Robotics and Computer–Integrated Manufacturing 72 (2021) 102180

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Assembly sequence and path planning for monotone and nonmonotone assemblies with rigid and flexible parts

Ellips Masehian [a,*], Somayé Ghandi [b]

[a] Industrial and Manufacturing Engineering Department, California State Polytechnic University, Pomona 91768, USA
[b] Department of Industrial Engineering, Faculty of Engineering, University of Kashan, Kashan, Iran

ARTICLE INFO

Keywords:
Assembly sequence and path planning
Flexible parts
Nonmonotone assembly plan
SPP-Flex algorithm
Bidirectional Exploration-Exploitation Tree (BXXT)
Finite element analysis

ABSTRACT

The Assembly Sequence and Path Planning (ASPP) problem deals with finding a proper sequence of parts to be assembled into a finished product and short assembly paths for each part. The problem is a combination of Assembly Sequence Planning (ASP) and Assembly Path Planning (APP) subproblems, which are both NP-complete and therefore intractable at large sizes. Nearly in all works on ASPP, it is assumed that planning is monotone (i.e., parts are moved only once, without considering intermediate placements) and each part is completely rigid. These are simplifying, yet limiting assumptions, since most assembled products like ships, aircraft, and automobiles are composed of rigid and flexible parts, and the generation of assembly sequence and path plans for most real-world products requires intermediate placement of parts to be taken into account. None of the existing works in the literature, however, have handled nonmonotone ASPP problems for rigid and flexible parts, and this issue remains largely untouched. In this paper, we present a new method called SPP-Flex for solving monotone and nonmonotone ASPP for rigid and flexible parts. SPP-Flex first utilizes a Directional Assembly Stress Matrix (DASM) for describing interference relations between all pairs of parts and the amounts of compressive stresses needed for assembling flexible parts and then obtains an initial tentative assembly sequence using a simple new greedy heuristic. Next, short assembly paths are iteratively computed and planned from initial to goal configurations of all parts using a novel sampling-based path planner called BXXT. If finding a free path for an active part fails due to obstruction of a previously assembled part, then such a part is identified, relocated, and its path replanned until the active part is moved to its final position. In case of failure again, if the part is flexible, through finite element analysis, it is determined if the part can still be assembled by undergoing elastic deformation. To evaluate the performance of the SPP-Flex and its components, two new products were designed and solved by four combinations of ASP and APP methods 20 times each, and the means and standard deviations of five performance criteria (total path length, total number of generated nodes and edges in the search tree, total number of collision (interference) checks, and total runtime) were calculated. Analysis of the computational results showed that the proposed greedy heuristic sequence planner together with the BXXT path planner/replanner outperformed other variations with at most 4.6% average gap in path length and 2.1% average gap in runtime compared to the best-found solution in all runs.

1. Introduction

Assembly Planning (AP) is the process of creating a detailed assembly plan to build a final product from separate parts by taking into account the final product geometry, available resources to manufacture that product, fixture design, feeder and tool descriptions, etc. Assembly planning is one of the most important processes in manufacturing products since assembly processes use up to 20% of the total manufacturing cost and more than 50% of the total production time [55]. Therefore, efficient assembly plans can reduce manufacturing costs and time significantly. The Assembly Planning problem has been shown to be an NP-complete problem [38] and covers three main subproblems: Assembly Sequence Planning (ASP), Assembly Line Balancing (ALB), and Assembly Path Planning (APP). The ALB problem tries to group and allocate the assembly operations into some workstations with approximately equal assembly times while precedence constraints between operations are satisfied. Therefore, ALB is at the macro level and does not involve the part geometries. On the other hand, ASP and APP

* Corresponding author.
E-mail addresses: masehian@cpp.edu (E. Masehian), s.ghandi@kashanu.ac.ir (S. Ghandi).

https://doi.org/10.1016/j.rcim.2021.102180
Received 10 February 2020; Received in revised form 18 April 2021; Accepted 19 April 2021
Available online 17 May 2021

Table 1
Some soft computing/metaheuristic algorithms for ASP.

Advanced Immune-based Strategy	[6]
Ant Colony Optimization (ACO)	[24,73]
Artificial Immune System (AIS)	[9]
Artificial Neural Networks (ANN)	[10,11,27,64]
Bacterial Chemotaxis algorithm	[85]
Breakout Local Search algorithm (BLS)	[20]
Enhanced Harmony Search (HS)	[74]
Firefly Algorithm (FA)	[81]
Genetic Algorithm (GA)	[8, 23]
Hybrid Ant-Wolf Algorithm (HAWA)	[1]
Imperialist Competitive Algorithm (ICA)	[84]
Memetic Algorithm (MA)	[18]
Particle Swarm Optimization (PSO)	[47,70,75,83]
Psychoclonal algorithm	[69]
Scatter Search algorithm (SS)	[19]
Simulated Annealing (SA)	[28,51]

are at the micro level, meaning that they heavily depend on the shape and geometry of the parts and the final assembly while the precedence constraints are equally important. In this paper, since we deal with assembly sequence and path planning of rigid and flexible parts, those two problems are reviewed in this section.

1.1. Assembly sequence planning

The ASP problem concerns with finding a sequence of collision-free operations that bring the assembly parts $p_1, \ldots, p_n$ together, having given the geometry of the final product A and the positions of parts in the final product. Since the ASP problem is shown to be NP-hard [47], in recent years intensive research efforts have been put in developing intelligent methods to solve the ASP problem as they have been able to improve the efficiency of finding optimal assembly sequences while avoiding the combinatorial explosion problem with the increase of the number of assembly components. Many soft computing/metaheuristic algorithms have been developed to solve the problem, such as the works mentioned in Table 1. More surveys on the ASP and its methods are presented in [33,58] and [16].

Here it is appropriate to elaborate on how the current paper improves two of our previous publications [20] and [19]:

In [20] a Breakout Local Search algorithm (BLS) was proposed for solving (only) the ASP problem under the following assumptions: (1) the method could handle rigid parts only and parameters like elasticity, force, and toleranced geometries were not considered; (2) the APP subproblem was not tackled; (3) movements of the parts were limited to translations at the six main directions ($\pm x$, $\pm y$, $\pm z$), and translating along any other direction and/or rotation about any axis were not accommodated; (4) other than the previously assembled parts, no obstacles (e.g., nearby fixtures or tools) within the assembly workspace were considered; (5) the plans for assembly sequences were considered to be *monotone*, meaning that once assembled (i.e., moved to its final configuration), each part cannot move again to one or more intermediate locations. The last assumption is very limiting as most real-world assembly plans are non-monotone. As will be presented later in this paper, all of the above assumptions have been discarded and the problem is modeled more realistically by considering rigid and flexible parts, ASP and APP plans, arbitrary translations and rotations, workspace obstacles, and non-monotone plans. Also, our other work [19] employed the Scatter Search algorithm (SS) to solve the ASP problem, and had the same simplifying assumptions as mentioned earlier, except that it could handle flexible parts. Again, the current paper is still significantly different from that work.

1.2. Assembly path planning

The APP problem consists of computing feasible (and preferably optimal) paths for adding parts to a subassembly, having given the geometry of the final product, the initial and final positions of parts, and the workspace in which the assembly operations take place. Assembly paths are formed according to the assembly sequences that are the output of the ASP problem. APP is an NP-hard problem and can be formulated as a motion planning problem. A configuration q is a minimal set of parameters defining the location of a mobile system in the world, and *Configuration Space C* is the set of all configurations. In the

Table 2
Major solution approaches to assembly and disassembly path planning.

Approach	Main methods
Graph-based	*Blocking Graph* [77,45,44,78,43], and [57]
Discretizes the space via constructing a graph	*Visibility Graph* [53,80]
Grid-based	*Potential Fields* [82]
Discretizes the space via constructing a regular mesh grid	
Sampling-based	*Probabilistic Roadmaps (PRM)* [67]
Abstracts the space by sampling a finite number of points in it	*Rapidly-Exploring Random Trees (RRT)* [3,42,71,49]
Space Decomposition	*Separating Directions* [61,60,50,72,66]
Simplifies the problem by dividing its solution space into subspaces or subproblems and solving each one	*Swept Volume* [29,30]
	Motion Space [25,68,17]
Interactive	*Interactive Path Planning* [34]
Uses human intelligence and interaction to solve complex tasks	*Interactive Path Verification* [15]

case of a system involving *n* mobile objects m_i (e.g. the parts of the assembly), the Composite Configuration Space *C* is the Cartesian product of the configuration spaces of all objects, C_{mi}. Given the initial configuration q_{init}, the problem consists in finding a feasible path in *C* from q_{init} to a final assembled configuration q_{ass}. A comprehensive survey on the APP and its methods is presented in [21], which reviews the state-of-the-art of the APP and Dis-Assembly Path Planning (DAPP) problems and their solution approaches through two new taxonomies. The survey also exposes and analyzes the characteristics and applications of the reviewed works widely. Table 2 presents the five major solution approaches that have been developed for the APP/DAPP problems.

Among the main APP approaches, Sampling-based methods have proven to be relatively more successful in solving problems with a higher number of parts as they do not explicitly construct the Configuration space (C-space), which grows exponentially with the number of parts. However, the main challenge for these methods is the ‘narrow passage’ problem (requiring the planner to sample in the tight parts of the C-space), and coping with it more effectively is still at the focus of path planning researchers. These methods were originally proposed for robot motion planning in the mid-1990s, and unlike early path planners which constructed the C-space explicitly and geometrically, they randomly sample numerous collision-free configurations (position and orientation) and connect them by some edges to form a graph which is then searched to obtain start-to-goal paths. In this way, they prevent explicit construction and exhaustive search of the C-space and thus significantly reduce the calculation time compared to exact optimal methods.

Sampling-based approaches create a search tree by sampling *nodes* (or configurations) from the C-space and then searching for solutions (paths) based on the generated tree. Each node represents a unique position and orientation of an object currently being considered for path planning. For example, for a free-flying part in 3D space, a node is shown by $q = (x, y, z, \alpha, \beta, \gamma)$. It can be verified through a collision-checking procedure if the part at the position and orientation defined by the node *q* does collide with any other object. In case of no collision, the node is labeled as ‘free’ and appended to a search tree. An *edge* is a hypothetical collision-free line segment connecting two free nodes, and a tree (resp. graph) is composed of acyclic (resp. cyclic) connections of nodes in the configuration space. Two major sampling-based algorithms are Probabilistic Roadmap Method (PRM) developed by [39] and Rapidly-exploring Random Tree (RRT) proposed by [41]. There are numerous variations to these two basic planners, such as MAPRM [76], Lazy PRM [7], Fuzzy PRM [52], PRM* [36], Semi-Lazy PRM [5], RRT-Connect [40], ML-RRT [13], T-RRT [4], RRT* [37], and Bi-RRT [35].

Since ASP does not take into account workspace objects (e.g., fixtures, grippers, robots, etc.) that obstruct the motion of parts and APP does not consider the precedence, interrelations, and sequence of parts, it is best to combine them and perform Assembly Sequence and Path Planning (ASPP), which is also a computationally hard problem. There are relatively fewer works on ASPP in the literature: one work is [29] in which a new Genetic Algorithm and Ants Algorithm (GAAA) was designed to rapidly plan an assembly sequence based on which through a Boundary Representation (B-Rep) filling algorithm, assembly paths were planned interactively with the help of the user. In [72] a planner was developed for finding optimal assembly sequences for parts stably manipulated by robots. A subassembly generating Breakout Local Search algorithm (SABLS) for micro-assembly applications was proposed in [71], in which after performing the ASP, the parts are manipulated to their final positions in the subassemblies using a path planning algorithm based on RRT*.

In all of the above-reviewed works, the main assumption is that the plans for assembly sequence and paths are monotone, meaning that once a part is moved to its assembled place, it will not obstruct the assembly of subsequent parts. This assumption, however, is a simplifying yet limiting one, since generating assembly sequence and path plans for most real-world assembled products requires intermediate placement of parts to be taken into account. An exception, though, is our previous work [49] in which a new method was presented to solve the ASPP problem by first generating a heuristic sequence plan and then planning the parts’ paths using sampling-based methods. The method can handle nonmonotone assembly sequence plans through its path replanning feature, which allows it to identify the assembled parts that block the assembly of an active part, relocate them to a collision-free zone, assemble the active part, and re-assemble the blocking parts back to their final location.

In most of the works on ASP, APP, and ASPP, the underlying assumption is that all parts of the assembly are rigid (not flexible) and their shapes do not change during the assembly process. However, considering the parts perfectly rigid is unrealistic since flexible parts with *articulated, toleranced*, or *flexible* geometries, as defined in the next section and such as springs, snap fits and pins, rubbers and plastics, interference fit assemblies, revolute joints, etc., have a major role in manufacturing industries. Unfortunately, taking flexibility into account introduces additional degrees of complexity to the ASP and APP problems, and that is why the number of works addressing non-rigid assembly planning is relatively very few, as reviewed below.

1.3. *Considering flexibility of parts in assembly planning*

Most real-world assembled products like ships, airplanes, and automobiles are composed of rigid and flexible parts, and so automatic generation of assembly sequence plans for such products requires the flexibility of flexible parts to be taken into account. The solution to the ASP and APP problem of a product with rigid and flexible parts depends heavily on the model that is used to simulate the deformation behavior of flexible parts. In general, there are two approaches in modeling the flexibility of flexible parts: *Geometrical* approach, in which single or multiple control points or shape parameters are used for manual adjustment of a flexible part in order to apply changes in its shape and model its desired configuration and *Physical* approach, in which some sort of integration mechanisms and physical principles (e.g., material characteristics, environmental constraints, and externally applied forces on a part) are used to compute the shapes or motions of flexible objects. The most commonly used strategies for building flexible objects with physical properties are: (1) Mass-spring; (2) Finite Element Methods; and (3) Point-Based systems [22].

Assembly sequence planning of flexible parts has rarely been addressed in the literature. [79] described the state of an assembly by a set of relations among the features of its components together with the basic algorithms to determine the possibility of performing a particular operation on string-like assemblies. A few other works deal with creating *contact states* of the flexible parts of an assembly. In that approach, information for the contact states is extracted from the geometrical models of the parts and the assembly and is used to construct a graph representation proper for assembly sequence and path planning. [59] identified different possible contact states between a linear flexible object and a rigid polyhedral body, and listed the feasible transitions between these states. A further elaboration on this formalism characterizes contact states by their stability and defining contact state transition classes [2]. A systematic review of the existing modeling techniques for volumetric, planar, and linear flexible objects is presented in [32] according to the type of manipulation goal: path planning, folding/unfolding, topology modifications, and assembly. Manipulation of Flexible Linear Objects (DLO) has potential applications in aerospace and automotive assembly. Most of the literature on planning for flexible objects focuses on a single DLO at a time. But in [62,63] a problem formulation for attaching a set of interlinked DLOs to a support structure through a set of clamping points was provided and a prototype algorithm was presented to generate a solution in terms of primitive manipulation actions. None of the above works, however, directly focuses on ASP with flexible parts, and as a result, this field of research has remained largely unexplored.

Flexible parts can undergo considerable variations in their shape and may be of two types: *Articulated*, and *Flexible*. Articulated parts are usually composed of one or two links connected to a base by revolute joints (somewhat similar to simple manipulators) and can change shape in certain directions. These parts are widely used in medical instruments, consumer products, toys, and household appliances [56]. The only work we found on (dis)assembly path planning of articulated objects is [14], which generalized its solution method to the protein-ligand interaction problem. Flexible parts, on the other hand, can freely and reversibly change their shape in the form of tension, compression, twist, or bend in as much as the modulus of elasticity of their substance permits.

(Dis)assembly path planning of flexible objects has been addressed in very few works: [46] proposed a method for virtual assembly via haptic to simulate assembly operations of an elastic tube, and [26] presented a method for automatically planning a smooth and collision-free path for a wiring harness to be inserted into the engine compartment of a car. Also, [31] proposed an approach to utilize wire tracing operation in recognizing the wire harness for automatic mating, and in [48] an assembly simulation method was developed to simulate the assembly process of multi-branch cables. Also, [65] proposed a new assembly strategy for learning skills from manual teaching to carry out the assembly process. To fit the teaching data, a Gaussian mixture model was used and the compliance control method was applied to conduct the assembly.

In none of the reviewed articles, the assembly sequence and path planning have been considered simultaneously for flexible (in addition to rigid) parts. The present paper aims to fill this gap by proposing a method for ASPP for rigid and flexible parts with nonmonotone assemblies. Based on our analysis of about 85 related and cited works in the ASP/APP literature, we can state that it is the first planner having the following attributes all together: (1) it is designed for directly solving ASPP problems (i.e., assembling a product from unassembled parts), thus useful for cases when ASPP cannot be obtained by the "assembly by disassembly" methodology, which is less challenging and contains reversing the solution to the disassembly sequence and path planning to reach the assembly sequence and path, (2) it can consider obstacles (e.g., fixtures, worktables, robot links, etc.) in the workspace, (3) it allows planning translational and rotational movements for parts, (4) it can handle nonmonotone ASP plans, and (5) it can solve ASPP problems for both rigid and flexible parts.

The planner proposed in this paper (named SPP-Flex) has the following differences from the method presented in our previous work (ASPPR) [49]: (*i*) The ASPPR can handle rigid parts only, whereas the SPP-Flex can plan for rigid and flexible (deformable) parts such as snap pins, belts, springs, etc. by incorporating parameters like elasticity, force, and toleranced geometry of such parts in the model. Accordingly, we have defined a new matrix called the *Directional Assembly Stress Matrix* (DASM) with elements representing the required stresses for assembling each part in each main direction. In addition, a new component, namely the Finite Element Analysis via the Abaqus™ software, has been incorporated into the SPP-Flex, which assists in defining the DASM and also determining whether parts' deformability can be utilized to resolve blockages during the assembly. (*ii*) In the ASPPR, a greedy heuristic method was used for planning the initial assembly sequence, after which the path planning module was triggered. There was no assessment of the effect of the ASP quality on the overall success of the ASPPR algorithm. However, in the present work, we have used two ASP algorithms, a (greedy) heuristic method and a (near-optimal) Breakout Local Search (BLS) method, and investigated the impact of their performance on the accomplishment of the ASPP task. (*iii*) A new matrix called *Intersections Count Array* (Q_p) has been defined in this paper to organize and record the number of times already-assembled parts or other fixed obstacles block the assembly of a part during its sampling-based path planning process, as presented in the detailed pseudocodes of the algorithm.

In the next section, we present the nature, model, and assumptions of the tackled assembly sequence and path planning problem, and in Section 3 we introduce the developed new SPP-Flex method together with its components and details. In Section 4, four variations and five performance criteria of the SPP-Flex method are described and implemented on two newly designed monotone and nonmonotone assemblies with both rigid and flexible parts, and the results are compared and analyzed. Finally, concluding remarks are presented in Section 5.

2. Problem assumptions

Before describing the details of our method for assembly sequence and path planning of assemblies with rigid and flexible parts, we present the assumptions on the nature of the tackled problem's model and the solution approach.

In solving the ASPP problem, we must find a feasible sequence for assembling the parts, as well as calculate a set of plans for the motions of some or all parts from their initial to final configurations. Thus, it is essential to consider the dimension, geometry, and constraints of the parts, as well as the limitations of the workspace where the task is performed, and the type of moves the parts are allowed to make.

- *Dimension*: The parts in the problem model can be 2D polygons or 3D polyhedrons, and may be convex or nonconvex in Euclidean space.
- *Types of components*: The problem model includes only assembly parts and a number of stationary obstacles in the workspace, which can represent jigs, fixtures, worktables, and part containers. Moving parts such as assembly tools, equipment, grippers, or robots, which are hardly considered in any previous work, are not considered here as well and remain open problems in this field.
- *Part geometry*: We consider two types of parts: perfectly rigid, which are assumed to have unique geometry, and flexible, which have flexibility and deviation (tolerance) from their mathematical or CAD model.
- *Types of movements*: A combination of rotation and translation is considered for moving parts, which enables solving a wide variety of assembly path planning problems.
- *Constraints*: It is assumed that the parts have precedence constraints in sequence planning and cannot have collisions with obstacles or intersections with other parts in path planning. In addition, physical properties of the parts such as applied forces as well as mechanical constraints (such as part deformations under tensional, torsional or

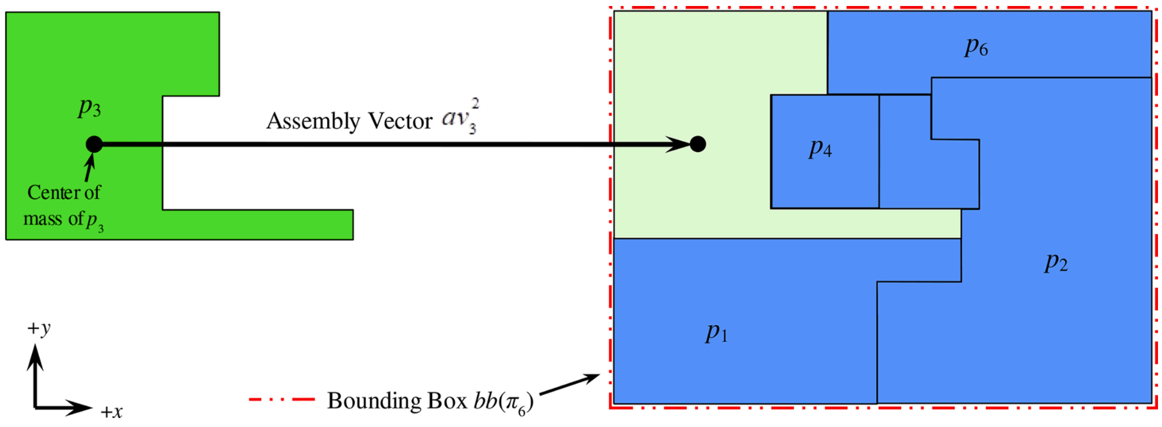


Fig. 1. Illustration of the Bounding Box and Assembly Vector used for assembly sequence planning of a sample product. Here part p_3 is the 6th part that is assembled along $+x$, therefore $\pi_6 = 3$, and $\delta_6 = d_2$.

compressional stresses) are incorporated into (dis)assembly operations.

The difficulty of an ASPP problem does not depend only on the number of parts in the assembly. The way the parts should be juxtaposed in the final assembly greatly affects the complexity of the problem. The most important attributes of an assembly from the planning standpoint are scale, sequentiality, monotonicity, linearity, and coherence, as described below:

- *Scale*: The scale of the addressed ASPP model is considered *Fine*, meaning that the free space between parts of the assembly is so narrow that small positional errors in locating parts in the assembly will lead to missing a feasible solution.
- *Sequentiality*: This refers to the maximum number of moving sub-assemblies with respect to one another in any assembly operation. Most real products can be assembled via sequential assembly plans, in which at any instant, only one part is in motion and the other parts remain fixed. As in the majority of previous researches, we also assume that the ASPP problem is sequential.
- *Monotonicity*: This refers to the need for the intermediate placement of at least one part of the assembly, which means some parts need to be moved more than once in order to solve the problem. A monotone problem needs no intermediate placements, while a nonmonotone problem does have such a requirement. Here we assume that the ASPP problem can be monotone or nonmonotone.
- *Linearity*: We assume that the problem is linear, meaning that all assembly operations involve the inserting of a single part into the rest of the assembly. In a nonlinear assembly, some parts need to be pre-assembled (forming a sub-assembly) and then inserted into their final place in the assembly.
- *Coherence*: In a coherent plan, each assembled part will touch at least one previously assembled part, whereas in incoherent plans parts can be placed anywhere, without touching another part. We assume the plans can be coherent or incoherent.

3. The SPP-Flex algorithm

In this section, we present the proposed Assembly Sequence and Path Planner for Rigid and Flexible parts, named SPP-Flex, which adopts a greedy heuristic approach to assembly sequence planning and a stochastic approach to assembly path planning. The overview of the method is as follows:

First, based on the parts' precedence relations, geometrical dimensions, material, density, Young's modulus, coefficient of friction, as well as the types of elements used for modeling the parts' deformation behavior and the direction of exerted forces, an Assembly Stress Matrix (ASM) is constructed using the Abaqus™ software that contains all applied stress to any two parts along four (in 2D) or six (in 3D) main rectilinear directions ($\pm x$, $\pm y$, $\pm z$). Then through the heuristic procedure `GH_Sequence_Planning()`, an assembly plan (a permutation of parts and assembly directions) is generated incrementally which in each step has minimal (could be nonzero) stress of parts in the ASM. The initial and final configurations of the part (q_{init} and q_{goal}, respectively) are then calculated based on the assembly sequence plan in subroutine `Initial_and_Goal_Configurations()`. Afterward, starting from the first part in the sequence and using a sampling-based path planning method, a short and collision-free assembly path $\boldsymbol{\tau}_i$ is planned for the active part p_i from its q_{init} to q_{goal} in subroutine `BXXT_Path_Planning()`.

If a path could not be found for a part after *iter*_max number of attempts (which happens in nonmonotone problems or when the assembly sequence is infeasible), we identify a previously assembled part that maximally obstructs p_i in reaching its final configuration, relocate it to a free intermediate configuration using the `Relocate()` routine, and try to replan a new feasible path for assembling p_i using the `BXXT_Path_Planning()` subroutine. If failed again, we relocate another blocking part and do path replanning for p_i. This procedure is repeated until either a collision-free path is found for assembling p_i or no path is found after max_*relocations* number of attempts. In the latter case, the movement of the part p_i is simulated from its last position to the goal configuration along the direction d_i using the Abaqus™ software in the `FEA_Test()` routine, which is a professional finite element analysis and computer-aided engineering tool. In this routine, moving the part p_i continues until either it reaches its final position in the assembly by tightly passing through blocking objects due to elastic deformation, or at least one of the previously assembled parts or the part p_i itself enters its plastic deformation region, in which case the planner reports no path is found.

Whenever a valid path is found for a part, it is smoothened using the procedure `Smoothen_Path()` and the array *Plan* is updated by concatenating the number and assembly path of the part just moved. The whole procedure is repeated for all parts until either the assembly is complete or at some point, no path can be found for a certain part. The overall solution to the problem is then reported as the row array of *Plan*. The main algorithm of the SPP-Flex is presented in Fig. A.1 in the Appendix.

Other attributes of the SPP-Flex method are as follows:

– *Scope*: Its scope is a combination of global and local planning and benefits from the advantages of both. In the ASP subproblem, the planner uses the partial (local) information of the parts' interference matrix, and in the APP subproblem, it uses the global information about the C-space (while not computing it explicitly) to sample collision-free configurations and connect them locally to form a random tree.
– *Completeness*: Because of implementing sampling-based path planners, the proposed SPP-Flex is *probabilistically complete*, which means it guarantees to find a solution to the problem if its available processing time approaches infinity and report failure if no feasible solution exists. Probabilistic completeness is an inherent property of sampling-based or stochastic search methods.

The following subsections describe each component of the algorithm in detail.

3.1. The assembly sequence planning component

A solution to the assembly sequence planning is a permutative sequence of ordered pairs of parts with the direction of their assembly operation, in the form of $AS = \langle(\pi_1, \delta_1), \ldots, (\pi_n, \delta_n)\rangle$. π_i denotes the i th part that is assembled and δ_i is the direction of the assembly operation of part π_i which takes a value among $d_k = \{-x, +x, -y, +y, -z, +z\}$ $(k = 1, \ldots, 6)$. In order to compute the AS, we need to form some matrices and variables defined as follows:

– $bb(\pi_i)$: *Bounding box* (a rectangle in 2D or a rectanguloid in 3D) that bounds all the parts $\{\pi_1, \pi_2, \ldots, \pi_{i-1}\}$ assembled prior to the (current) active part π_i and has axes aligned with the main Cartesian directions x, y, and z.
– av_i^k: The *assembly vector* of part p_i along direction d_k, which starts from the center of mass of the part in its initial position and extends toward the final position of that point in the final assembly. The way a part's initial position in any direction is determined is discussed at the end of this section.

Fig. 1 illustrates the Bounding Box and Assembly Vector concepts.

– *ASM*: The *Assembly Stress Matrix* is defined as an $n\times(n\bullet k)$ matrix for all pairs of parts p_i and p_j $(i = 1, \ldots, n;\ j = 1, \ldots, n)$ and for all directions $d_k \in\{-x, +x, -y, +y\}$ for 2D parts and $d_k \in\{-x, +x, -y, +y, -z, +z\}$ for 3D parts, in the following form:

$$ASM = \begin{array}{c} \overbrace{\begin{array}{cccccc} -x & +x & -y & +y & -z & +z \end{array}}^{p_1} \quad \cdots \quad \overbrace{\begin{array}{cccccc} -x & +x & -y & +y & -z & +z \end{array}}^{p_n} \\ \left[\begin{array}{cccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & \cdots & s_{1,n}^1 & s_{1,n}^2 & s_{1,n}^3 & s_{1,n}^4 & s_{1,n}^5 & s_{1,n}^6 \\ s_{2,1}^1 & s_{2,1}^2 & s_{2,1}^3 & s_{2,1}^4 & s_{2,1}^5 & s_{2,1}^6 & \cdots & s_{2,n}^1 & s_{2,n}^2 & s_{2,n}^3 & s_{2,n}^4 & s_{2,n}^5 & s_{2,n}^6 \\ & & \vdots & & & & - & & & \vdots & & & \\ s_{n,1}^1 & s_{n,1}^2 & s_{n,1}^3 & s_{n,1}^4 & s_{n,1}^5 & s_{n,1}^6 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \tag{1}$$

For any combination of p_i, p_j, and d_k, the element $s_{i,j}^k$ of the ASM matrix is constructed as follows:

(1) If the part p_j does not block the movement of part p_i along d_k, then we set $s_{i,j}^k = 0$.
(2) If both parts are rigid and the part p_j blocks the movement of part p_i along d_k, then we set $s_{i,j}^k = M$, where M is a large positive number greater than the maximum stress that can be exerted by the assembler.
(3) If the part p_j blocks the movement of part p_i along d_k and at least one of the parts is flexible, then their assembly operation is simulated in the Abaqus™ software to verify if the blockage can be resolved due to the flexibility of the part(s). If so, $s_{i,j}^k$ is set to the sum of the maximum stress (in MPa) exerted on parts p_i and p_j during the process of assembling part p_i along direction d_k after part p_j has been assembled.
(4) If the Abaqus decides that the blockage cannot be resolved despite considering the flexibility of the parts, then we set $s_{i,j}^k = M$.

It should be noted that the blockage relation between parts can be extracted using an existing CAD assembly model. For instance, in [54] the geometries and interrelations of parts in an assembly creating in CAD were used to determine if incrementally assembling parts would maintain the stability of the assembly. In SPP-Flex, for each combination of p_i, p_j and d_k, we check whether part p_j blocks the movement of part p_i along d_k in the CAD model. Based on the 'Assembly by Disassembly' strategy, an assembly plan is obtained by disassembling a whole product into its constituting parts and then reversing the order of disassembly. Since for two rigid parts, we only consider the geometric constraints, there is a bijection between assembly and disassembly sequences and paths. Therefore, we first examine the disassembly of the parts and then determine the blockage relationship between the parts in the assembly mode, in the opposite direction of disassembly. For this purpose, we deactivate all the assembly parts except the two parts p_i and p_j, and then move part p_i along the direction $-d_k$ to be transferred to a point outside the bounding box of the whole assembly. Then we check if part p_i will collide with part p_j while moving. In case of collision, it is concluded that the part p_j blocks the movement of part p_i along d_k.

– $ds_i^k = \sum_{j=1}^n s_{j,i}^k$: *Directional Stress* of a part p_i along direction d_k, which is equal to the sum of applied stresses to all other parts if assembled along d_k, supposing that part p_i is already assembled and may block subsequent parts (hence the order of the subscripts in the notation is switched).
– *DASM*: The *Directional Assembly Stress Matrix* is defined as a size $n \times k$ matrix of ds_i^k for all parts p_i and for all directions d_k ($k = 4$ for 2D parts and $k = 6$ for 3D parts), in the following form:

$$DASM = \begin{array}{c} \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{array} \begin{array}{c} \begin{array}{cccccc} -x & +x & -y & +y & -z & +z \end{array} \\ \left[\begin{array}{cccccc} ds_1^1 & ds_1^2 & ds_1^3 & ds_1^4 & ds_1^5 & ds_1^6 \\ ds_2^1 & ds_2^2 & ds_2^3 & ds_2^4 & ds_2^5 & ds_2^6 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ ds_n^1 & ds_n^2 & ds_n^3 & ds_n^4 & ds_n^5 & ds_n^6 \end{array}\right] \end{array} \tag{2}$$

$$ASM = \begin{array}{c|cccc|cccc|cccc|cccc|cccc|cccc|}
 & \multicolumn{4}{c|}{p_1} & \multicolumn{4}{c|}{p_2} & \multicolumn{4}{c|}{p_3} & \multicolumn{4}{c|}{p_4} & \multicolumn{4}{c|}{p_5} & \multicolumn{4}{c|}{p_6} \\
 & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y \\
\hline
p_1 & 0 & 0 & 0 & 0 & M & 53 & M & M & 0 & M & M & 0 & 0 & 0 & M & 0 & 0 & 0 & M & 0 & 0 & 0 & M & 0 \\
p_2 & 49 & M & M & M & 0 & 0 & 0 & 0 & 0 & M & M & M & 0 & M & M & 0 & 0 & M & M & M & 0 & M & M & 0 \\
p_3 & M & 0 & 0 & M & M & 0 & M & M & 0 & 0 & 0 & 0 & M & 0 & M & M & M & 0 & M & 0 & M & 0 & M & 0 \\
p_4 & 0 & 0 & 0 & M & M & 0 & 0 & M & 0 & M & M & M & 0 & 0 & 0 & 0 & M & 0 & 0 & M & M & 0 & M & 0 \\
p_5 & 0 & 0 & 0 & M & M & 0 & M & M & 0 & M & 0 & M & 0 & M & M & 0 & 0 & 0 & 0 & 0 & M & 0 & M & 69 \\
p_6 & 0 & 0 & 0 & M & M & 0 & 0 & M & 0 & M & 0 & M & 0 & M & 0 & M & 0 & M & 62 & M & 0 & 0 & 0 & 0
\end{array}$$

$$DASM = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49+M & M & M & 5M \\
p_2 & 5M & 53 & 3M & 5M \\
p_3 & 0 & 5M & 3M & 4M \\
p_4 & M & 3M & 4M & 2M \\
p_5 & 2M & 2M & 62+3M & 3M \\
p_6 & 3M & M & 5M & 69
\end{array}, \quad \pi_1 = \arg\min_i \{ds_i^k\} = \begin{cases} 3 & \text{if } \delta_1 = -x \\ 2 & \text{if } \delta_1 = +x \\ 1 & \text{if } \delta_1 = -y \\ 6 & \text{if } \delta_1 = +y \end{cases}$$

$k = 1$ is randomly selected and so $(\pi_1, \delta_1) = (3, -x)$.

$$AAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 0 & M & M & 0 \\
p_2 & 0 & M & M & M \\
p_4 & 0 & M & M & M \\
p_5 & 0 & M & 0 & M \\
p_6 & 0 & M & 0 & M
\end{array}, \quad UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & M & M & 4M \\
p_2 & 4M & 53 & 2M & 4M \\
p_4 & 0 & 3M & 3M & M \\
p_5 & M & 2M & 62+2M & 3M \\
p_6 & 2M & M & 4M & 69
\end{array}, \quad AAS+UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & 2M & 2M & 4M \\
p_2 & 4M & 53+M & 3M & 5M \\
p_4 & 0 & 4M & 4M & 2M \\
p_5 & M & 3M & 62+2M & 4M \\
p_6 & 2M & 2M & 4M & 69+M
\end{array}$$

Assembling p_4 along $-x$ has the minimum stress and so $(\pi_2, \delta_2) = (4, -x)$.

$$AAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 0 & M & 2M & 0 \\
p_2 & 0 & 2M & 2M & M \\
p_5 & 0 & M & M & M \\
p_6 & 0 & 2M & 0 & 2M
\end{array}, \quad UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & M & M & 3M \\
p_2 & 3M & 53 & 2M & 3M \\
p_5 & 0 & 2M & 62+2M & 2M \\
p_6 & M & M & 3M & 69
\end{array}, \quad AAS+UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & 2M & 3M & 3M \\
p_2 & 3M & 53+2M & 4M & 4M \\
p_5 & 0 & 3M & 62+3M & 3M \\
p_6 & M & 3M & 3M & 69+2M
\end{array}$$

Assembling p_5 along $-x$ has the minimum stress and so $(\pi_3, \delta_3) = (5, -x)$.

$$AAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 0 & M & 3M & 0 \\
p_2 & 0 & 3M & 3M & 2M \\
p_6 & 0 & 3M & 62 & 3M
\end{array}, \quad UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & M & M & 2M \\
p_2 & 2M & 53 & M & 2M \\
p_6 & 0 & M & 2M & 0
\end{array}, \quad AAS+UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & 2M & 4M & 2M \\
p_2 & 2M & 53+3M & 4M & 4M \\
p_6 & 0 & 4M & 62+2M & 3M
\end{array}$$

Assembling p_6 along $-x$ has the minimum stress and so $(\pi_4, \delta_4) = (6, -x)$.

$$AAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 0 & M & 4M & 0 \\
p_2 & 0 & 4M & 4M & 2M
\end{array}, \quad UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & M & M & M \\
p_2 & M & 53 & M & M
\end{array}, \quad AAS+UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 49 & 2M & 5M & M \\
p_2 & M & 53+4M & 5M & 3M
\end{array}$$

Assembling p_1 along $-x$ has the minimum stress and so $(\pi_5, \delta_5) = (1, -x)$.

$$AAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_2 & 49 & 5M & 5M & 3M
\end{array}, \quad UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_2 & 0 & 0 & 0 & 0
\end{array}, \quad AAS+UAS = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_2 & 49 & 5M & 5M & 3M
\end{array}$$

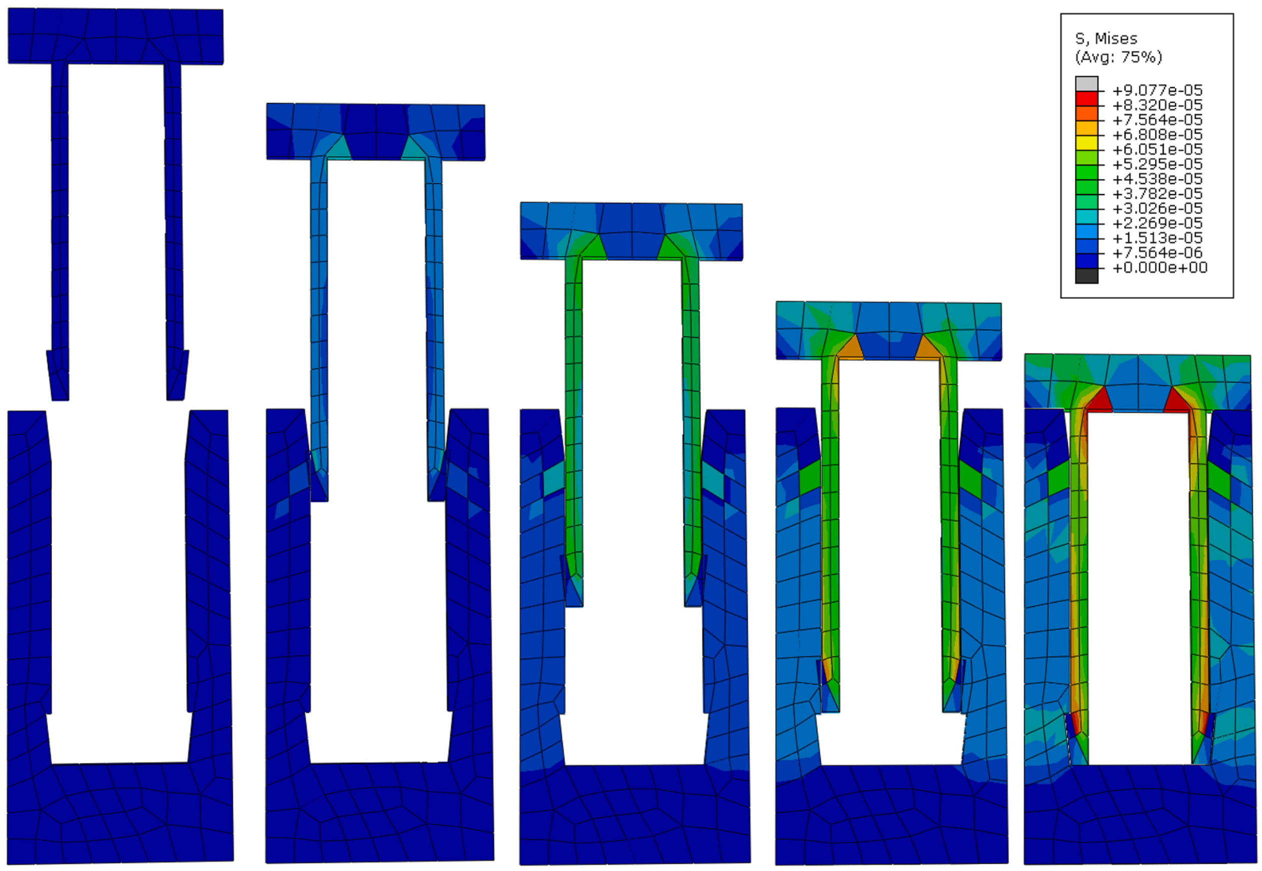
Assembling p_2 along $-x$ has the minimum stress and so $(\pi_6, \delta_6) = (2, -x)$ is selected.

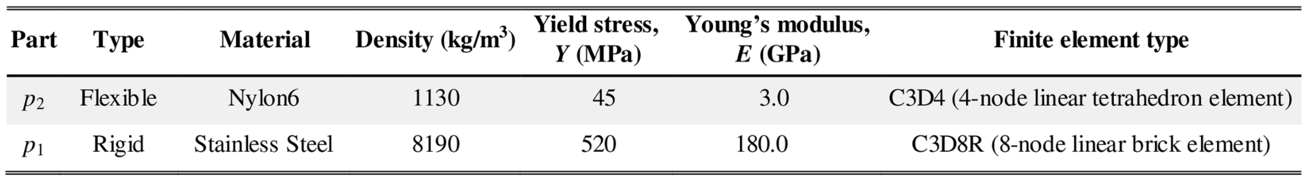
Assembly Sequence $AS = \langle(3, -x), (4, -x), (5, -x), (6, -x), (1, -x), (2, -x)\rangle$

Fig. 2. An example of generating an assembly sequence plan for a given ASM matrix. M represents a large stress beyond the elastic deformation zones of the parts.

Table 3
Main modules of the Abaqus™ FEA software used for verifying the possibility of assembling flexible parts.

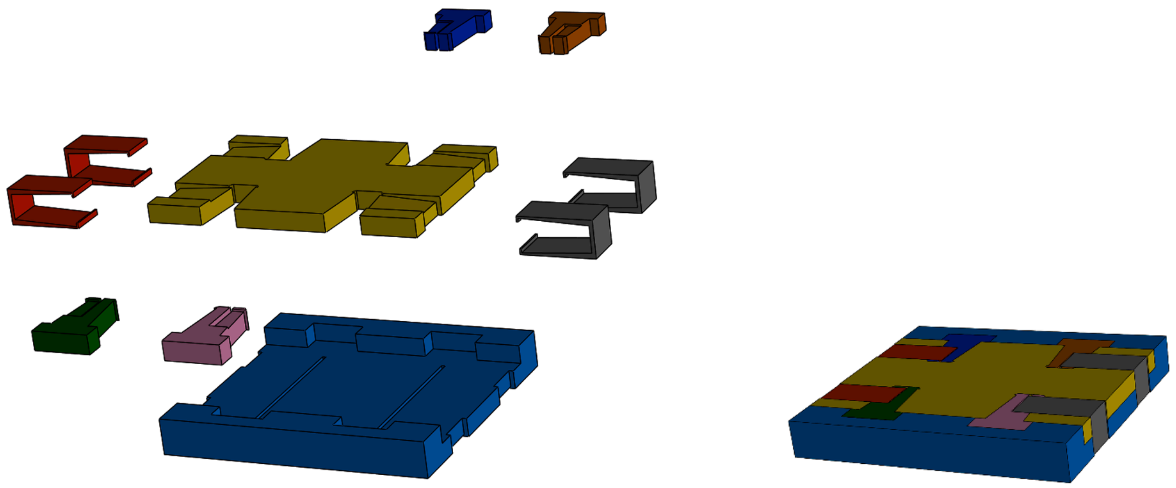
Module Name	Operation
Part module	Creates or inputs the geometrical features of all the flexible and rigid parts
Property module	Assigns physical (e.g., density) and mechanical (e.g., Young's modulus, Poisson's ratio, yield stress) properties of the parts
Assembly module	Locates the assembled and active parts at their last configurations
Step module	Sets the static and general conditions of the parts in the 'Initial step' and applies dynamic forces and motions to the active part in the 'Dynamic step'
Interaction module	Adds all surface-surface contacts and interactions between the parts to the initial step
Load module	Applies the following affecting forces: (1) body force exerted by the assembler on the active part, (2) friction forces along contact surfaces between the active part and other objects, (3) the gravity force applied to all objects along direction –*y*. Also, it applies boundary conditions to all other parts to constrain their displacement and rotation while the active part is moving
Mesh module	Forms seeds and meshes for the flexible parts (hexahedron-shaped) and rigid objects (tetrahedron-shaped)
Job module	Creates a finite element analysis job of types: Full analysis, Recover, and Restart. We used the Full analysis option





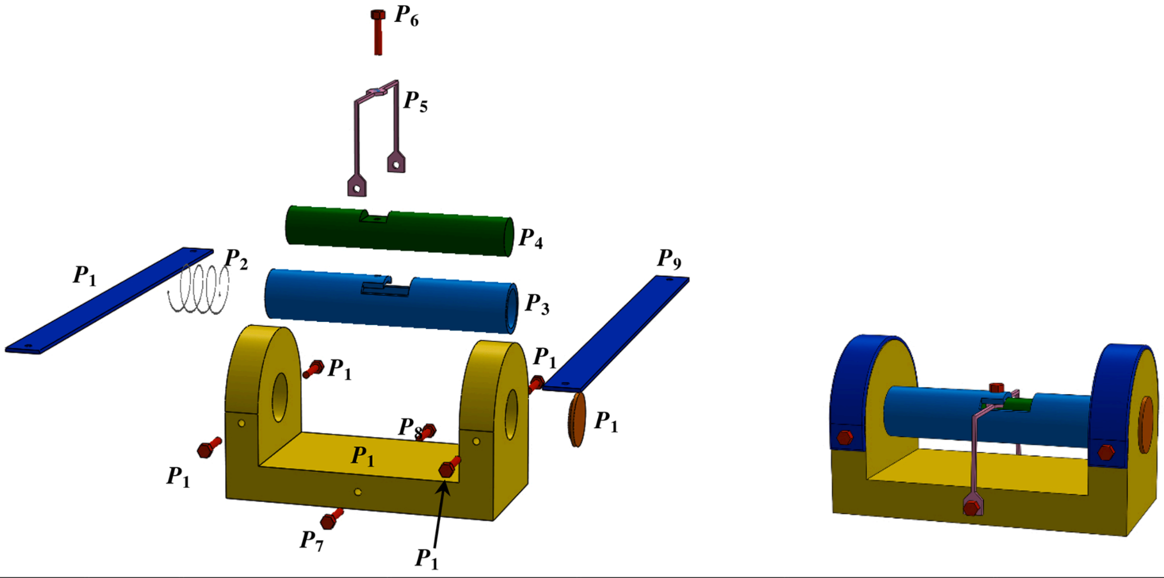
Part	Type	Material	Density (kg/m^3)	Yield stress, *Y* (MPa)	Young's modulus, *E* (GPa)	Finite element type
p_2	Flexible	Nylon6	1130	45	3.0	C3D4 (4-node linear tetrahedron element)
p_1	Rigid	Stainless Steel	8190	520	180.0	C3D8R (8-node linear brick element)

Fig. 3. (*From left to right*) Simulation snapshots of assembling the flexible part p_1 (top) toward its goal position inside the rigid part p_2 along the direction –*y*. The coefficient of friction between the surfaces is 0.35.



Part	Type	Material	Density (kg/m^3)	Yield stress, Y (MPa)	Young's modulus, E (GPa)	Finite element type
p_1	Rigid	Aluminum	2700	240	68.9	C3D4 (4-node linear tetrahedron)
p_{10}		Stainless Steel	8190	520	180.0	
p_2, p_3, p_4, p_5	Flexible	Nylon6	1130	45	3.0	
p_6, p_7, p_8, p_9		PVC	1300	52	3.3	

Fig. 4. The Compact Box assembly with two rigid (parts p_1 and p_{10}) and eight flexible parts. The coefficients of friction between rigid-rigid, rigid-flexible, and flexible-flexible contacts are 0.1, 0.1, and 0.05, respectively.



Part	Type	Material	Density (kg/m^3)	Yield stress, Y (MPa)	Young's modulus, E (GPa)	Finite element type
p_1, p_{15}	Rigid	Aluminum	2700	240.0	68.9	C3D4 (4-node linear tetrahedron)
$p_3, p_4, p_6, p_7, p_8,$ $p_{10}, p_{11}, p_{13}, p_{14}$		Stainless Steel	8190	520.0	180.0	
p_2	Flexible	Stainless Steel	8190	520.0	180.0	
p_5		Copper	8960	33.3	120.0	
p_9, p_{12}		Rubber	1522	17.5	0.001	C3D8R (8-node linear brick)

Fig. 5. Shaft Housing assembly with 11 rigid and 4 flexible (parts $p_2, p_5, p_9,$ and p_{12}) parts. The coefficients of friction between rigid-rigid, rigid-flexible, and flexible-flexible contacts are 0.1, 0.1, and 0.05, respectively.

Table 4
Mean and standard deviation values of the performance criteria produced by running the SPP-Flex 20 times for each scenario and each product.

Scenarios	Criteria																			
	Compact Box Assembly										Shaft Housing Assembly									
	TPL		TNN		TNE		TCC		TT		TPL		TNN		TNE		TCC		TT	
	M	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
GH+BXXT	1136.1	52.1	113.3	7.4	89.6	4.4	1935.3	109.2	986.5	53.4	987.3	62.5	179.8	10.2	132.6	8.1	3417.9	193.6	1456.3	68.2
GH+BRRT	1278.9	75.7	154.2	8.8	127.6	6.6	2134.9	131.2	1254.6	67.6	1123.8	69.7	213.4	11.9	157.9	8.1	3911.0	219.7	1679.2	83.8
BLS+BXXT	1139.2	71.2	112.4	6.3	87.3	5.6	1931.2	104.8	1175.4	56.2	985.2	57.2	184.2	10.5	137.9	7.2	3429.1	185.9	1649.9	99.1
BLS+BRRT	1302.8	77.8	151.9	6.7	126.8	7.3	2129.0	126.6	1461.5	70.0	1145.6	67.2	218.9	12.6	163.8	9.9	4011.1	231.5	1879.3	110.3

Now we can present the method of planning the assembly sequence using the above nomenclature. To do so, we implement a greedy heuristic algorithm that gives higher assembly priority to a part for which the sum of the stress needed for assembling it into the existing assembly (*AAS*) and the stress needed for assembling all subsequent parts (*UAS*) is minimal. In this way, the probability of creating a 'more feasible' solution increases significantly. Note that we do not consider the workspace obstacles in this stage. The proposed greedy heuristic algorithm is coded as a subroutine called `GH_Sequence_Planning()` and is described in Fig. A.2.

Fig. 2 demonstrates the procedure of obtaining an assembly sequence from a given ASM matrix.

It is noted that since the parts to be assembled next are added sequentially, the algorithm is greedy and thus there is no guarantee to come up with an optimal (and even feasible) sequence. In fact, since the ASP problem is NP-hard [47], finding a feasible solution may require exponential time in the worst case, and that is why most researchers attempt to solve it using metaheuristic methods. In our approach, however, instead of spending time on finding a feasible sequence, we accept the output assembly sequence (which is still better than random sequences) and proceed to the APP component. However, due to its replanning capability, the APP component will cope with any infeasible sequence by temporarily relocating blocking parts (that create infeasibility) to intermediate positions and returning to their final locations after assembling all other necessary parts.

3.2. Determining the initial and goal configurations

Once the output of the ASP component is obtained in the form of $AS = \langle(\pi_1, \delta_1), \ldots, (\pi_n, \delta_n)\rangle$, the initial (disassembled) configuration of each part should be calculated. This is done using the geometrical information (*Geom_parts*) and the assembly direction of each part. In order to determine the initial position of a part π_p along its assembly direction δ_p, we need to project a hypothetical ray from the center of mass (CoM) of the part in its final (assembled) position along the reverse direction of δ_p. Since the bounding box is aligned with the main Cartesian directions, this ray will intersect and be outward normal to one of the faces of the current bounding box. Then the initial position (x, y, z) of the part is obtained by placing its center of mass on the ray with a distance of 1.5 times the part's dimension in that direction away from the bounding box (to let sufficient maneuvering space later in the path planning phase). In fact, the initial position is the start point of the assembly vector $av_{\pi_p}^{\delta_p}$. The orientations (α, β, γ) of the part about the main axes are assumed to be zero. As a result, the initial configuration of a 3-dimensional part π_p will be the six-tuple:

$$q_{init}(\pi_p) = \left(x.start\left(av_{\pi_p}^{\delta_p}\right),\ y.start\left(av_{\pi_p}^{\delta_p}\right),\ z.start\left(av_{\pi_p}^{\delta_p}\right),\ 0,\ 0,\ 0\right), \tag{3}$$

and the final (assembled) configuration of the part is simply the coordinates of the part's center of mass in its assembled position, or in other words, it is the endpoint of the assembly vector $av_{\pi_p}^{\delta_p}$:

$$q_{goal}(\pi_p) = \left(x.end\left(av_{\pi_p}^{\delta_p}\right),\ y.end\left(av_{\pi_p}^{\delta_p}\right),\ z.end\left(av_{\pi_p}^{\delta_p}\right),\ 0,\ 0,\ 0\right). \tag{4}$$

The initial and goal configurations of all parts are determined in the subroutine `Initial_and_Goal_Configurations()` (line 6 in Fig. A.1).

3.3. The assembly path planning component

Considering the advantages of Sampling-based path planners (as mentioned in Section 1.2) we propose a new variation of the RRT method which is adapted to the context of assembly planning and incorporated into the general SPP-Flex method for both path planning and replanning. We name the new planner as BXXT.

For finding the assembly path of an active (selected) part π_p, the BXXT uses two strategies to search the C-space: *Exploitation* and *Exploration*. The planner is a bidirectional search method which forms two trees called T_{active} and $T_{passive}$ initially rooted in $q_{\rm init}(\pi_p)$ and $q_{\rm goal}(\pi_p)$, respectively (as indicated in lines 2–3 of the pseudocode in Fig. A.3), and grows them toward each other until either they get connected or a predefined number of iterations is exceeded without success. Firstly, in the Exploitation phase, the node $q_{\rm init}(\pi_p)$ is attempted to be directly connected to $q_{\rm goal}(\pi_p)$ through a hypothetical straight line in the C-space, which is interpolated into a number of equidistant intermediate nodes *step_size* apart (lines 6–8). If all these nodes are in C_{free} (i.e., the obstacle-free zones of the C-space), then the nodes and in-between edges are added to T_{active}, and the T_{active} and $T_{passive}$ trees are connected (lines 10–17), meaning that the part can move from its initial (unassembled) position to its final (assembled) position without any collision with other objects, in which case the path planning is completed (lines 19–21). Otherwise, starting from the root of T_{active}, the algorithm adds the interpolated nodes and their in-between edges to the T_{active} until an obstruction is encountered. Next, the Exploration phase starts with selecting a random configuration $q_{\rm rand}$ in the C_{free} and finding a node $n_{\rm close}$ on T_{active} which is nearest to $q_{\rm rand}$ (lines 22–24), and then expanding the T_{active} directly from $n_{\rm close}$ toward $q_{\rm rand}$ through interpolation and augmentation of new intermediate nodes and edges until either $q_{\rm rand}$ is reached or an obstruction is encountered (lines 25–34).

At this stage, the T_{active} and $T_{passive}$ are switched, meaning that T_{active} becomes the tree rooted in $q_{\rm goal}(\pi_p)$ and $T_{passive}$ becomes the tree rooted in $q_{\rm init}(\pi_p)$ (line 39). Now the Exploitation phase is relaunched, meaning that a random node $n_{\rm rand}$ is selected on the new $T_{passive}$ as a (temporary) goal point and the nearest node on T_{active} (called $n_{\rm near}$) to $n_{\rm rand}$ is found (lines 6–7). Then, T_{active} is expanded from $n_{\rm near}$ to $n_{\rm rand}$ by interpolation and node augmentation until either $n_{\rm rand}$ is reached (in which case the path planning ends) or an obstruction is encountered (lines 8–17). In the latter case, the Exploration phase is executed, which is explained in the previous paragraph. Again the T_{active} and $T_{passive}$ trees are switched and successive Exploitation and Exploration phases are executed until either the two trees are connected or *iter*_max iterations have passed. In that former case, the path from $q_{\rm init}(\pi_p)$ to $q_{\rm goal}(\pi_p)$ is reported as the assembly path $\boldsymbol{\tau}(\pi_p)$ of part π_p, and in the latter case, the path connecting

$q_{\text{init}}(\pi_p)$ to a node on its rooted tree which is nearest to $q_{\text{goal}}(\pi_p)$ (called $q_{\text{intermediate}}$) is returned as the path $\boldsymbol{\tau}(\pi_p)$ of part π_p (lines 41–42). In both cases, the resulting path is smoothed as is customary in random tree methods [12] (line 43). In fact, defining $q_{\text{intermediate}}$ makes the SPP-Flex algorithm capable of solving nonmonotone ASPP problems as well.

As indicated in the pseudocode in Fig. A.3, in addition to generating the assembly path of the p-th part $\boldsymbol{\tau}(\pi_p)$, another output of the path planner is the *Intersections Count Array* $Q_p = [\pi_1, \pi_2, \ldots, \pi_{p-1} \mid o_1, o_2, \ldots, o_m]$, initially having all zeros (m is the number of obstacles in the workspace). As mentioned above (and in the pseudocode), a rooted tree (T_{active}) is expanded from an existing node (n_{near} or n_{close}) toward a random configuration (n_{rand} or q_{rand}) by successively adding the free configurations on their hypothetical connecting line until a configuration q_j is found to be in collision with an already assembled part or a static obstacle like a fixture. The collision checking procedure is executed in the `In_Collision()` subroutine, which checks if any vertex, edge, or face of the active part π_p at configuration q_j intersects with any other vertex, edge, or face in the workspace. In the case of an intersection, the colliding objects (parts or obstacles) are added to the set *CO*. The array Q_p is then updated via incrementing by 1 any of its elements that correspond to an assembled part in *CO*. In this way, Q_p is updated repeatedly until the path planning algorithm terminates, yielding an assembly path. The final Q_p, therefore, shows the number of times each previously assembled part ($p-1$ parts in total) and workspace obstacle (m in total) has obstructed the growth of the random tree of the p-th part being assembled. For instance, $Q_6 = [34, 6, 27, 78, 41 \mid 12, 0, 47]$ shows the number of times the five already assembled parts π_1, π_2, π_3, π_4, π_5 and the existing workspace obstacles o_1, o_2, o_3 intersected the configurations generated for planning the path of part π_6, indicating that the fourth assembled part has had the most collisions.

3.4. Identifying and relocating blocking parts

The information stored in the Intersections Count Array Q_p is useful when the path planner fails to find a start-to-goal assembly path and it is required to relocate a previously assembled part to a temporary position to allow assembling of the current part. In fact, the part with the highest value in Q_p (let us name it π_b) is the first candidate for relocation as its excessive intersection implies that it is very close to the active part and on the way of its assembly path.

Upon identifying the blocking part π_b, the algorithm tries to move it to another free configuration in C-space and eliminate its interference with the active part. The new location of π_b is determined by constructing a tree T_{relocate} rooted in $q_{\text{goal}}(\pi_b)$ and iteratively expanding it toward a random free configuration q_{rand} (again by adding interpolated free nodes). This process is repeated as many as max*_removal_attempts* times, after which among the nodes of the T_{relocate}, the one with the largest distance (difference) from $q_{\text{goal}}(\pi_b)$ is found (q_{far}). Now the part π_b can be safely removed from its current position to the configuration q_{far}. Once π_b is relocated, it is added to a list called *replanning_list* to remind us that it must be removed back to its final position after assembling the active part π_p. The pseudocode in Fig. A.4 presents the details of the above procedure.

If after relocating π_b, the active part π_p still cannot be assembled due to interference with other parts, the procedure `Relocate()` is repeated for the part with the next highest nonzero value in Q_p and the newly relocated part is included in the *replanning_list*. If within at most $p-1$ attempts (i.e., back to the first assembled part) it was possible to clear the assembly path of π_p, then we will replan the paths of all the parts in the *replanning_list* from their relocated configurations back to their final (assembled) configuration (lines 22–26 and 32–36 in Fig. A.1).

3.5. The FEA component

This component is executed when it is still not possible to assemble the active part π_p even after relocating all the previously assembled parts because of geometric interferences. Consequently, the SPP-Flex algorithm checks if part-part or obstacle-part obstructions can be resolved if the flexibility of the parts are taken into account. The main operation in this component consists of simulating the movement of the active part from its last configuration toward its goal configuration through the assembled parts and workspace obstacles and analyzing the stresses it undergoes when contacting other surfaces. This is done using the Abaqus™ software and the `FEA_Test()` in which the movement of π_p is simulated and continued until either it reaches its final configuration in the assembly (e.g., a collision-free path is found for assembling π_p) or at least one of the blocking objects or the part π_p enters into its plastic deformation zone. In that case, the SPP-Flex algorithm halts and reports that it failed to plan a collision-free assembly path for π_p and therefore the whole assembly job is not doable. The simulation of the `FEA_Test()` routine is performed via different modules of the software, as described in Table 3. The results of the FEM analysis are generated and reported by the software typically in a few minutes.

Fig. 3 shows snapshots of assembling the flexible part p_1 along the direction $-y$ after that the rigid part p_2 has been assembled. The movement is simulated in Abaqus by exerting forces to p_1 until it encounters p_2 that is constrained to be fixed. Since the inner width of the p_1 and the outer width of the p_2 form a *transition fit*, the interference between the two bodies causes tensile and compressive stresses to some elements of the parts. While the forces are maintained and the upper part tightly slides downward, the sum of stresses exerted to the flexible part at each time step (e.g., 1 second) is calculated and compared to its Yield stress. If the maximum stress along the whole movement (until p_1 reaches its goal) lies within the part's elastic zone, then it is concluded that the assembly is feasible. Otherwise, the part will undergo plastic deformation before being assembled, and thus failure is reported. In Fig. 3, the magnitudes of applied von Mises stresses on different elements of the parts are represented by a color scale (according to the contour plot legend), which ranges from dark blue, light blue, green, yellow, orange, to red for indicating the lowest to highest stresses.

The FEA component is triggered also when an active flexible part cannot reach its final configuration even if no other object is obstructing its assembly path. This happens when the part must change its shape (within its elastic deformation zone) at the final assembled position. An example is a rubber band needed to bend and twist around a subassembly to strap it, as shown in Figs. 7(l) and 7(n). In such cases, the part is translated along its assembly path until it comes to contact with its neighboring assembled part(s) in the final assembly and cannot move further despite not having reached its final configuration. Here the FEA Component is launched to simulate the elastic deformation of the part by interactively exerting forces along proper directions that lead to its shape change as required by the final assembly. It is noted that automatic planning of such deformation merely using a motion planning algorithm is extremely difficult and computationally expensive since the number of dimensions of the C-space increases dramatically as the number of moving elements of the flexible part (i.e., its degrees of freedom) increases. That is why very few works exist in the literature on motion planning of flexible objects, mostly specific to linear objects like wires and tubes.

4. Experimental results

In this section, we present the results of solving the ASPP problem by the proposed SPP-Flex method for two newly designed multipart products comprised of rigid and flexible parts. The first problem is called Compact Box assembly (Fig. 4) and is monotone, meaning that the parts of the product need to be moved directly from their initial (disassembled) position to the goal (assembled) position only once, and no intermediate positioning is required. The second problem, called Shaft Housing assembly (Fig. 5), is nonmonotone, meaning that one or more of the parts should be moved more than once, to one or more intermediate positions.

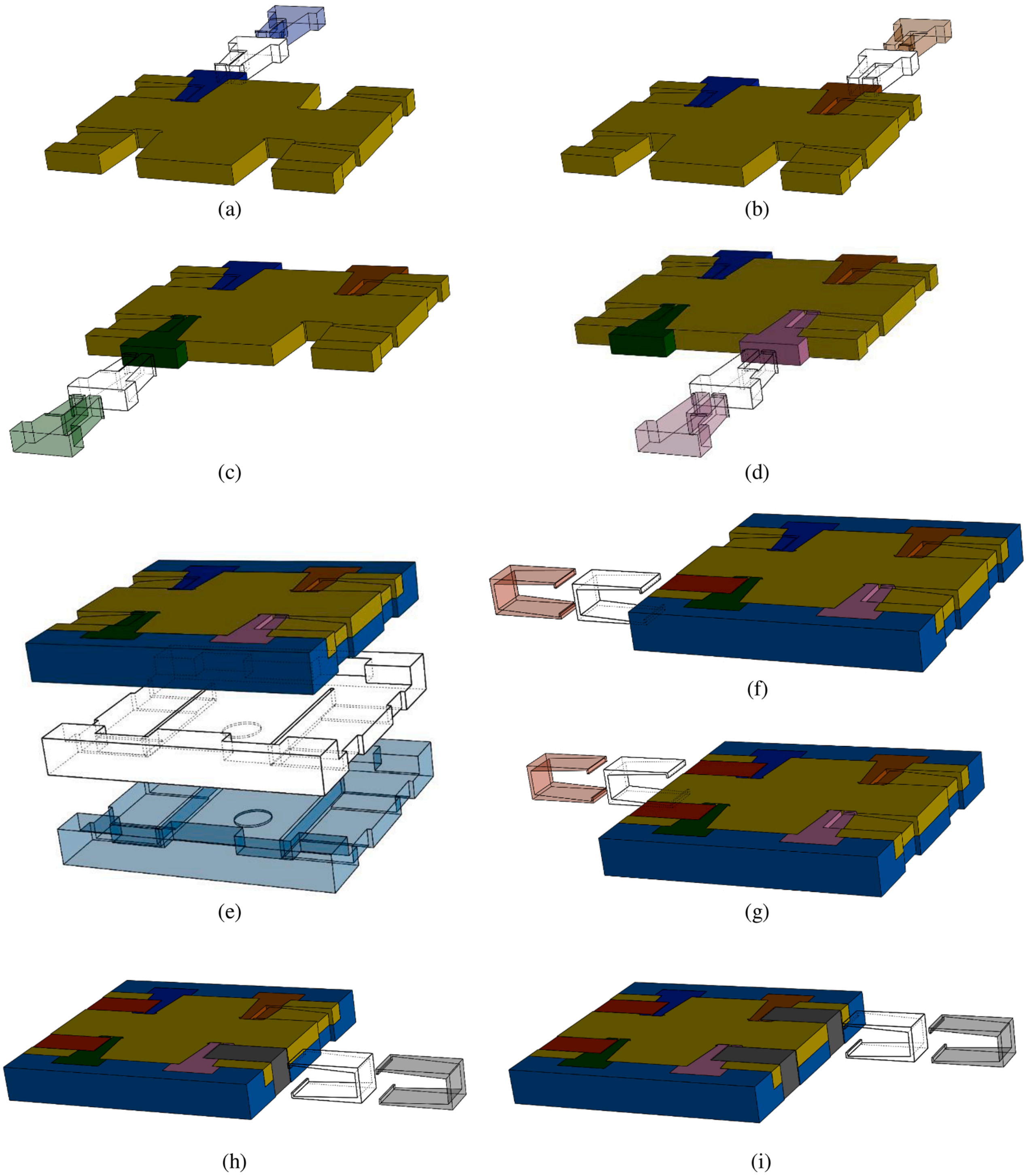
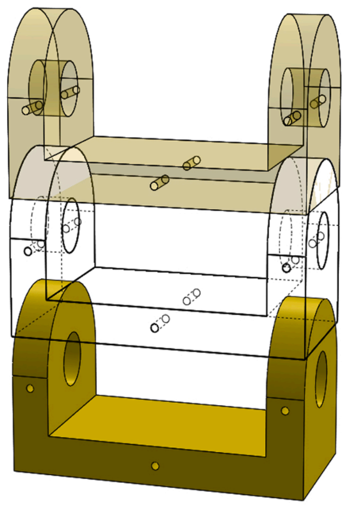
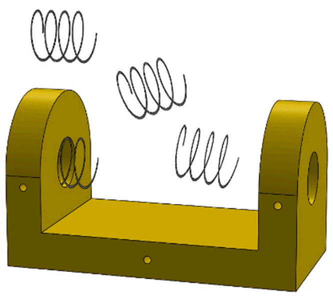
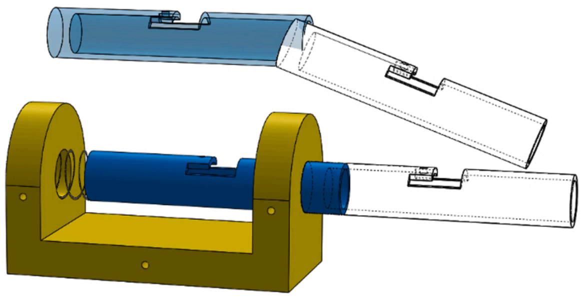
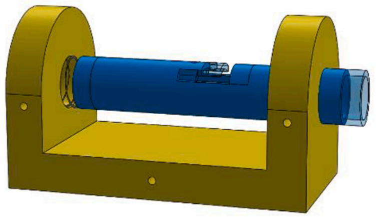


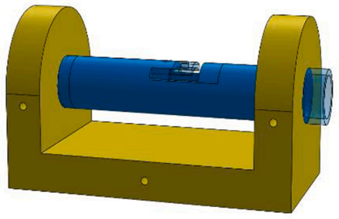
Fig. 6. Snapshots of assembly sequence and path planning of the Compact Box assembly.


(a) p_1 is assembled to its final position.


(b) p_2 is assembled to its final position.


(c) p_3 moves toward its final position but p_2 blocks it.


(d) Flexible part p_2 is deformed within its elastic deformation range to allow assembling part p_3.


(e) Flexible part p_2 is deformed within its elastic deformation range to allow assembling part p_3.

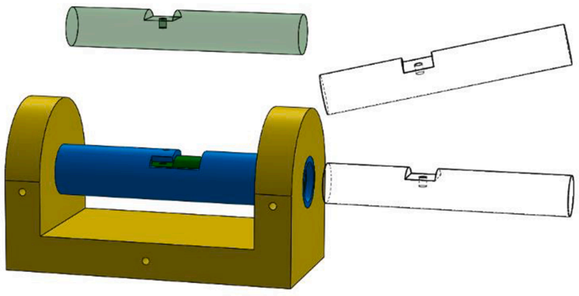
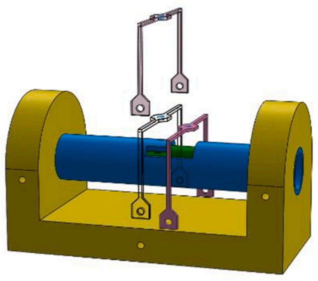

(f) p_4 is assembled to its final position

Figure 7. Snapshots of assembly sequence and path planning of the Shaft Housing assembly.


(g) p_5 moves toward its final position but p4 blocks it.

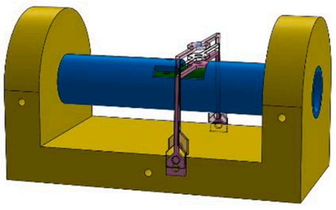
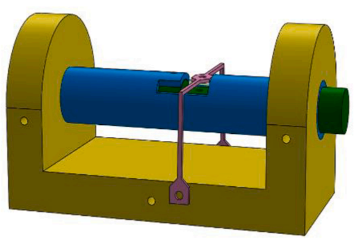
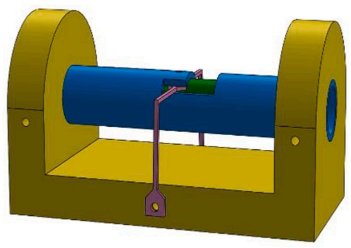

(h) Flexible part p_5 is deformed within its elastic deformation

Fig. 7. Snapshots of assembly sequence and path planning of the Shaft Housing assembly.

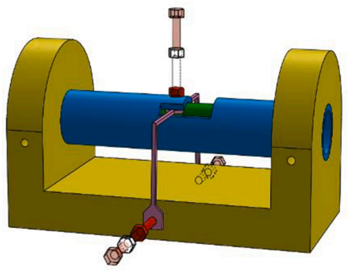
range to allow assembling it.



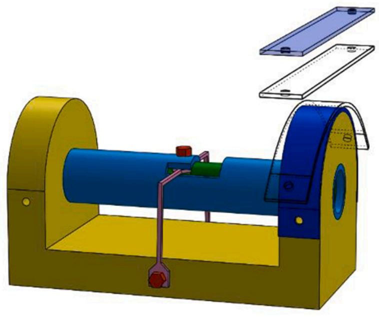
(i) p_4 moves to an intermediate position to allow assembling p5.



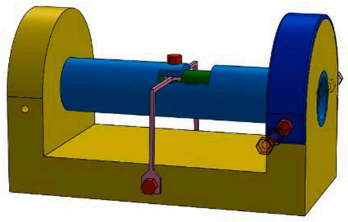
(j) p_4 and p_5 move to their final positions.



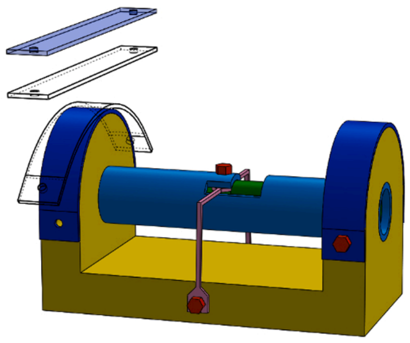
(k) p_6, p_7, and p_8 are assembled to their final positions.



(l) Flexible part p_9 is deformed and assembled to its final position.

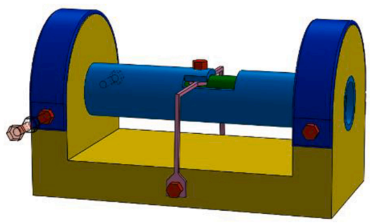


(m) p_{10} and p_{11} are assembled to their final positions.

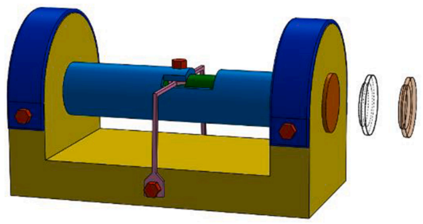


(n) Flexible part p_{12} is deformed and assembled to its final position.

Figure 7. *Continued*.



(o) p_{13} and p_{14} are assembled to their final positions.



(p) p_{15} is assembled to its final position.

Figure 7. *Continued*.

Fig. 7. (*continued*).

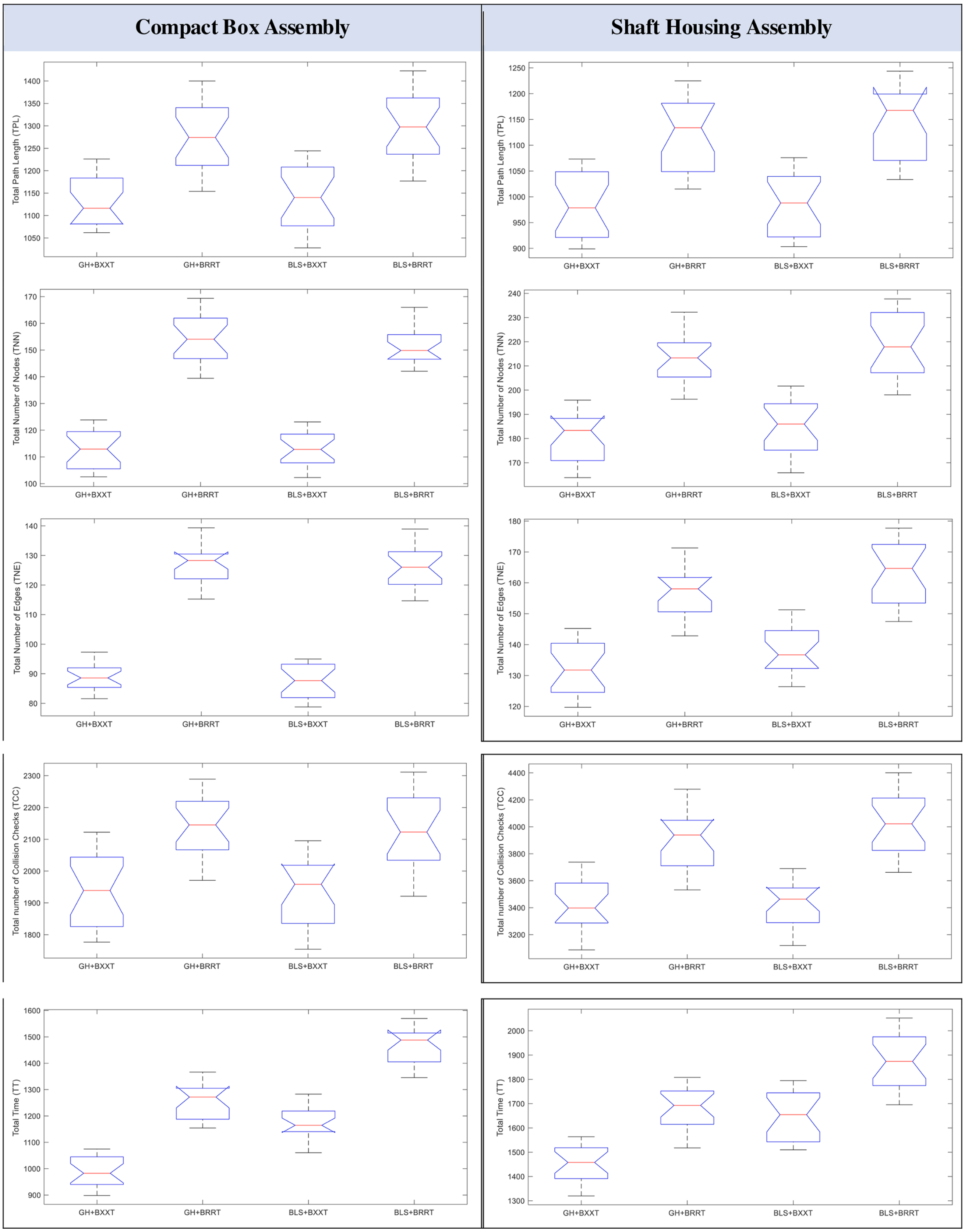


Fig. 8. Boxplots of the TPL, TNN, TNE, TCC, and TT criteria after solving the ASPP problems for the Compact Box and Shaft Housing assemblies by the compared methods 20 times each.

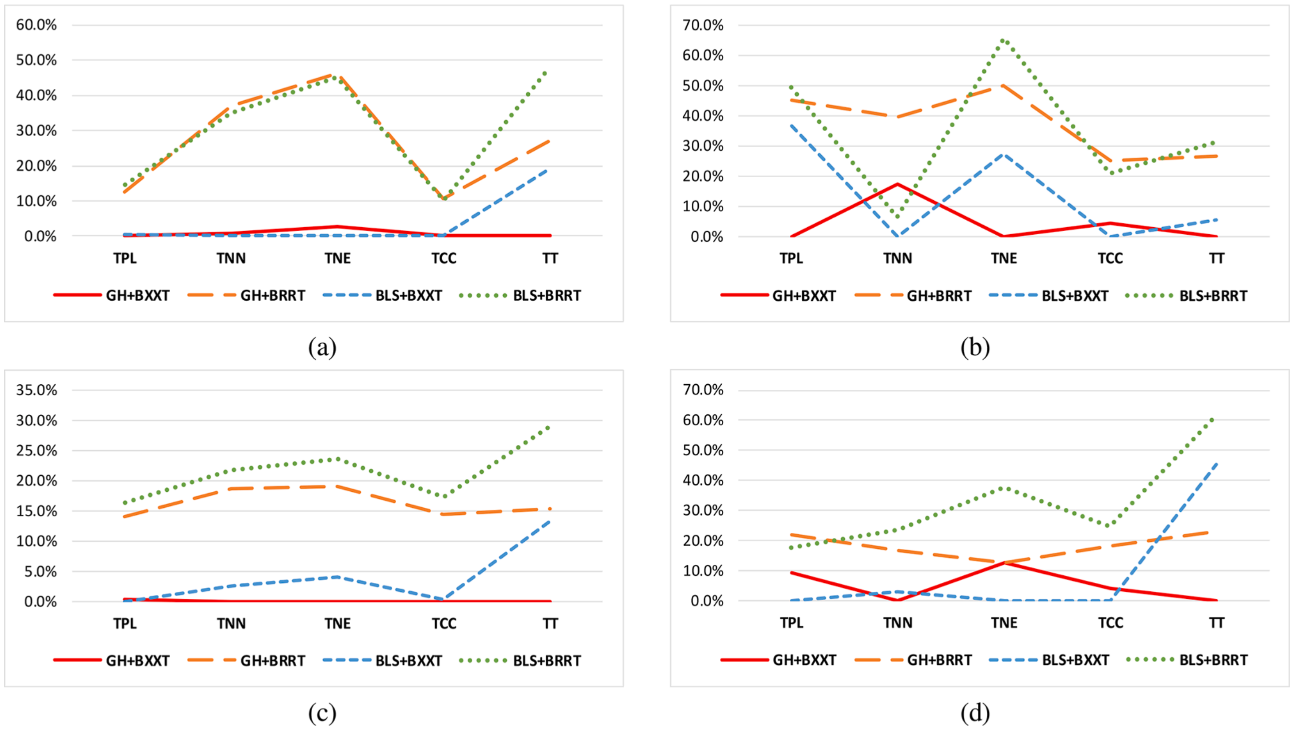


Fig. 9. Percentage of the Average Relative Gap of each scenario compared to the best among all scenarios in each of the TPL, TNN, TNE, TCC, and TT performance criteria: (a) mean of relative gaps for the Compact Box assembly, (b) standard deviation of relative gaps for the Compact Box assembly, (d) mean of relative gaps for the Shaft Housing assembly, (b) standard deviation of relative gaps for the Shaft Housing assembly.

Table 5
Percentage of the Average Absolute Gap of each scenario relative to the global best-found value in each performance criterion (TPL, TNN, TNE, TCC, and TT). The Gaps are calculated using the formulas presented in the legend.

Scenarios	Criteria									
	Compact Box Assembly					Shaft Housing Assembly				
	$\overline{Gap}_{TPL}$	$\overline{Gap}_{TNN}$	$\overline{Gap}_{TNE}$	$\overline{Gap}_{TCC}$	$\overline{Gap}_{TT}$	$\overline{Gap}_{TPL}$	$\overline{Gap}_{TNN}$	$\overline{Gap}_{TNE}$	$\overline{Gap}_{TCC}$	$\overline{Gap}_{TT}$
GH+BXXT	3.7%	3.6%	3.3%	6.8%	1.9%	4.6%	5.3%	3.4%	2.3%	2.1%
GH+BRRT	15.6%	39.1%	45.4%	13.3%	30.2%	16.8%	21.7%	22.1%	17.4%	18.3%
BLS+BXXT	3.9%	2.9%	3.3%	6.5%	22.1%	4.5%	7.9%	3.8%	3.1%	16.3%
BLS+BRRT	17.7%	37.1%	44.5%	13.0%	51.2%	19.1%	24.8%	26.6%	20.3%	32.1%

$\overline{Gap}_{TPL} = \frac{\sum_{i=1}^{k}(TPL_i - TPL_{\min})}{TPL_{min}} \times 100\%$, $\overline{Gap}_{TNN} = \frac{\sum_{i=1}^{k}(TNN_i - TNN_{\min})}{TNN_{min}} \times 100\%$, $\overline{Gap}_{TNE} = \frac{\sum_{i=1}^{k}(TNE_i - TNE_{\min})}{TNE_{min}} \times 100\%$$\overline{Gap}_{TCC} = \frac{\sum_{i=1}^{k}(TCC_i - TCC_{\min})}{TCC_{min}} \times 100\%$,$\overline{Gap}_{TT} = \frac{\sum_{i=1}^{k}(TT_i - TT_{\min})}{TT_{min}} \times 100\%$

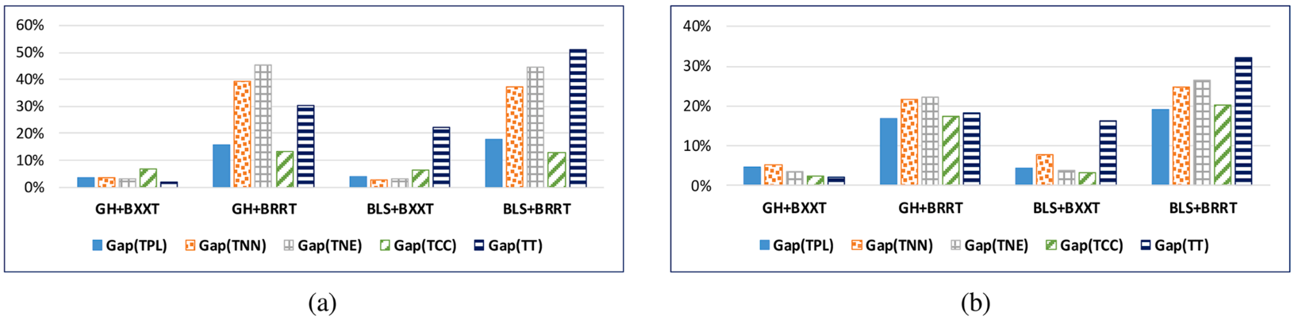


Fig. 10. Percentage of the Average Absolute Gap of each scenario relative to the global best-found value for each performance criterion (TPL, TNN, TNE, TCC, TT) for (a) the Compact Box, and (b) the Shaft Housing assemblies.

The SPP-Flex method was coded in Matlab™ and run on an Intel™ Core-i7 1.8 GHz CPU with 8 GB of RAM. Each problem was solved 20 times, and the best sequence and mean values obtained for the following five criteria were calculated:

(1) TPL: Total path length of all assembled parts from their initial positions to final assembled positions (in length units),
(2) TNN: Total number of nodes generated in the configuration (search) space (see Section 3.3 for a definition of a node),
(3) TNE: Total number of edges in the search tree or graph,
(4) TCC: Total number of collision checks in order to verify if a sampled node is collision-free. Collision check for a configuration node requires checking the intersections of vertices, edges, and faces of the part being assembled with those of all already assembled parts,
(5) TT: Total time of assembly sequence and path planning of all parts (in seconds).

As mentioned earlier, according to our literature review (Section 1.3) and to the best of our knowledge, part flexibility has not been considered for monotone or nonmonotone assembly sequence and path planning in previous related works, and thus we could not find a matching algorithm to compare with the SPP-Flex. However, in order to evaluate the effectiveness and efficiency of the algorithm and particularly its two main modules, namely, the assembly sequence and assembly path planning components, we considered two alternative methods for each of the ASP and APP components and ran the SPP-Flex with all four possible combinations (scenarios) to evaluate its performance.

The following two alternatives were considered for the ASP component of the SPP-Flex:

(a) The *Greedy Heuristic* (GH) algorithm proposed in Section 3.1,
(b) The *Breakout Local Search* (BLS) algorithm developed by us in [20] for solving the ASP problem for rigid parts only. In that work, we showed that BLS outperforms several search methods like Simulated Annealing (SA), Genetic Algorithms (GA), Memetic Algorithms (MA), Immune System-Particle Swarm Optimization hybrid method (IPSO), Harmony Search (HS), Multi-start Local Search (MLS), and Iterative Local Search (ILS). Therefore, selecting BLS as an alternative to the GH seems appropriate and reasonable. It should be noted that small modifications were made to the BLS algorithm to enable it to solve the ASP of rigid and flexible parts.

Also, the following two alternatives were considered for the APP component of the SPP-Flex:

(a) The *Bidirectional Exploration-Exploitation Tree* (BXXT) path planner proposed in Section 3.3,
(b) The *Bidirectional Rapidly exploring Random Tree* (BRRT) algorithm [35]. In BRRT, two search trees are constructed with roots on the initial and goal nodes. Through alternating expansion, the trees grow towards each other's nearest nodes with edge lengths equal to a step size until they are connected. Bidirectional searches usually outperform unidirectional searches in terms of efficiency and effectiveness, especially in problems known as 'bug trap'. For each part to be assembled, a maximum number of attempts are made to establish new branches on the search tree.

Based on the different combinations of the above ASP and APP approaches, the following four scenarios were created and investigated:

ASP	**APP**	
	Bidirectional Exploration-Exploitation Tree	**Bidirectional Rapidly-exploring Random Tree**
Greedy Heuristic	GH+BXXT	GH+BRRT
Breakout Local Search	BLS+BXXT	BLS+BRRT

Each of the four scenarios was run 20 times for solving the ASPP problem on each of the Compact Box and Shaft Housing assemblies, resulting in 160 simulations in total. Table 4 presents the mean and standard deviation of the five performance criteria (TPL, TNN, TNE, TCC, and TT) for different scenarios and products, and Figs. 6 and 7 show snapshots of assembling the Compact Box and Shaft Housing products via the GH+BXXT scenario.

4.1. *Analysis of results*

In order to analyze and interpret the obtained computational results, they are depicted in Fig. 8 as boxplots for values of the TPL, TNN, TNE, TCC, and TT criteria for all scenarios and products obtained after running each combination 20 times. For each box, the central mark the median value, the bottom and top edges represent the first and third quartiles, and the lower and upper whiskers respectively show the minimum and maximum values of 20 runs.

Fig. 8 demonstrates that the scenarios GH+BXXT and BLS+BXXT (the first and third boxes in each diagram) outperform the scenarios GH+BRRT and BLS+BRRT (the second and fourth boxes) in both solution quality and execution time criteria. The reason is obviously the better performance of the BXXT over BRRT, as it does both exploration and exploitation in the configuration space. Note that except for the path planning component, all other elements of the SPP-Flex remain the same for the two groups of scenarios. Therefore, we can conclude that the BXXT planner is quite effective and efficient compared to the BRRT.

In order to further determine the effect of the ASP component (GH or BLS) on the overall performance of the SPP-Flex, we plot two sets of data, (*i*) *average relative gap* of each scenario compared to the best among all scenarios (Fig. 9), and (*ii*) *average absolute gap* of each scenario relative to the global best-found solution (Table 5 and Fig. 10). These plots suggest that SPP-Flex is most powerful when its assembly sequence planning and assembly path planning components are selected to be the introduced Greedy Heuristic (GH) and the Bidirectional Exploration-Exploration Tree planner (BXXT), respectively.

As can be seen in Figs. 9 and 10, The GH+BRRT and BLS+BRRT scenarios were found to be the least effective. They find solutions with large gaps in all criteria, both in mean and standard deviation statistics. In addition, although the scenario BLS+BXXT yielded the best mean gap values of TNN, TNE and TCC for the Compact Box assembly, it required considerably higher runtimes compared to scenario GH+BXXT. The reason is that the BLS algorithm needs to generate several solutions in order to find a good assembly sequence with the smallest total stress among all generated solutions, which takes longer time compared to the fast Greedy Heuristic algorithm, and is also less robust since its standard deviation is larger compared to the GH+BXXT. Hence, we can make the following conclusions: (1) the GH+BXXT scenario is the best among other combinations as it provides a good trade-off for all metrics (at most 4.6% and 2.1% gaps in path length and runtime, respectively), and (2) the SPP-Flex is not sensitive to the quality of the assembly sequence it employs as the starting sequence (i.e., whether it is an optimal sequence or not) as it will be able to adjust and enhance it through the identifying and relocating blocking parts and path replanning capabilities.

5. Conclusions

The assembly sequence and path planning (ASPP) problem is a major problem in the assembly planning of industrial products, which comprises up to 50% of the total production time and more than 20% of the total manufacturing cost [55]. Both of its subproblems (ASP and APP) are categorized as NP-complete problems, and therefore finding global optimal solutions are not practical for most real-world problems, thus justifying the implementation of greedy and metaheuristic algorithms. Solution complications increase when the geometrical shapes of the parts can undergo deformation and nonmonotone assemblies (requiring locating the parts more than once) are considered. The above assumptions create a challenging problem that has not been tackled duly in the past as nearly all existing works in the field of ASP and APP deal with merely rigid parts.

Since most real-world assembled products like ships, aircraft, and automobiles are composed of rigid and flexible parts, automatic generation of assembly sequence and path plans for such products requires the flexibility of flexible parts to be taken into account. In this paper, we have presented a new method called SPP-Flex for solving the ASPP problem for rigid and flexible parts, which requires incorporating parameters like elasticity, force, and toleranced geometry of such parts in the model.

SPP-Flex has three main components: (1) an ASP component, which is a greedy heuristic that in each iteration tries to locally minimize the applied stress between parts being assembled along the main directions, (2) an APP component, which employs a sampling-based stochastic path planner (BXXT) to plan start-to-goal paths for all parts while avoiding workspace obstacles, and (3) an FEA component, which simulates the behavior of contacting rigid and flexible parts along a given assembly path using the Abaqus software. SPP-Flex is the first in its kind that is specifically designed for solving the ASPP problem (other few works extract a solution to the ASPP by reversing the solution to the Disassembly sequence and path planning (DASPP)), considers obstacles in the workspace, can handle products with rigid and flexible parts, allows planning translational and rotational assembly movements for parts, is probabilistically complete (will come up with a solution within sufficient time), and can handle monotone and nonmonotone ASP plans.

Within the framework of the SPP-Flex, we propose a novel concept called *Intersections Count Array, Q_p*, which despite its simple structure, makes it possible for the parts to be reassembled multiple times. An implication of this feature is that the SPP-Flex is not affected by infeasible initial assembly sequences and can ‘correct’ infeasibilities through relocating and replanning assembled parts, thus ‘undoing’ incorrect or untimely assemblies. This feature empowers the SPP-Flex to handle nonmonotone assembly planning problems, a rarely tackled case in the assembly planning literature.

To test and measure the effectiveness of the SPP-Flex, two new products, namely Compact Box and Shaft Housing assemblies, were designed and solved with this method. Both of the products are comprised of rigid and flexible parts, and the Compact Box is a monotone assembly, while the Shaft Housing is nonmonotone. To analyze the effectiveness and efficiency of the proposed ASP and APP planners, five performance criteria were defined and four different variations of the SPP-Flex—i.e., combinations of Greedy Heuristic (GH) and Breakout Local Search (BLS) sequence planners with the Bidirectional Rapidly-exploring Random Tree (BRRT) and Bidirectional Exploration-Exploitation Tree (BXXT) path planners—were coded and implemented in solving the test products for 20 times ($4 \times 2 \times 20 = 160$ simulations in total). Analysis of the computational results showed that the SPP-Flex method is most effective (yielding short assembly paths) and efficient (yielding low computational times) when its ASP and APP components are the GH and BXXT, respectively, with at most 4.6% gap in path length and 2.1% gap in runtime. Also, it was observed that optimizing the initial assembly sequence (through BLS) and feeding it to the APP component has little impact on the solution quality (path length and random tree size) but increases the computational time compared to when a simple greedy heuristic is utilized for ASP.

A future research direction in the field is the assembly sequence and path planning for *non-sequential* (requiring more than two assembling hands) and *nonlinear* (where a group of parts can be assembled as subassemblies and then added to the final assembly) products, which frequently occur in real-world applications. Other future works include simulating and exerting forces along curved (nonlinear) assembly paths in the finite element analysis component, and considering high-dimensional composite configuration spaces for more accurately defining all possible deformations of flexible objects.

Authors statement

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix –. Pseudocodes of the Algorithms

Figs. A.1, A.2, A.3 and A.4

```
SPP-Flex($Geom\_parts$, $Mech\_parts$, $Geom\_assembled$, $Obs$)
Inputs:   $Geom\_parts$ = the local coordinates of the parts
          $Mech\_parts$ = the mechanical properties (e.g., yield stress) of the parts
          $Geom\_assembled$ = the geometric coordinates of the parts at their assembled (final) positions
          $Obs$ = the workspace obstacles' geometry and location
Output:   $Plan = \langle(\pi_1, \boldsymbol{\tau}(\pi_1)), (\pi_2, \boldsymbol{\tau}(\pi_2)), \ldots\rangle$ or $\varnothing$ (if failed to find any)

1.  $AS = [(\pi_1, \delta_1), \ldots, (\pi_n, \delta_n)] \leftarrow$ GH_Sequence_Planning($Geom\_parts$, $Mech\_parts$, $Geom\_assembled$)
2.  $Plan \leftarrow \varnothing$                  // the solution to the SPP-Flex problem
3.  $p \leftarrow 1$                       // counts the number of assembled or under assembly parts
4.  While $p \le n$
5.      $replanning\_list \leftarrow \varnothing$      // initializes the set of assembled parts that need to relocate after being assembled
6.      $[q_{\text{init}}(\pi_p), q_{\text{goal}}(\pi_p)] \leftarrow$ Initial_and_Goal_Configurations($\pi_p$, $AS$, $Geom\_assembled$)
7.      $[Q_p, \boldsymbol{\tau}(\pi_p)] \leftarrow$ BXXT_Path_Planning($\pi_p$, $q_{\text{init}}(\pi_p)$, $q_{\text{goal}}(\pi_p)$, $Obs$, $Plan$)
                                   // generates the assembly path of the part $\pi_p$ (may not reach $q_{\text{goal}}(\pi_p)$)
8.      $Plan = Plan \cup \{(\pi_p, \boldsymbol{\tau}(\pi_p))\}$
9.      $k \leftarrow 1$
10.     While $k \le p - 1$ AND $endpoint(\boldsymbol{\tau}(\pi_p)) \ne q_{\text{goal}}(\pi_p)$     // if the planner fails to connect $q_{\text{init}}(\pi_p)$ to $q_{\text{goal}}(\pi_p)$
11.  $\pi_b \leftarrow \pi_{\text{argmax}(Q_p)}$            // $\pi_b$ is an assembled part that has the most interference with the part $\pi_p$ while attempting
                                   to construct its search tree (i.e., the part corresponding to the largest number in $Q_p$)
12.         $\boldsymbol{\tau}(\pi_b) \leftarrow$ Relocate($\pi_b$, $q_{goal}(\pi_b)$, $Geom\_parts$, $Obs$, $Plan$)
                                   // attempts to remove the intersection by relocating the selected blocking part
13.         If $\boldsymbol{\tau}(\pi_b) \ne \varnothing$          // relocation has been successful
14.             $replanning\_list \leftarrow replanning\_list \cup \pi_b$
15.             $Plan = Plan \cup \{(\pi_b, \boldsymbol{\tau}(\pi_b))\}$
16.             $[Q_p, \boldsymbol{\tau}(\pi_p)] \leftarrow$ BXXT_Path_Planning($\pi_p$, $endpoint(\boldsymbol{\tau}(\pi_p))$, $q_{\text{goal}}(\pi_p)$, $Obs$, $Plan$)
17.             $Plan = Plan \cup \{(\pi_p, \boldsymbol{\tau}(\pi_p))\}$
18.         End
19.         $k \leftarrow k + 1$
20.     End
21.     If $end\text{-}point(\boldsymbol{\tau}(\pi_p)) = q_{\text{goal}}(\pi_p)$
22.         For $i = 1$ to size($replanning\_list$)
23.             $\pi_d \leftarrow replanning\_list(i)$
24.             $[Q_d, \boldsymbol{\tau}(\pi_d)] \leftarrow$ BXXT_Path_Planning($\pi_d$, $endpoint(\boldsymbol{\tau}(\pi_d))$, $q_{\text{goal}}(\pi_d)$, $Obs$, $Plan$)
                                               // generates new assembly paths for relocated parts to their goals
25.             $Plan \leftarrow Plan \cup \{(\pi_d, \boldsymbol{\tau}(\pi_d))\}$
26.         End
27.         $p \leftarrow p + 1$                       // goes to the next part
28.     Else                                   // when there exists at least a blocking object that cannot be relocated
29.         Perform FEA_Test($\pi_p$, $endpoint(\boldsymbol{\tau}(\pi_p))$, $q_{\text{goal}}(\pi_p)$, $Geom\_parts$, $Mech\_parts$, $Obs$, $Plan$)
30.         If part $\pi_p$ can reach $q_{\text{goal}}(\pi_p)$ by tightly passing through blocking objects due to elastic deformation
31.             $Plan = Plan \cup \{(\pi_p,$ Interpolate($endpoint(\boldsymbol{\tau}(\pi_p))$, $q_{\text{goal}}(\pi_p)$, $step\_size$)$)\}$
32.             For $i = 1$ to size($replanning\_list$)
33.                 $\pi_d \leftarrow replanning\_list(i)$
34.                 $[Q_d, \boldsymbol{\tau}(\pi_d)] \leftarrow$ BXXT_Path_Planning($\pi_d$, $endpoint(\boldsymbol{\tau}(\pi_d))$, $q_{\text{goal}}(\pi_d)$, $Obs$, $Plan$)
                                               // replans new assembly paths for relocated parts to their goals
35.                 $Plan \leftarrow Plan \cup \{(\pi_d, \boldsymbol{\tau}(\pi_d))\}$
36.             End
37.             $p \leftarrow p + 1$                   // goes to the next part
38.         Else                               // when part $\pi_p$ cannot be assembled even after considering flexibility
39.             $Plan \leftarrow \varnothing$
40.             $p \leftarrow n + 1$                   // jumps to beyond the last part, hence terminating the algorithm
41.         End
42.     End
43. End
```

Fig. A.1. The main algorithm of the SPP-Flex.

```
GH_Sequence_Planning(Geom_parts, Mech_parts, Geom_assembled)
        Output:    $AS = [(\pi_1, \delta_1), \ldots, (\pi_n, \delta_n)]$ = the assembly sequence and direction of the parts

1.  Construct the ASM                                  // the assembly stress matrix
2.  For $i = 1$ to $n$
3.      For all $d_k \in \{-x, +x, -y, +y, -z, +z\}$    // construct the DASM matrix
4.              $ds_i^k \leftarrow \sum_{j=1}^{n} s_{j,i}^k$   // $ds_i^k$ is the Directional Stress of assembling part $p_i$ along direction
            $d_k$
5.      End
6.  End
7.  $\delta_1 \leftarrow d_k$, $k$ = Random(1, …, 6)     // Randomly select a direction $d$ among all assembly directions
8.  $\pi_1 \leftarrow \arg\min_i \{ds_i^k\}$             // $\pi_1$ is the part with the minimal $DASM(p_i, \delta_1)$
9.  $AS \leftarrow \{(\pi_1, \delta_1)\}$               // The first part and direction of the assembly sequence
10. For $h = 1$ to $n - 1$
11.         For $p_i \notin \{\pi_1, \ldots, \pi_h\}$
12.             For all $d_k \in \{-x, +x, -y, +y, -z, +z\}$
13.   $AAS(p_i, d_k) \leftarrow \sum_{\forall p_j \in \{\pi_1, \ldots, \pi_h\}} s_{i,j}^k$   // total amount of stress that all assembled parts impose to $p_i$
14.   $UAS(p_i, d_k) \leftarrow \sum_{\forall p_j \notin \{\pi_1, \ldots, \pi_h\}} s_{j,i}^k$   // total amount of stress that $p_i$ will impose to all other unassembled parts
15.             End
16.         End
17.         $(\pi_{h+1}, \delta_{h+1}) \leftarrow \arg\min_{i,k} \{AAS(p_i, d_k) + UAS(p_i, d_k)\}$   // $(\pi_{h+1}, \delta_{h+1})$ is the pair with minimal $AAS + UAS$
18.         If there are more than one part with minimal $AAS + UAS$
19.         Then select the part with smaller $AAS(p_i, d_k)$
20.         this part is still not unique If
21.         Then select a part that if assembled, it will have contact with some of the already assembled parts
22.         End
23.         End
24.         $AS \leftarrow AS \cup \{(\pi_{h+1}, \delta_{h+1})\}$
25. End
```

Fig. A.2. Pseudocode of the assembly sequence planning algorithm.

```
BXXT_Path_Planning($\pi_p$, $q_{\text{init}}(\pi_p)$, $q_{\text{goal}}(\pi_p)$, $Obs$, $Plan$)
Outputs:  $Q_p$ = the interference count of parts ($\pi_1, \pi_2, \ldots, \pi_{p-1}$) and $\pi_p$
          $\boldsymbol{\tau}(\pi_p)$ = The path of part $\pi_p$ connecting $q_{\text{init}}(\pi_p)$ to $q_{\text{goal}}(\pi_p)$ (or if not possible, to $q_{\text{intermediate}}(\pi_p)$)

1.  $\boldsymbol{\tau}(\pi_p) \leftarrow \varnothing$          // initializes the assembly path
2.  $Q_p \leftarrow [\overbrace{0 \;\; 0 \;\; \cdots \;\; 0}^{p-1} | \overbrace{0 \;\; \cdots \;\; 0}^{m}]$    // initializes the count of interferences of previously assembled parts ($\pi_1$, $\pi_2, \ldots, \pi_{p-1}$) and workspace obstacles $o_1, o_2, \ldots, o_m$ with $\pi_p$
3.  Initialize Tree $T_{active} \leftarrow \{q_{\text{init}}(\pi_p)\}$     // a tree rooted on $q_{\text{init}}(\pi_p)$
4.  Initialize Tree $T_{passive} \leftarrow \{q_{\text{goal}}(\pi_p)\}$     // a tree rooted on $q_{\text{goal}}(\pi_p)$
5.  $i \leftarrow 1$          // counts the number of attempts to establish a new branch on the search tree
6.  While $i \le iter\_max$          // Exploitation Phase
7.      $n_{\text{rand}} \leftarrow$ a random node on $T_{passive}$     // in the first iteration $n_{\text{rand}} = q_{\text{goal}}(\pi_p)$
8.      $n_{\text{near}} \leftarrow$ a node on $T_{active}$ nearest to $n_{\text{rand}}$     // in the first iteration $n_{\text{near}} = q_{\text{init}}(\pi_p)$
9.      [$n_{\text{near}}, q_1, q_2, \ldots, n_{\text{rand}}$] $\leftarrow$ Interpolate($n_{\text{near}}, n_{\text{rand}}, step\_size$)
                // finds equidistant intermediate points on the line connecting $n_{\text{near}}$ and $n_{\text{rand}}$
10.     $j \leftarrow 1$
11.     While $q_j \ne n_{\text{rand}}$     // examines if consecutive nodes on the line are collision-free
12.         $CO$ = In_Collision($\pi_p, q_j, Geom\_parts, Obs, Plan$)   // the set of objects in collision with $\pi_p$ at $q_j$
13.         If $CO = \varnothing$     // i.e., if configuration $q_j$ is collision-free
14.             $T_{active} \leftarrow T_{active} \cup \{q_j\}$     // adds the new collision-free node to the tree
15.             $T_{active} \leftarrow T_{active} \cup edge(q_{j-1}, q_j)$     // adds a new edge to the tree (assume $q_0 = n_{\text{near}}$)
16.             $j \leftarrow j + 1$
17.         Else
18.             Update $Q_p$ via incrementing by 1 any of its elements that correspond to an assembled part in $CO$
19.             Exit While
20.         End
21.     End
22.     If $q_{j-1} = n_{\text{rand}}$     // checks if the trees are connected
23.         $T_a \leftarrow T_{active} \cup T_{passive}$     // the two trees are merged and the search terminates
24.         $i \leftarrow iter\_max$
25.     Else          // Exploration Phase
26.         $q_{\text{rand}} \leftarrow$ a random free configuration for part $\pi_p$ sampled from C-space
27.         $n_{\text{close}} \leftarrow$ a node on $T_{active}$ nearest to $q_{\text{rand}}$
28.         [$n_{\text{close}}, q_1, q_2, \ldots, q_{\text{rand}}$] $\leftarrow$ Interpolate($n_{\text{close}}, q_{\text{rand}}, step\_size$)
                // finds equidistant intermediate points on the line connecting $n_{\text{close}}$ and $q_{\text{rand}}$
29.         $k \leftarrow 1$
30.     While $q_k \ne q_{\text{rand}}$     // examines if consecutive nodes on the line are collision-free
31.         $CO$ = In_Collision($\pi_p, q_k, Geom\_parts, Obs, Plan$)   // the set of objects in collision with $\pi_p$ at $q_k$
32.         If $CO = \varnothing$     // i.e., if configuration $q_k$ is collision-free
33.             $T_{active} \leftarrow T_{active} \cup \{q_k\}$     // adds the new collision-free node to the tree
34.             $T_{active} \leftarrow T_{active} \cup edge(q_{k-1}, q_k)$     // adds a new edge to the tree (assume $q_0 = n_{\text{close}}$)
35.             $k \leftarrow k + 1$
36.         Else
37.             Update $Q_p$ via incrementing by 1 any of its elements that correspond to an assembled part in $CO$
38.             Exit While
39.         End
40.     End
41.         $T_a \leftarrow$ either of $T_{active}$ or $T_{passive}$ that is rooted on $q_{\text{init}}(\pi_p)$
42.     End
43.     $i \leftarrow i + 1$
44.     Swap($T_{active}, T_{passive}$)     // so $T_{active}$ is not always the tree that contains $q_{\text{init}}(\pi_p)$
45. End
46. $q_{\text{intermediate}} \leftarrow \operatorname{argmin}(\|n, q_{\text{goal}}(\pi_p)\|)$, ($\forall$ nodes $n \in T_a$)  // finds on $T_a$ the closest node to $q_{\text{goal}}(\pi_p)$ (could be itself)
47. $\boldsymbol{\tau}(\pi_p)$ = Backtrack($T_a, q_{\text{intermediate}}, q_{\text{init}}(\pi_p)$)     // backtracking makes $q_{\text{init}}(\pi_p)$ the endpoint of the path
48. $\boldsymbol{\tau}(\pi_p)$ = Smoothen_Path(Reverse($\boldsymbol{\tau}(\pi_p)$))     // the path is reversed to make it start from $q_{\text{init}}(\pi_p)$
```

Fig. A.3. Pseudocode of the proposed BXXT algorithm for planning the assembly path of part π_p.

```
Relocate ($\pi_b$, $q_{goal}(\pi_b)$, $Geom\_parts$, $Obs$, $Plan$)
Output:   $\boldsymbol{\tau}(\pi_b)$ = The relocation path for part $\pi_b$ from $q_{goal}(\pi_b)$ to a random free configuration

1.  $\boldsymbol{\tau}(\pi_b) \leftarrow \varnothing$
2.  Initialize $T_{relocate} \leftarrow \varnothing$          // a tree rooted on $q_{goal}(\pi_b)$
3.  For $i = 1$ to $max\_removal\_attempts$
4.      $q_{rand} \leftarrow$ a random free configuration for part $\pi_b$ sampled in C-space
5.      $[q_{goal}(\pi_b), q_1, q_2, \ldots, q_{rand}] \leftarrow$ Interpolate($q_{goal}(\pi_b)$, $q_{rand}$, $step\_size$)
                                // finds equidistant intermediate points on the line connecting $q_{goal}(\pi_b)$ to $q_{rand}$
6.      $k \leftarrow 1$
7.      While $q_k \neq q_{rand}$          // examines if consecutive nodes on the line are collision-free
8.          $CO$ = In_Collision($\pi_b$, $q_k$, $Geom\_parts$, $Obs$, $Plan$)   // the set of objects in collision with $\pi_b$ at $q_k$
9.          If $CO = \varnothing$                          // i.e., if configuration $q_k$ is collision-free
10.             $T_{relocate} \leftarrow T_{relocate} \cup \{q_k\}$           // adds a new node to the tree
11.             $T_{relocate} \leftarrow T_{relocate} \cup edge(q_{k-1}, q_k)$   // adds a new edge to the tree (assume $q_0 = n_{near}$)
12.             $k \leftarrow k + 1$
13.         Else
14.             exit While
15.         End
16.     End
17. End
18. $q_{far} \leftarrow \arg\max(\|q, q_{goal}(\pi_b)\|)$ (for all $q \in T_{relocate}$)   // finds on $T_{relocate}$ the node with largest distance to $q_{goal}(\pi_b)$
19. $\boldsymbol{\tau}(\pi_b) \leftarrow$ the path connecting $q_{goal}(\pi_b)$ to $q_{far}$
```

Fig. A.4. Procedure for relocating the blocking part to an intermediate configuration.

References

[1] M.F.F. ab Rashid, A hybrid Ant-Wolf Algorithm to optimize assembly sequence planning problem, Assem. Autom. 37 (2017) 238–248.

[2] J. Acker, D. Henrich, Manipulation of deformable linear objects: from geometric model towards program generation, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2005, pp. 1541–1547.

[3] I. Aguinaga, D. Borro, L. Matey, Path-planning techniques for the simulation of disassembly tasks, Assem. Autom. 27 (2007) 207–214.

[4] I. Aguinaga, D. Borro, L. Matey, Parallel RRT-based path planning for selective disassembly planning, Int. J. Adv. Manuf. Technol. 36 (2008) 1221–1233.

[5] H. Akbaripour, E. Masehian, Semi-lazy probabilistic roadmap: a parameter-tuned, resilient and robust path planning method for manipulator robots, Int. J. Adv. Manuf. Technol. 89 (2017) 1401–1430.

[6] M.R. Bahubalendruni, B. Deepak, B.B. Biswal, An advanced immune based strategy to obtain an optimal feasible assembly sequence, Assem. Autom. 36 (2016) 127–137.

[7] R. Bohlin, L.E. Kavraki, Path planning using lazy PRM, in: Proceedings of the ICRA'00. IEEE International Conference on Robotics and Automation, 2000, IEEE, 2000, pp. 521–528.

[8] F. Bonneville, C. Perrard, J.-.M. Henrioud, A genetic algorithm to generate and evaluate assembly plans, in: Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, ETFA, 1995, pp. 231–239.

[9] P.-.B. Cao, R.-.B. Xiao, Assembly planning using a novel immune approach, Int. J. Adv. Manuf. Technol. 31 (2007) 770–782.

[10] C.P. Chen, Design of a real-time AND/OR assembly scheduler on an optimization neural network, J. Intell. Manuf. J. Intell. Manuf. 3 (1992) 251–261.

[11] W.-.C. Chen, P.-.H. Tai, W.-.J. Deng, L.-.F. Hsieh, A three-stage integrated approach for assembly sequence planning using neural networks, Expert Syst. Appl. 34 (2008) 1777–1786.

[12] H.M. Choset, S. Hutchinson, K.M. Lynch, G. Kantor, W. Burgard, L.E. Kavraki, S. Thrun, Principles of Robot motion: theory, algorithms, and Implementation, MIT press, 2005.

[13] J. Cortés, L. Jaillet, T. Siméon, Molecular disassembly with Rrt-Like algorithms, ICRA (2007) 3301–3306.

[14] J. Cortés, L. Jaillet, T. Siméon, Disassembly path planning for complex articulated objects, IEEE Trans. Rob. 24 (2008) 475–481.

[15] L. Da Xu, C. Wang, Z. Bi, J. Yu, AutoAssem: an automated assembly planning system for complex products, IEEE Trans. Ind. Inf. 8 (2012) 669–678.

[16] B. Deepak, G. Bala Murali, M.R. Bahubalendruni, B. Biswal, Assembly sequence planning using soft computing methods: a review, Proc. Inst. Mech. Eng. Part E J. Process Mech. Eng. 233 (3) (2018) 653–683., https://doi.org/10.1177/0954408918764459, 0954408918764459.

[17] E. Fogel, D. Halperin, Polyhedral assembly partitioning with infinite translations or the importance of being exact. Algorithmic Foundation of Robotics VIII, Springer, 2009.

[18] L. Gao, W. Qian, X. Li, J. Wang, Application of memetic algorithm in assembly sequence planning, Int. J. Adv. Manuf. Technol. 49 (2010) 1175–1184.

[19] S. Ghandi, E. Masehian, Assembly sequence planning of rigid and flexible parts, J. Manuf. Syst. 36 (2015) 128–146.

[20] S. Ghandi, E. Masehian, A breakout local search (BLS) method for solving the assembly sequence planning problem, Eng. Appl. Artif. Intell. 39 (2015) 245–266.

[21] S. Ghandi, E. Masehian, Review and taxonomies of assembly and disassembly path planning problems and approches, Comput. Aided Des. (2015).

[22] Gibson, S.F. & Mirtich, B. 1997. A Survey of Deformable Modeling in Computer Graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratory, Cambridge, MA.

[23] M. Givehchi, A.H. Ng, L. Wang, Spot-welding sequence planning and optimization using a hybrid rule-based approach and genetic algorithm, Robot. Comput. Integr. Manuf. 27 (2011) 714–722.

[24] J. Guo, P. Wang, N. Cui, Adaptive ant colony algorithm for on-orbit assembly planning, in: Proceedings of the Second IEEE Conference on Industrial Electronics and Applications, ICIEA, 2007, pp. 1590–1593.

[25] D. Halperin, J.-.C. Latombe, R.H. Wilson, A general framework for assembly planning: the motion space approach, Algorithmica 26 (2000) 577–601.

[26] T. Hermansson, R. Bohlin, J.S. Carlson, R. Söderberg, Automatic assembly path planning for wiring harness installations, J. Manuf. Syst. 32 (3) (2013) 417–422, https://doi.org/10.1016/j.jmsy.2013.04.006.

[27] D. Hong, H. Cho, A neural-network-based computational scheme for generating optimized robotic assembly sequences, Eng. Appl. Artif. Intell. 8 (1995) 129–145.

[28] D. Hong, H. Cho, Generation of robotic assembly sequences using a simulated annealing, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 1999, pp. 1247–1252.

[29] C. Hui, L. Yuan, Z. Kai-Fu, Efficient method of assembly sequence planning based on GAAA and optimizing by assembly path feedback for complex product, Int. J. Adv. Manuf. Technol. 42 (2009) 1187–1204.

[30] C. Hui, W. Shengmin, Q. Xi, Space swept algorithm based assembly path planning method for aircraft, J. Beijing Univ. Aeronaut. Astronaut. 6 (2010) 012.

[31] X. Jiang, Y. Nagaoka, K. Ishii, S. Abiko, T. Tsujita, M. Uchiyama, Robotized recognition of a wire harness utilizing tracing operation, Robot. Comput. Integr. Manuf. 34 (2015) 52–61.

[32] P. Jiménez, Survey on model-based manipulation planning of deformable objects, Robot. Comput. Integr. Manuf. 28 (2012) 154–163.

[33] P. Jiménez, Survey on assembly sequencing: a combinatorial and geometrical perspective, J. Intell. Manuf. J. Intell. Manuf. 24 (2013) 235–250.

[34] X. Jin, T. Zhang, H. Yang, An analysis of the assembly path planning of decelerator based on virtual technology, Phys. Procedia 25 (2012) 170–175.

[35] M. Jordan, A. Perez. Optimal Bidirectional Rapidly-Exploring Random Trees. Technical Report MIT-CSAIL-TR-2013-021, Computer Science and Artificial

Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 2013.
[36] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Robot. Res. 30 (2011) 846–894.
[37] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli, S. Teller, Anytime Motion Planning Using the RRT, Institute of Electrical and Electronics Engineers, 2011.
[38] L. Kavraki, J.-.C. Latombe, R.H. Wilson, On the complexity of assembly partitioning, Inf. Process Lett. 48 (1993) 229–235.
[39] L. Kavraki, P. Svestka, M.H. Overmars, Probabilistic Roadmaps For Path Planning in High-Dimensional Configuration Spaces, Unknown Publisher, 1994.
[40] J.J. Kuffner, S.M. Lavalle, RRT-connect: an efficient approach to single-query path planning, in: Proceedings of the IEEE International Conference on Robotics and Automation, 2000 ICRA'00, IEEE, 2000, pp. 995–1001.
[41] Lavalle, S.M. 1998. Rapidly-exploring Random trees: A new Tool For Path Planning.
[42] D.T. Le, J. Cortés, T. Siméon, A path planning approach to (dis) assembly sequencing, in: Proceedings of the IEEE International Conference on Automation Science and Engineering, IEEE, 2009, pp. 286–291.
[43] S. Lee, H. Moradi, Disassembly sequencing and assembly sequence verification using force flow networks, in: Proceedings of the IEEE International Conference on Robotics and Automation, 1999, IEEE, 1999, pp. 2762–2767.
[44] T. Lozano-Perez, R.H. Wilson, Assembly sequencing for arbitrary motions, in: Proceedings of the IEEE International Conference on Robotics and Automation 1993, IEEE, 1993, pp. 527–532.
[45] T. Lu, B. Zhang, P. Jia, Assembly sequence planning based on graph reduction, in: Proceedings of the 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering, TENCON'93, IEEE, 1993, pp. 119–122.
[46] Q. Luo, J. Xiao, Haptic rendering involving an elastic tube for assembly simulations, in: Proceedings of the 6th IEEE International Symposium on Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005. (ISATP 2005), IEEE, 2005, pp. 53–59.
[47] H. Lv, C. Lu, An assembly sequence planning approach with a discrete particle swarm optimization algorithm, Int. J. Adv. Manuf. Technol. 50 (2010) 761–770.
[48] N. Lv, J. Liu, X. Ding, H. Lin, Assembly simulation of multi-branch cables, J. Manuf. Syst. 45 (2017) 201–211.
[49] E. Masehian, S. Ghandi, ASPPR: a new Assembly Sequence and Path Planner/ Replanner for monotone and nonmonotone assembly planning, Comput. Aided Des. 123 (2020), 102828.
[50] H. Mosemann, T. Bierwirth, F. Wahl, S. Stoeter, Generating polyhedral convex cones from contact graphs for the identification of assembly process states, in: Proceedings IEEE International Conference on Robotics and Automation, 2000 ICRA'00., IEEE, 2000, pp. 744–749.
[51] S. Motavalli, A.-.U. Islam, Multi-criteria assembly sequencing, Comput. Ind. Eng. 32 (1997) 743–751.
[52] C.L. Nielsen, L.E. Kavraki, A two level fuzzy PRM for manipulation planning, in: Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000.(IROS 2000), IEEE, 2000, pp. 1716–1721.
[53] J.H. Oliver, H.-.T. Huang, Automated path planning for integrated assembly design, Comput. Aided Des. 26 (1994) 658–666.
[54] L.-.M. Ou, X. Xu, Relationship matrix based automatic assembly sequence generation from a CAD model, Comput. Aided Des. 45 (2013) 1053–1067.
[55] C. Pan, Integrating CAD Files and Automatic Assembly Sequence Planning, Iowa State University, 2005. PHD thesis.
[56] A.K. Priyadarshi, S.K. Gupta, Algorithms for generating multi-stage molding plans for articulated assemblies, Robot. Comput. Integr. Manuf. 25 (2009) 91–106.
[57] S. Rakshit, S. Akella, The Influence of Motion Path and Assembly Sequence On the Stability of Assemblies, Robotics: Science and Systems IX, 2013.
[58] M.F.F. Rashid, W. Hutabarat, A Tiwari, A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches, Int. J. Adv. Manuf. Technol. 59 (2012) 335–349.
[59] A. Remde, D. Henrich, H. Wom, Manipulating deformable linear objects-contact state transitions and transition conditions, in: Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999. IROS'99., IEEE, 1999, pp. 1450–1455.
[60] B. Romney, Atlas: an automatic assembly sequencing and fixturing system. Geometric Modeling: Theory and Practice, Springer, 1997.
[61] B. Romney, C. Godard, M. Goldwasser, G. Ramkumar, An efficient system for geometric assembly sequence generation and evaluation, Comput. Eng. (1995) 699–712.
[62] A. Shah, L. Blumberg, J. Shah, Planning for manipulation of interlinked deformable linear objects with applications to aircraft assembly, IEEE Trans. Autom. Sci. Eng. (2018) 1–16.
[63] A.J. Shah, J.A. Shah, Towards manipulation planning for multiple interlinked deformable linear objects, in: Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA),, IEEE, 2016, pp. 3908–3915.
[64] C. Sinanoglu, H.R. Börklü, An assembly sequence-planning system for mechanical parts using neural network, Assem. Autom. 25 (2005) 38–52.
[65] J. Song, Q. Chen, Z. Li, A peg-in-hole robot assembly system based on Gauss mixture model, Robot. Comput. Integr. Manuf. 67 (2021), 101996.
[66] Q. Su, A hierarchical approach on assembly sequence planning and optimal sequences analyzing, Robot. Comput. Integr. Manuf. 25 (2009) 224–234.
[67] S. Sundaram, I. Remmler, N.M. Amato, Disassembly sequencing using a motion planning approach, in: Proceedings of the 2001IEEE International Conference on Robotics and Automation, 2001, ICRA, IEEE, 2001, pp. 1475–1480.
[68] U. Thomas, M. Barrenscheen, F.M. Wahl, Efficient assembly sequence planning using stereographical projections of c-space obstacles, in: Proceedings of the IEEE International Symposium on Assembly and Task Planning, 2003., IEEE, 2003, pp. 96–102.
[69] M. Tiwari, A. Kumar, A. Mileham, Determination of an optimal assembly sequence using the psychoclonal algorithm, Proc. Inst. Mech. Eng. Part B J. Eng. Manuf. 219 (2005) 137–149.
[70] Y.-.J. Tseng, F.-.Y. Yu, F.-.Y. Huang, A green assembly sequence planning model with a closed-loop assembly and disassembly sequence planning using a particle swarm optimization method, Int. J. Adv. Manuf. Technol. 57 (2011) 1183–1197.
[71] V. Venkatesan, J. Seymour, D.J. Cappelleri, Microassembly sequence and path planning using sub-assemblies, J. Mech. Robot. 10 (6) (2018) 1–14, https://doi.org/10.1115/1.4041333.
[72] W. Wan, K. Harada, K. Nagata, Assembly sequence planning for motion planning, Assem. Autom. 38 (2018) 195–206.
[73] J. Wang, J. Liu, Y. Zhong, A novel ant colony algorithm for assembly sequence planning, Int. J. Adv. Manuf. Technol. 25 (2005) 1137–1143.
[74] L. Wang, Y. Hou, X. Li, S. Sun, An enhanced harmony search algorithm for assembly sequence planning, Int. J. Model. Ident. Control 18 (2013) 18–25.
[75] Y. Wang, J. Liu, Chaotic particle swarm optimization for assembly sequence planning, Robot. Comput. Integr. Manuf. 26 (2010) 212–222.
[76] S.A. Wilmarth, N.M. Amato, P.F. Stiller, MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space, in: Proceedings of the 1999 IEEE International Conference on Robotics and Automation, 1999, IEEE, 1999, pp. 1024–1031.
[77] R.H. Wilson, On Geometric Assembly Planning, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1992.
[78] R.H. Wilson, L. Kavraki, J.-.C. Latombe, T. Lozano-Pérez, Two-handed assembly sequencing, Int. J. Robot. Res. 14 (1995) 335–350.
[79] J. Wolter, E. Kroll, Toward assembly sequence planning with flexible parts, in: Proceedings of the 1999 IEEE International Conference on Robotics and Automation,, IEEE, 1996, pp. 1517–1524.
[80] S. Yi, Y. Jianfeng, L. Yuan, Y. Haicheng, An assembly-path-planning algorithm for improving aircraft assembly, J.-Northwest. Polytech. Univ. 19 (2001) 121–124.
[81] Z.B.L.M.Z. Yi, M Jianhua, Research on assembly sequence planning based on firefly algorithm, J. Mech. Eng. 11 (2013) 25.
[82] J. Yoon, Assembly simulations in virtual environments with optimized haptic path and sequence, Robot. Comput. Integr. Manuf. 27 (2011) 306–317.
[83] H. Zhang, H. Liu, L. Li, Research on a kind of assembly sequence planning based on immune algorithm and particle swarm optimization algorithm, Int. J. Adv. Manuf. Technol. (2013) 1–14.
[84] W. Zhou, J. Yan, Y. Li, C. Xia, J. Zheng, Imperialist competitive algorithm for assembly sequence planning, Int. J. Adv. Manuf. Technol. 67 (2013) 2207–2216.
[85] W. Zhou, J.-.R. Zheng, J.-.J. Yan, J.-.F. Wang, A novel hybrid algorithm for assembly sequence planning combining bacterial chemotaxis with genetic algorithm, Int. J. Adv. Manuf. Technol. 52 (2011) 715–724.