix form (m is the number of leaves) that maintains the same information provided by the trees. For any given matrix, the corresponding tree could be restored easily.

Consequently, for a given set of N trees and a total of n different components, there is an unknown $n \times n$ square matrix that represents the consensus tree of all the available N trees. There are different definitions for the consensus tree depending on the applied consensus method [15] such as the strict consensus and majority rule consensus. In this model, the consensus tree is defined as the tree that has the minimum sum of distances from each of the individual trees. Such a distance is measured using the commonly used measure in phylogenetics known as *Robinson-Foulds* distance [29]. Besides, the consensus tree must be a binary tree and has to also include all components of the individual trees, where each component is to appear only once.

Hence, a Genetic Algorithm [30] is applied to a set of initial randomly generated consensus trees (initial population) to search for the optimal consensus matrix that has the minimum sum of Robinson-Foulds distances from each of the individual matrices. Figure 3 shows an IDEF0 model of the proposed method illustrating the main activities as well as inputs, outputs, controls and mechanisms for each activity.

4.2 Tree encoding scheme

A new tree-to-matrix encoding scheme, based on the one introduced by ElMaraghy and AlGeddawy [31], is used to calculate the Robinson-Foulds distance between any two trees as well as to support Genetic Algorithm implementation. In encoding a tree of m leaves into a matrix, the information to be represented by the matrix is the hierarchy of the m elements which is equivalent to those elements belonging to the same node. This is carried out through the encoding scheme as shown in Fig. 4.

Two types of tree information are encoded: *sequence of leaves* and *topology* or tree structure. Sequence of the leaves is encoded by the diagonal elements of the encoding matrix, while topology is encoded by the locations of binary (0–1) elements above the diagonal. Elements of value 1 above the diagonal signify the group of leaves belonging to the same node. For example, the four elements 1, 3, 2 and 4 on the tree shown in Fig. 4, belong to the same node. Hence, the cell corresponding to the first column, where leaf 1 (left most leaf of the node) is located in the matrix; and the second row, where leaf 4 (right most leaf of the node) is located in the matrix, is given a value of 1. In this way, the hierarchy and grouping relationships among elements are maintained in the encoded matrix. In any encoded matrix, the cell (1, 1) will always take the value 1 (this is the root node to which all leaves belong).

4.3 Generating initial population of master assembly sequence trees

A Genetic Algorithm starts with an initial set of solutions known as initial population. This would be in the form of assembly sequence trees of n leaves or its equivalent matrices, where n is the total number of different leaves of all individual trees. Randomly generating an initial feasible population of assembly sequences; either in tree or matrix format, is challenging. One approach is to randomly generate a set of $n \times n$ matrices with random integers (from 1 to n) in the diagonal cells and a number of $n - 1$ ones (equivalent to number of nodes) in different cells above the diagonal. The main difficulty with this approach is that the resulting matrices are likely to be infeasible assembly sequence trees. Controlling the random generation process so that only valid assembly sequence matrices are generated or converting infeasible matrices into valid ones is complicated.

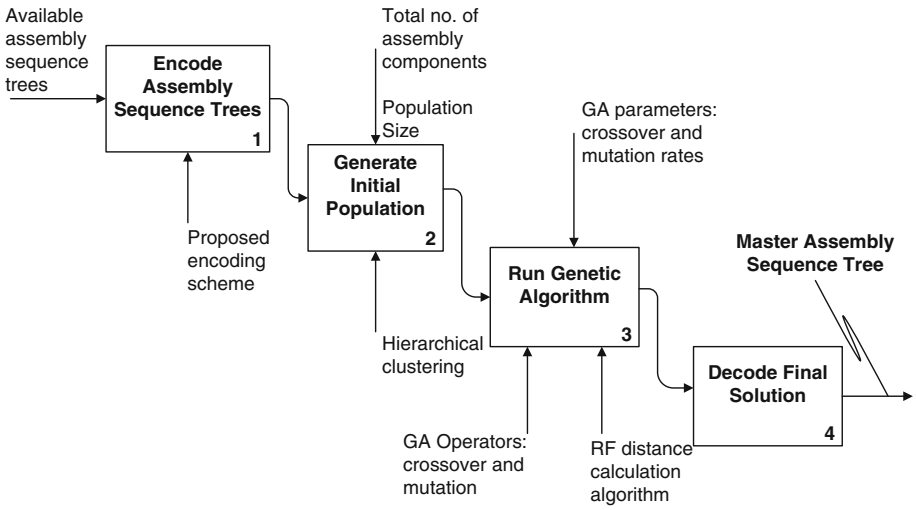

Fig. 3 IDEF0 model for GA-based search for consensus trees

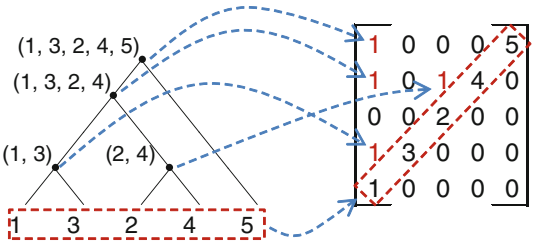

Fig. 4 Proposed tree-to-matrix encoding scheme

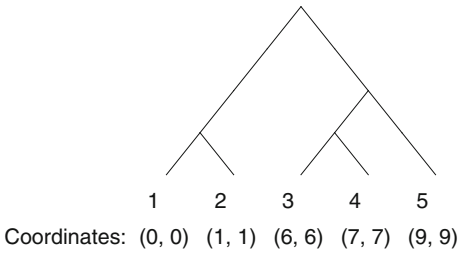

Fig. 5 Tree of 5 leaves generated by hierarchical clustering

A much simpler solution and easier to implement is proposed where valid matrices are randomly generated directly. To generate an $n \times n$ matrix, representing a feasible assembly sequence tree of n leaves, a set of n random coordinates are generated. An agglomerative (bottom up) hierarchical clustering algorithm, based on Euclidian distance as a proximity measure and single-linkage (nearest neighbor) as a clustering method [32], is applied to build the corresponding binary hierarchical clustering tree, also known as dendrogram. Dendrogram is an equivalent representation to assembly sequence trees used in this method.

The "Linkage" function of the Statistics toolbox of MATLAB® was used for hierarchical clustering. The Linkage function output is converted into the proposed matrix form before proceeding to the Genetic Algorithm iterations. Figure 5 shows the tree obtained by hierarchical clustering if the coordinates (0, 0), (1, 1), (6, 6), (7, 7) and (9, 9) are randomly generated. Equal integer values of x and y coordinates are used here for simplicity, but in the actual algorithm implementation real numbers, which are not necessarily equal, are used to minimize the possibility of having ties in proximity which occurs when two or more minimum Euclidian distances between different clusters have the same value during the agglomerative process. Ties in proximity could be totally eliminated in single-linkage clustering by adding/subtracting a small value to/from the coordinates of leaves resulting in ties.

4.4 Robinson-Foulds distance and fitness function

The Robinson-Foulds distance [29], is the most widely used metric for comparing phylogenetic trees [33]. Given two trees *T1* and *T2*, both having m number of leaves, *C1* is a set that includes $m - 1$ subsets, each represents one of the $m - 1$ nodes of *T1* and the elements inside each subset are the elements belonging to the node representing the subset. Similarly, *C2* contains $m - 1$ subsets representing the $m - 1$ nodes of *T2*. Robinson-Foulds distance (*RF*) is then given by Eq. 1, where "Δ" refers to symmetric difference (a set theory operation). Eq. 1 could be further detailed as in Eq. 2, where "\" refers to set difference operation. Hence, *RF* is simply a normalized count of the nodes (i.e. clusters of leaves) that exist in one tree, but not the other.

$$RF\,(T1,\,T2) \;=\; {}^{1}\!/_{2}\,|C1\,\Delta\,C2| \tag{1}$$

$$RF\,(T1,\,T2) \;=\; {}^{1}\!/_{2}(|C1 \setminus C2| \;+\; |C2 \setminus C1|) \tag{2}$$

For instance, the two trees *T1* and *T2* shown in Fig. 6 each has five leaves and four nodes. For *T1*, $C1 = \{\{1, 3, 2, 4, 5\}, \{1, 3, 2, 4\}, \{1, 3\}, \{2, 4\}\}$ and for *T2*, $C2 = \{\{1, 2, 3, 4, 5\}, \{1, 2, 3, 4\}, \{1, 2\}, \{3, 4\}\}$. The order of sets

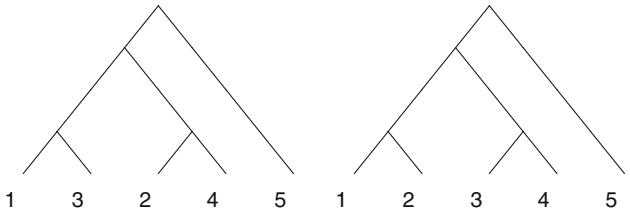

Fig. 6 Two Trees *T1* and *T2* with *RF (T1, T2)* = *2*

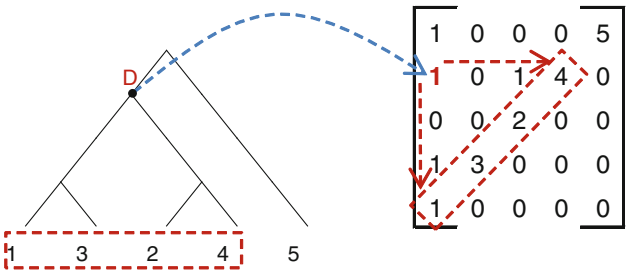

Fig. 7 Obtaining the subsets of the C set for a given tree

within *C1* and *C2* or order of elements within any of their subsets has no significance. By substituting in Eq. 2, we get *RF (T1, T2)* = ½ (2 + 2) = 2.

Many algorithms have been developed for calculating the Robinson-Foulds distance with different computational efficiency in finding and storing the *C* sets). Some of these algorithms are exact [34] and others are approximate [33]. The most recognized algorithm for calculating the Robinson-Foulds distance is Day's algorithm [35]. In this research, an algorithm based on the proposed matrix representation has been developed. Obtaining the *C* set for any given tree becomes rather straightforward given the used matrix encoding scheme. With reference to Fig. 7, the subset of the *C* set that represents node *D* of the shown tree simply includes the group of consecutive diagonal elements {1, 3, 2, 4}, identified by the position of the cell representing *D* in the corresponding matrix.

Throughout the proposed GA iterations, a fitness function is used to assess the fitness of any candidate solution (master assembly sequence tree). For a given set of available assembly sequence trees *N*, with a total of *n* different components, and a candidate master tree *MT*, the fitness function is the average of the Robinson-Foulds distances between the candidate master tree *MT* and every individual tree *T* out of the *N* available trees (Eq. 3).

$$Fitness = \frac{\sum_{i=1}^{i=N} \boldsymbol{RF}(\boldsymbol{MT}, \boldsymbol{T_i})}{\boldsymbol{N}} \quad (3)$$

In most cases, *MT* has more components than *T*. Thus, during calculating Robinson-Foulds distance between a given master tree *MT* and any individual tree *T*, all elements that exist in *MT* but not in *T* are ignored and temporarily removed from the *C* set of *MT*.

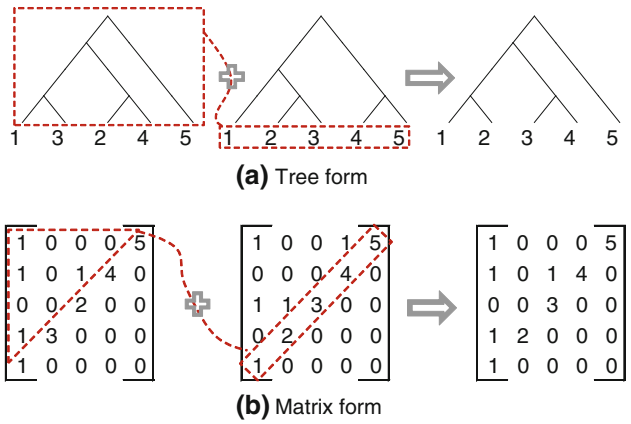

Fig. 8 Proposed crossover operator

4.5 Genetic algorithm operators

Genetic Algorithm (GA) is an evolutionary optimization meta-heuristic originally introduced by Holland [30], inspired by the process of natural selection. In a typical Genetic Algorithm, crossover and mutation are the main operators by which a new generation of solutions evolves from the current population. An effective GA needs well-designed crossover and mutation operators as well as proper adjustments of their rates. Furthermore, employing a local search routine within a meta-heuristic significantly improves its performance.

4.5.1 Crossover

Crossover is the main GA operator and it is the mechanism by which a new solution (offspring) is generated from the combination (mating) of two randomly selected solutions (parents). A specially designed crossover operation is developed to allow proper information exchange between any two combined trees which guarantees the feasibility of the new tree. According to the proposed crossover operation, for two given randomly selected matrices, a new matrix is generated by obtaining (inheriting) the topology part of one matrix (elements above the diagonal) and the leaves sequence part of the other matrix (diagonal elements). Hence, there is no chance of generating matrices that are not equivalent to valid assembly sequence trees. Figure 8a, b illustrate the proposed crossover operator mechanism in both tree and matrix forms. The tree form is shown for clarification.

4.5.2 Mutation

Mutation is an essential GA operator to help evolve new solutions that would not be obtained by crossover alone. Without a proper mutation operator, a premature

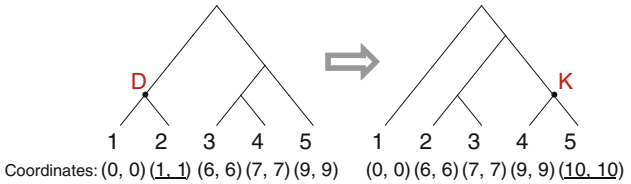

Fig. 9 Proposed topology mutation operator

convergence towards a local optimum solution could easily occur. Three mutation operators where developed; two of them involve mutating sequences of leaves (diagonal elements of the matrix form) and the third mutates topologies (elements above the diagonal in the matrix form).

The first mutation operator reverses the sequence of leaves of a randomly selected tree. The second swaps two randomly selected leaves. The third randomly alters the topology of a randomly selected tree. As mentioned in Sect. 4.3, the initial population is produced by random generation of coordinates that are then clustered into trees through agglomerative hierarchical clustering. Hence, performing the topology mutation task without generating invalid trees could be simply done by assigning a new random coordinate to a randomly selected leaf of a randomly selected tree then rebuilding the tree. Figure 9 shows how the tree in Fig. 5 would look like if the coordinate of the second leaf from the left (leaf 2) was changed from (1, 1) to (10, 10) without changing the sequence of the leaves. According to that perturbation, node *D* in the original tree disappears and new node *K* is formed instead in the mutated tree.

4.5.3 Local search

The last constituent of the implemented Genetic Algorithm is a local search routine applied, during each GA iteration, to the current global best solution in order to find better ones locally. Similar to the third mutation operator, three local search operations are carried out to examine minor altered solutions from the global best one so far. The first operation is to swap two randomly selected coordinates. The second is to alter the value of a randomly selected coordinate by adding or subtracting a randomly generated value within pre-defined limits. The third operation is to insert randomly selected coordinates in a randomly selected position. After each local search operation a new tree (topology and sequence) is built and assessed and the global best solution is updated accordingly.

Each time any of these local search operations is applied it is kept iterating for a predefined number of iterations (e.g. 20 iterations for each local search operation). Furthermore, the global best solution is always kept in any newly generated population.

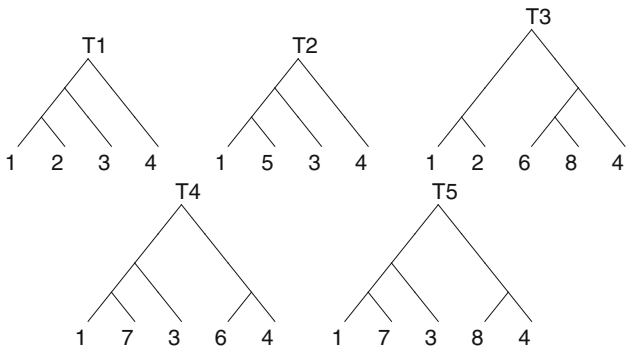

Fig. 10 Assembly sequence trees for family of five variants

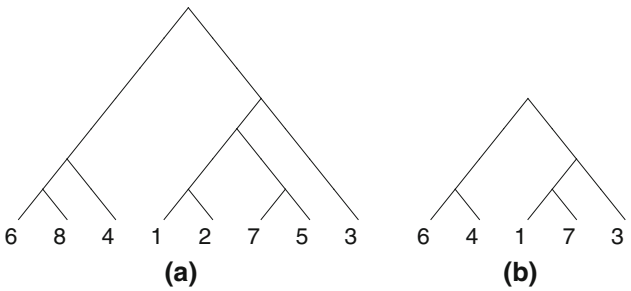

Fig. 11 **a** Master assembly sequence tree and **b** Extracted tree for a new family variant

5 Illustrative example

A hypothetical example is presented to illustrate the proposed method. Consider five assembly sequences for five variants in a given product family, involving a total of eight different components. It is required to find the master assembly sequence tree of that product family, and then extract the assembly sequence of a new variant which has a new combination of those components. The trees representing the assembly sequences for this product family are shown in Fig. 10. The results of the test runs have favored the following values of the GA parameters: 0.7 for crossover, 0.1 for each mutation operator, 25 for no. of local search iterations and 100 for population size.

Upon running the GA using these parameters values, the optimal consensus tree (Fig. 11a) with zero Robinson-Foulds distance from any of the five trees was obtained in 27 s on a PC of Intel Core2 Quad 2.83 GHZ processor and 4 GB Ram. This consensus tree is the master assembly sequence tree for the considered product family. The assembly sequence tree for a new product variant that has the components 1, 3, 4, 6, and 7 could then be extracted from the tree in Fig. 11a by ignoring any component that is not present in the new product components. Accordingly, the assembly sequence tree for the new product variant is shown in Fig. 11b.

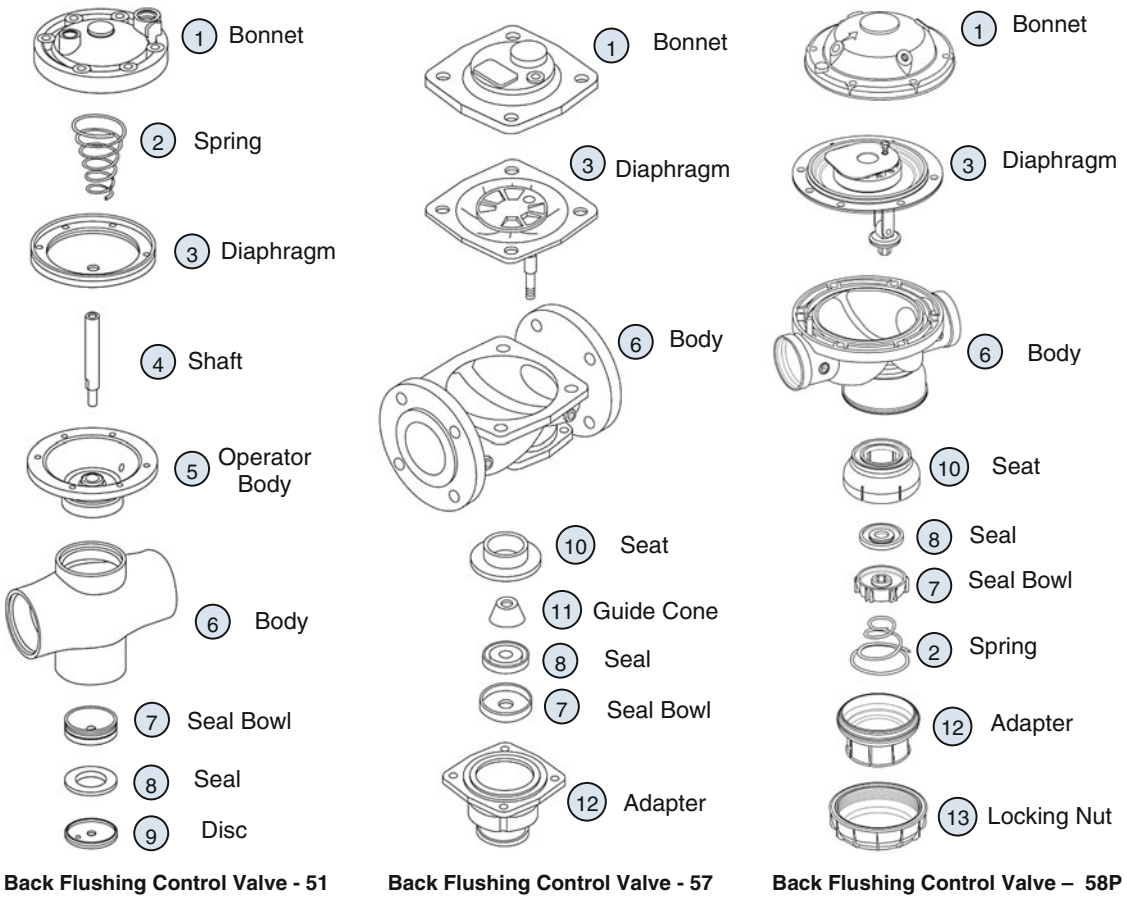

Fig. 12 Exploded views for the family of control valves extracted from the manufacturer catalogue [36]

6 Case study

The proposed method is further demonstrated using a real case study involving three different variants of a back-flushing control valve adopted from Dorot® product catalogue [36]. Back-flushing is a water filtration solution that discharges the water flow which carries dirt and particles out of the system. Those three back-flushing valve variants are considered existing/legacy designs by the manufacturer with known and consistently developed assembly sequence plans. The three variants are schematically shown in Fig. 12 where both the first and third variants have 9 components, while the second variant has 8 components. The total number of different components is 13. Each component in the figure is labeled with a name and a number.

Figure 13 shows the assembly sequence trees extracted for each valve variant from the exploded views provided in the manufacturer catalogue. The objective is to construct the master assembly sequence of this family of valves and to use it to extract the assembly sequence of a new valve variant. The master assembly sequence tree (consensus tree) obtained using the proposed method is shown in Fig. 14. The tree was obtained in <3 min on the same PC used in the illustrative example using the same values of the algorithm parameters.

The objective function of the obtained consensus tree is non-zero, which means that there exists a Robinson-Foulds distance between the master tree and some of the three trees. Particularly, the master tree has a distance of 2 from tree *T1* while it has a zero distance from the other two trees *T2* and *T3*. This is actually attributed to the different order for assembling the spring (component 2) in the assembly trees of the first and third variants. A perfect consensus tree (with a zero objective function) does not always exist, hence, the algorithm is designed to terminate when it exceeds a pre-defined number of iterations (100 in this example) without improving the objective function.

A new back flushing valve (back-flushing control valve—62) has the following eight components: bonnet (1), diaphragm (3), chamber (14), spring (2), shaft (4), body (6), seat (10), and adapter (12). Its assembly sequence tree shown in Fig. 15 was extracted from the master tree by considering only those eight components out of the 13 components in the tree. The obtained master

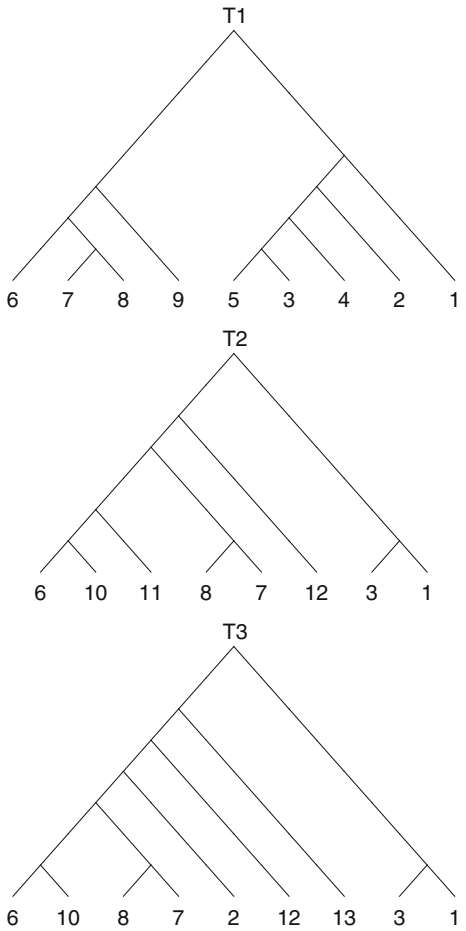

Fig. 13 Assembly sequence trees for a family of three back-flushing control valves

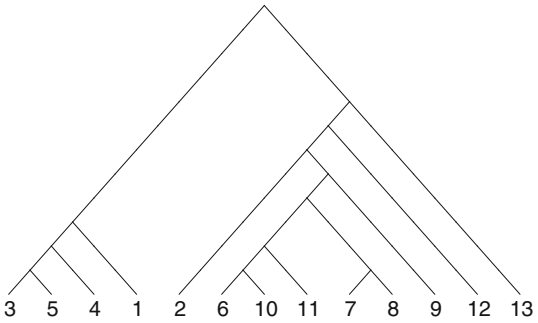

Fig. 14 Obtained master assembly sequence tree for a family of three back-flushing valves

plan is not a perfect one (it has a non-zero objective function), thus any component with conflicting assembly sequence data, which is the spring in this example (component 2), should be checked by the planner. The new variant also has a new component, chamber (14), which is not in the three existing variants. The position of

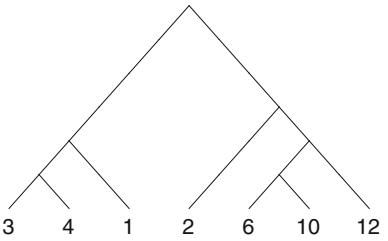

Fig. 15 Assembly sequence tree for a new valve extracted from the master tree

that component within the extracted assembly sequence should be then decided by the planner. This greatly reduces the assembly sequence planning effort for the new variant.

The advantage of the proposed method is best demonstrated by its ability to generate the assembly sequence of groups of components that has never existed together in any of the studied variants such as the shaft (4) and the seat (10) as well as the shaft (4) and the adapter (14). Such an advantage cannot be directly achieved using individual process plans retrieval methods.

7 Summary and conclusions

A novel method has been developed for generating a master assembly sequence for sequential non-linear product assembly applications. Given assembly sequence trees for a set of variants of a certain product family, a master assembly sequence tree is derived. Individual assembly sequence trees can be extracted for new variants that fall within, or significantly overlap with, the scope of the studied family of products. The method is capable of generating assembly sequences for groups of components that did not exist in any of the considered individual variants. This is an advantage over existing individual process plan retrieval methods. Partial assembly trees, Robinson-Foulds distance, consensus trees, hierarchical clustering and Genetic Algorithms were employed in the proposed method.

For cases involving conflicting sequences (i.e. having multiple different assembly sequences for the same combination of components), the proposed method generates the master assembly sequence with the most probable position for those conflicting components. When extracting the assembly sequence for a new variant, the planner should revise the assembly sequence of such conflicting components. The proposed method is being extended in order to allow automatic identification of conflicting components.

The quality and scope of the obtained master assembly sequence and subsequently extractable sequences for new variants depend on the quality and scope of the used assembly sequence data.

References

1. De Fazio TL, Whitney DE (1987) Simplified generation of all mechanical assembly sequences. IEEE J Robotics Autom RA-3 (6):640–658
2. ElMaraghy HA, Knoll L (1991) Design and automatic assembly sequence generation of a d.c. motor. Int J Veh Des 12(5–6):672–683
3. ElMaraghy HA, Laperriere L (1992) Modelling and sequence generation for robotized mechanical assembly. Robotics Auton Syst 9(3):137–147. doi:10.1016/0921-8890(92)90050-9
4. ElMaraghy HA, Rondeau JM (1992) Automatic robot program synthesis for assembly. Robotica 10:113–123
5. Laperriere L, ElMaraghy HA (1994) Assembly sequences planning for simultaneous engineering applications. Int J Adv Manuf Technol 9(4):231–244
6. Dini G, Failli F, Lazzerini B, Marcelloni F (1999) Generation of optimized assembly sequences using genetic algorithms. CIRP Ann Manuf Technol 48(1):17–20
7. Sinanoglu C, Borklu HR (2005) An assembly sequence-planning system for mechanical parts using neural network. Assembly Autom 25(1):38–52. doi:10.1108/01445150510578996
8. Wang Y, Liu JH (2010) Chaotic particle swarm optimization for assembly sequence planning. Robotics Comput Integr Manuf 26(2):212–222. doi:10.1016/j.rcim.2009.05.003
9. Romanowski CJ, Nagi R (2004) A data mining approach to forming generic bills of materials in support of variant design activities. J Comput Inf Sci Eng 4(4):316–328. doi:10.1115/1.1812556
10. Hegge HMH, Wortmann JC (1991) Generic bill-of-material. A new product model. Int J Prod Econ 23(1–3):117–128. doi:10.1016/0925-5273(91)90055-x
11. Martinez MT, Favrel J, Ghodous P (2000) Product family manufacturing plan generation and classification. Concurr Eng Res Appl 8(1):12–23. doi:10.1106/7p7h-5ejt-glct-nu8d
12. Lai H-Y, Huang C-T (2003) Integrated assembly plan generation system for grouped product families. Int J Prod Res 41(17):4041–4061
13. Gupta S, Krishnan V (1998) Product family-based assembly sequence design methodology. IIE Trans 30(10):933–945
14. Adams EN (1972) Consensus techniques and the comparison of taxonomic trees. Syst Zool 21(4):390–397
15. Dong J, Fernandez-Baca D, McMorris FR, Powers RC (2010) Majority-rule (+) consensus trees. Math Biosci 228(1):10–15. doi:10.1016/j.mbs.2010.08.002
16. Azab A, Samy S, ElMaraghy H (2008) Modeling and optimization in assembly planning. Paper presented at the 2nd CIRP Conference on Assembly Technologies and Systems (CATS), Toronto, Canada
17. Whitney DE (2004) Mechanical assemblies: their design, manufacture, and role in product development, vol 1. Oxford University Press, USA
18. Azab A, ElMaraghy HA (2007) Mathematical modeling for reconfigurable process planning. CIRP Ann Manuf Technol 56(1):467–472. doi:10.1016/j.cirp.2007.05.112
19. Nilsson NJ (1980) Principles of artificial intelligence. Springer, Berlin
20. Miller JM, Hoffman RL (1989) Automatic assembly planning with fasteners. In: IEEE International conference on robotics and automation, Scottsdale, AZ, USA, pp 69–74
21. Homem de Mello LS, Sanderson AC (1989) A correct and complete algorithm for the generation of mechanical assembly sequences. In: IEEE International Conference on Robotics and Automation, Washington, DC, USA, pp 56–61. doi:10.1109/robot.1989.99967
22. Marian RM, Luong LHS, Abhary K (2002) Assembly sequence planning and optimisation using genetic algorithms.I. Automatic generation of feasible assembly sequences. Appl Soft Comput 2(2):223–253
23. Hong DS, Cho HS (1997) Generation of robotic assembly sequences with consideration of line balancing using simulated annealing. Robotica 15:663–673. doi:10.1017/s0263574797000799
24. Wang JF, Liu JH, Zhong YF (2005) A novel ant colony algorithm for assembly sequence planning. Int J Adv Manuf Technol 25(11–12):1137–1143. doi:10.1007/s00170-003-1952-z
25. Pan C, Smith SS, Smith GC (2006) Automatic assembly sequence planning from STEP CAD files. Int J Comput Integr Manuf 19(8):775–783. doi:10.1080/09511920500399425
26. Jones RE, Wilson RH, Calton TL (1998) On constraints in assembly planning. IEEE Trans Robotics Autom 14(6):849–863. doi:10.1109/70.736770
27. Wolter JD (1992) A combinatorial analysis of enumerative data structures for assembly planning. J Desi Manuf 2(2):93–104
28. Page RDM (1994) Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. Syst Biol 43(1):58–77
29. Robinson DF, Foulds LR (1981) Comparison of phylogenetic trees. Math Biosci 53(1–2):131–147. doi:10.1016/0025-5564(81)90043-2
30. Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. MIT press, Cambridge
31. ElMaraghy HA, AlGeddawy T (2012) Co-evolution of products and manufacturing capabilities and application in auto-parts assembly. Flex Serv Manuf J 24(2):142–170. doi:10.1007/s10696-011-9088-1
32. Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv (CSUR) 31(3):264–323
33. Pattengale ND, Gottlieb EJ, Moret BM (2007) Efficiently computing the Robinson-Foulds metric. J Comput Biol 14(6):724–735
34. Asano T, Jansson J, Sadakane K, Uehara R, Valiente G (2010) Faster computation of the Robinson-Foulds distance between phylogenetic networks. In: Amir A, Parida L (eds) Combinatorial pattern matching. Springer, Berlin, pp 190–201
35. Day WH (1985) Optimal algorithms for comparing trees with labeled leaves. J Classif 2(1):7–28
36. Dorot (2001) Dorot automatic control valves. Catalogue