Hindawi
Journal of Robotics
Volume 2023, Article ID 3264369, 20 pages
https://doi.org/10.1155/2023/3264369

Research Article

Assembly Sequence Planning for Rectangular Modular Robots with Accessibility Constraints

Anelize Zomkowski Salvi[1] **and Henrique Simas**[2]

[1]*Department of Control, Automation and Computation, Federal University of Santa Catarina, Blumenau 89036-002, Brazil*
[2]*Department of Mechanical Engineer, Federal University of Santa Catarina, Florianópolis 88040-900, Brazil*

Correspondence should be addressed to Anelize Zomkowski Salvi; anelize.salvi@udesc.br

Received 3 November 2022; Revised 28 August 2023; Accepted 8 September 2023; Published 25 November 2023

Academic Editor: Bingxiao Ding

Product assembly is the final step in a manufacturing process where the individual components of a product are joined together. Assembly sequence planning (ASP) can be defined as the problem of finding a collision-free sequence of operations that allow the product assembly. Considering that during assembly individual modules cannot pass through a gap only as large as a module side, the ASP problem can be extended to modular robots, more specifically to rectangular modular robots. The main ASPs presented in the literature that are applicable to rectangular modules do not allow configurations with narrow corridors, i.e., corridors which are too narrow for a robot to transverse. Furthermore, these ASPs do not allow preassembled substructures or the free selection of the assembly starting point. Thus, the main goal of this work is to extend the classes of rectangular modular robot configurations that can be assembled without violating the accessibility condition. This paper introduces three novel ASP for constructing planar target structures composed of rectangular modular robots. Each ASP is adequate for a different scenario. Original implementation results and mathematical proofs for the three novel ASPs are also presented. To the best of the authors' knowledge, this is the first work that presents, considering the accessibility condition, how to obtain centralized ASPs for assembling planar structures composed of rectangular modules with narrow corridors. Furthermore, the novel ASPs allow structures composed of subsets of preassembled modules and configurations with internal holes. They also allow the choice of the assembly starting point. Moreover, the third ASP proposed in this work allows achieving discontinuous assembly paths, i.e., wherever possible, the ASP allows a novel module to not connect to the latest added robot.

1. Introduction

Product assembly is the final step in the manufacturing process, where the individual components of a product are joined together. Assembly sequence planning (ASP) can be defined as the problem of finding a collision-free sequence of operations that allow the product assembly. According to Su [1], the purpose of the ASP is to determine a proper sequence of components and assembly operations. With the ordered sequence, the components can be located at the specified positions and fixed within the assembly operations to construct the final product. An inclusive review on assembly sequence generation and its automation can be found in a study by Bahubalendruni and Biswal [2].

ASP is one of the best known production scheduling problems, which has been proved to be a NP-hard problem, and ASP is a typical combinatorial explosion problem with the increase in the number of components in products [3]. A NP-hard problem is at least as hard as the hardest problems in NP. A decision problem H is NP-hard when for every problem L in NP, there is a polynomial-time many-one reduction from L to H. Concerning the assembly of mechanical products, according to Ghandi and Masehian [4], assembly generation methods can be divided into four general categories:

(1) *Human interaction methods*: This approach relies on user responses to a set of questions [5, 6].

(2) *Geometric feasible reasoning approaches*: These methods plan feasible assembly paths [7–12]. Recent methods also include an analysis about the ASP geometric feasibility along principal axes [13–15].

(3) *Knowledge-based reasoning approaches*: This approach considers geometrical and non-geometrical information, as human reasoning about the assembly process [16–18].

(4) *Intelligent methods*: These methods use multiple techniques as neural networks [19], heuristics [20–22], and optimization [23], among others.

It is important to notice that, in order to perform parallel assemblies, the problem of achieving mechanical stability during the assembly steps and the problem of identifying subassembly detection must be addressed. In this context, different approaches, including knowledge-based reasoning approach and intelligent methods, can be found in a study by Gulivindala et al. [24], Gulivindala et al. [25], Bahubalendruni et al. [26], Bahubalendruni and Biswal [27], and Bahubalendruni et al. [28].

The problem of determining an adequate assembly sequence can be extended to scenarios other than mechanical assemblies, for example, it can be extended to modular robots. Modular robots are robots capable of changing their morphology in order to perform various tasks and in order to adapt to different environments. For example, they can be employed to construct large marine structures, out in space, or even to construct furniture. For these robots, the problem of finding a sequence of collision-free operations necessary for bringing a configuration together can be regarded as the problem of finding an assembly sequence that ensures easy accessibility for individual modules.

Paulos et al. [29] and Seo et al. [30, 31] first addressed the ASP problem for modular robots. More specifically, they were the first authors to address the problem of how to determine an ASP for structures composed of rectangular modular robots. An important constraint must be considered during assembly: individual modules cannot pass through a gap only as large as the module side.

Paulos et al. [29] and Seo et al. [30, 31] considered the problem of constructing large scale floating structures in order to accelerate humanitarian missions or disaster relief. This can be achieved by assembling together many self-propelled shipping containers whose dimensions are specified by the International Standards Organization (ISO).

These ISO shipping containers are regarded as robotic boats or modules.

By considering such constraints for rectangular modules, Paulos et al. [29] and Seo et al. [30, 31] introduced the assembly problem for rectangular modules, which is addressed in this paper. However, their ASPs can only be applied to a restricted subclass of structures. More precisely, their ASPs do not allow configurations with narrow corridors, i.e., corridors that are too narrow for a robot to transverse. Furthermore, their ASPs do not allow preassembled substructures or the free selection of the assembly starting point.

Thus, the main goal of this work is to extend the classes of rectangular modular robot configurations, which can be assembled without violating the accessibility condition when compared to the main centralized ASPs in literature for modular robots [29–31]. Furthermore, as the classical ASPs for assembling mechanical parts were not applied for modular robots, this work can be only directly compared with the works of Paulos et al. [29] and Seo et al. [30, 31].

In this context, this work presents three novel ASPs for rectangular modular robots considering the same accessibility condition, as in a study by Paulos et al. [29] and Seo et al. [30, 31]. These novel ASPs can be successfully applied for different situations and can assembly novel classes of configurations, not solved in literature [32]. The main limitation is the overall complexity, which is discussed in detail in Section: Mathematical proofs for the proposed ASPs.

To the best of the author's knowledge, this is the first work that presents, considering the accessibility condition, how to obtain a centralized ASP for assembling planar structures composed of rectangular modules with the following characteristics:

(1) With narrow corridors, i.e., corridors that are too narrow for a robot to transverse.

(2) Composed of subsets of preassembled modules and configurations with internal holes.

(3) Choosing the ASP starting point.

(4) Achieving discontinuous assembly paths, i.e., wherever possible, the ASP allows a novel module not to be connected to the latest added robot.

It is important to notice that the accessibility condition imposed by Paulos et al. [29] and Seo et al. [30, 31] is generally employed for modular robots. In a study by Naz et al. [33], for example, this condition is considered to simplify the self-reconfiguration of cylindrical lattice-based modular robots. More specially, the method in a study by Naz et al. [33] maintains an empty cell between cylindrical modules during reconfiguration. Furthermore, in a study by Salvi et al. [34], the authors address the problem of assembling target structures composed of subsets of modules forming independent and free-flying robots. These configurations are considered without internal holes and with the same constraints, as in a study by Paulos et al. [29] and Seo et al. [30, 31].

There is an extensive literature on the self-assembly of modular robots with techniques in different directions [35–37]. However, in this correlated problem, the robots are not usually free flying. In this context, in a study by Tucci et al. [38], a distributed self-assembly algorithm for avoiding unfeasible docking positions, similar to the constraints imposed by Paulos et al. [29] and Seo et al. [30, 31], is presented. This algorithm for self-reconfiguration of modular robots has similar complexity to the herein presented centralized algorithm for free-flying rectangular modules.

Other important correlated problems are the autonomous robotic assembly of target modular structures [39–43] and the path planning problem. According to Orozco-Rosas et al. [44], path planning is the problem of the autonomous

navigating a mobile robot, where the objective is to conduct an autonomous mobile robot from one (start) position to the goal (final) position in an optimal and safe manner.

It is important to notice that this paper does not focus on the path planning problem. In fact, it only addresses the problem of determining an adequate assembly sequence for modular robot assembly, i.e., when each module or structure can be moved to its desired position in order to avoid conflicts during the assembly process. Thus, it is considered that during assembly, a path planning strategy should be applied in order to move each module or structure to its desired position.

According to Orozco-Rosas et al. [45], the path planning problem is a fundamental issue in mobile robot navigation because of the need of having algorithms to convert high-level specifications of tasks from humans into low-level descriptions of how to move. Inclusive surveys on path planning can be found in a study by Abujabal et al. [46], Lin et al. [47], Sánchez-Ibáñez et al. [48], and Patle et al. [49]. As example of possible path planning adequate for this problem, in a study by Kelly [50] and Gheneti et al. [51], a trajectory planning for shape shifting between configurations composed of rectangular robotic boats with the same accessibility conditions addressed in this paper is presented.

Furthermore, the results presented in this study can be extended to address the assembly process of modular micropositioning stages (MPS). This class of problems involves creating products capable of fulfilling various functions through the combination of distinct blocks. The ongoing trend of product miniaturization in diverse industrial domains has given rise to an increasing demand for MPS capable of executing micromanipulation and microassembly tasks with submicrometer or nanometer precision. These MPS play a pivotal role in achieving ultra-high precision while miniaturizing products in accordance with the emerging industrial trend [52].

In order to ensure broader adoption of flexure-based MPSs, two critical challenges must be addressed: achieving superior performance and maintaining cost-effectiveness. Presently, the design and fabrication processes for MPSs are both time-consuming and costly. Custom configurations are often tailored to specific applications, resulting in limited flexibility for broader usage. Consequently, the development of more adaptable and cost-efficient MPSs has become a pressing concern in the realm of micromanipulation and microassembly technology. For researchers interested in exploring relevant designs and gaining a comprehensive understanding of this topic, a comprehensive bibliographic review can be found in a study by Wu and Xu [53], Ding et al. [54], and Liao et al. [55].

In the next section, the materials and methods and basic definitions necessary for introducing the ASPs proposed in this paper are presented.

2. Materials and Methods

This paper introduces three novel ASP for constructing planar target structures composed of rectangular modular robots. Each ASP, as will be further explained in this section,

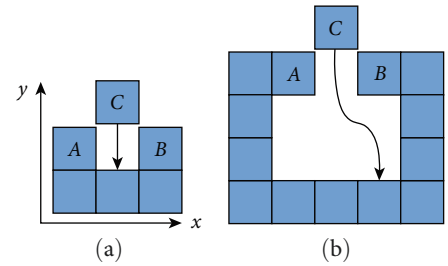

Figure 1: The gap between A and B blocks module C, thus the configurations present narrow corridors.

is adequate for a different scenario as, for example, for assembling configurations with or without internal holes composed of robot modules or of subsets of modules. However, before introducing the main characteristics of the proposed ASPs, some basic definitions must be discussed.

During assembly, the following condition about accessibility must be verified for all the ASPs proposed in this paper: *any rectangular module cannot pass through a gap only as large as a side of a module between two physical robots already assembled in the structure* [29]. Figure 1 shows two scenarios where the accessibility condition is not verified. This accessibility condition, first stated in a study by Paulos et al. [29], is incorporated into the three novel ASPs because it models common mechanical constraints found in modular robots. Other constraints can be easily incorporated into all the ASPs. Also, the ASPs herein proposed extend the classes of structures that can be assembled without accessibility issues when compared to those proposed in literature. Thus, in order to compare the ASPs proposed in this work with the methods found in literature, the same constraints introduced in a study by Paulos et al. [29] are herein employed.

Given a configuration, if the accessibility condition is not satisfied for any of its module, then the configuration is considered to have a narrow corridor [29]. In these configurations, a site(s) to be occupied by a module can only be reached by passing through a gap at most as large as the edge of a robot, as shown in Figure 1. It is important to notice that the ASPs herein proposed can assembly different classes of configurations with narrow corridors where the main algorithms in literature cannot be applied [29].

A configuration is considered composed of shape heterogeneous modular robots or robot systems for simplicity, if it is composed of subsets of modules forming independent and free-flying robots. Each of these robots is a collection of rectangular modules, where each robot module is represented by the coordinates of its centroid. This situation is shown in Figure 2(a) where a configuration composed of 12 different subrobots labeled from A to L is presented, along with the $x-y$ coordinate system [34].

An ASP is considered path continuous if the configuration is constructed in a continuous path, i.e., in each iteration, wherever it is possible, the novel robot connects to the latest added robot. This implies that at each iteration, if the previously added robot has a neighbor that can be added to the structure, the ASP must add this neighbor module to the

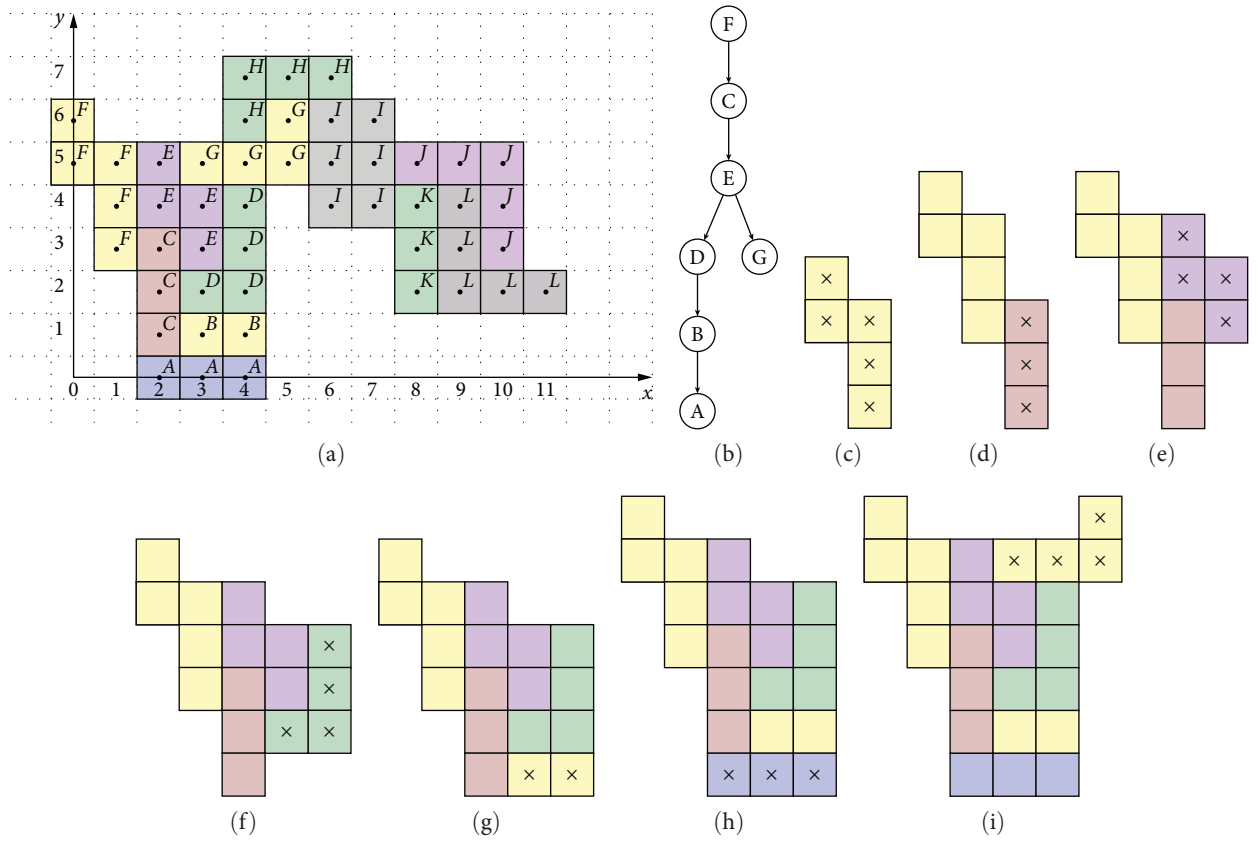

FIGURE 2: (c–i) Path continuous ASP for the configuration, as shown in (a). (b) The ASP tree.

structure. Otherwise, the ASP must search for a robot already in the structure that has neighbors to add. In this context, the algorithm is similar to a depth-first search in a graph where the search starts at a selected root node and explores as far as possible along each branch before backtracking [56].

The path continuous assembly is shown in Figure 2(c)–2(i), where the configuration, as shown in Figure 2(a), is assembled. The assembly sequence tree is shown in Figure 2(b). It can be noticed that the novel robot system is preferably added by the previously added one. In the step, as shown in Figure 2(i), this fact does not happen. In fact, in this step, the novel robot system G is not added by the previously added one, robot system A. The reason is that the robot system A has no novel neighbors to add to the structure. Thus, as shown in Figure 2(i), the ASP searches for an already assembled robot, which has neighbors that can be added to the robot system. Then, the robot system D finally adds robot system G to the structure. It is important to notice, however, that wherever a novel module or robot system is assembled by the herein proposed ASPs, the accessibility condition is always guaranteed. Thus, the process can be regarded as a conventional depth-first search with the introduction of the accessibility condition.

The first two ASPs differ because only ASP2 can be applied for configurations with internal holes, as shown in Figure 3. In this figure, the path continuous construction is obtained by allowing a novel robot system to be assembled by the previous robot systems or by the previously added holes. In fact, the holes are also herein modeled as robot systems.

The third ASP, ASP3, however, uses holes as a way to traverse the configuration in a noncontinuously way, i.e., shifting back and forth from the boundary of the holes. This situation is shown in Figure 4 where the arrows represent the assembly order and h_1, h_2, and h_3 represent the figure's internal holes.

Table 1 summarizes the use cases of the three novel ASPs proposed in this text.

In the ASPs proposed in this text, the problem of determining an assembly order can be regarded as the problem of choosing a path to transverse the whole structure adding as many robots as possible without violating the accessibility condition.

In this paper, the target structure is modeled as an undirected graph: each module is represented by a vertex and neighbors relations are represented by edges. Furthermore, powerful tools of graph theory [56] are applied in order to find an assembly sequence that does not violate the accessibility condition, i.e., a feasible path to transverse the graph.

Next sections introduce the novel ASPs, which constitute a complete and original method to assemble target structures with or without internal holes, composed of modular robots or of shape heterogeneous modular robots with the possibility of allowing discontinuous assembly paths. Furthermore, in the proposed ASPs, the assembly starting point can be freely chosen.

2.1. ASP1: Novel ASP for Path Continuous Construction of Configurations without Internal Holes. In the three novel ASPs, G is the class that contains information about the target

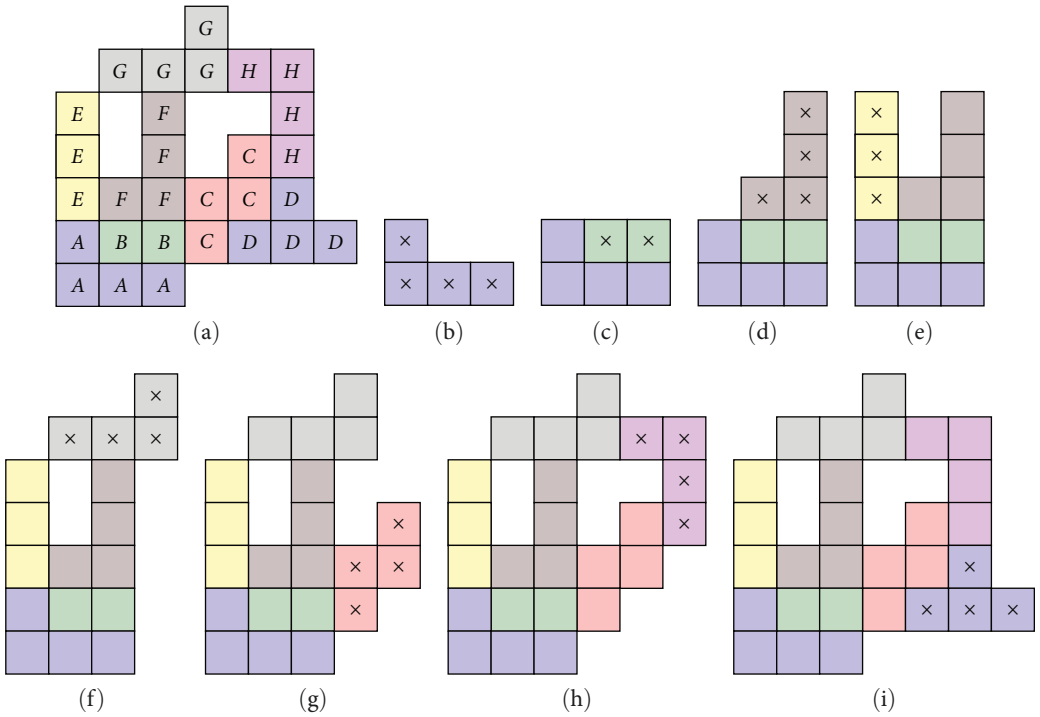

Figure 3: (b–i) Path continuous ASP for the configuration, as shown in (a), with robot system A as starting point.

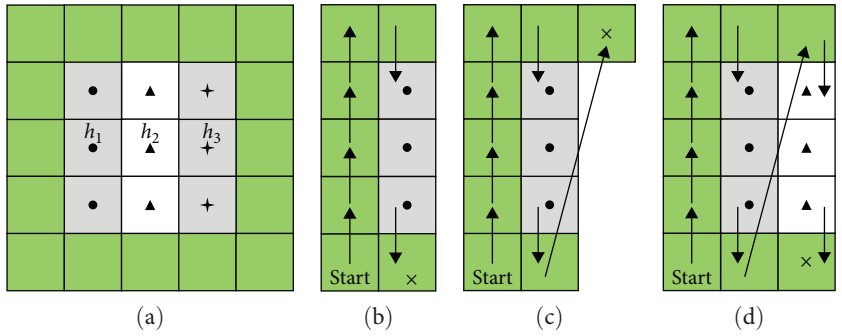

Figure 4: (b–d) An example of ASP3 for modular construction allowing discontinuous assembly paths for the configuration in (a).

Table 1: Novel ASPs use cases.

	Modular rectangular robots	Robot systems	Path continuous assembly	Internal holes	Path discontinuous assembly
ASP1	✓	✓	✓	—	—
ASP2	✓	✓	✓	✓	—
ASP3	✓	✓	✓	✓	✓

structure and about the ASP. The target structures can be represented by an undirected graph in the adjacency list form: a vertex represents a module or a robot system and an edge represents a neighbor relation between modules or between robot systems. In the class G, as introduced by Salvi et al. [34]:

(1) $G.adj[s]$ is the *adjacency list* of element s; in other words, the list of robots that are neighbors of s in the target structure.

(2) $G.Tree[s]$ is the list of elements added to an element s during the assembly process.

(3) $G.parent(s)$ is the *parent* of element s in the assembly process.

2.1.1. Algorithm 1: Main Procedure for ASP1. In order to introduce the novel ASP for path continuous construction, the case where configuration's have no internal holes is

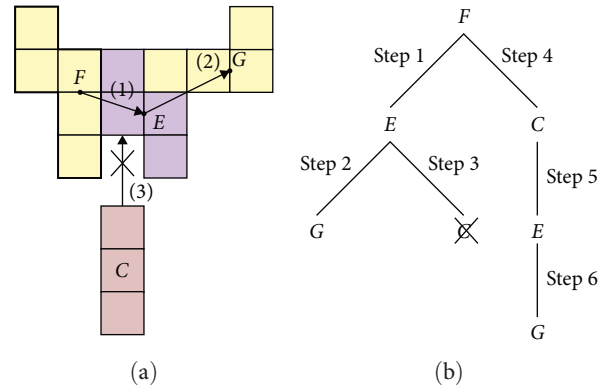

Figure 5: (a) If the algorithm places $F \rightarrow E \rightarrow G$ and then E placing robot C, an incorrect assembly sequence is encountered. Thus, the parent of C, robot E, must be deleted from the structure along with G, the unique robot belonging to E's assembly tree branch (represented in (b)).

first considered. This ASP was first presented by Salvi et al. [34].

This first ASP for path continuous construction can be applied for both configurations composed of rectangular modules by robot systems. Also, the ASP starting point can be freely chosen. Furthermore, the ASP for configurations with internal holes is an extension of the case herein discussed.

First, let us consider robots F, E, C, and G, as shown in Figure 5(a). The main idea is to transverse the structure in its depth, as in a depth-first search. Thus, each robot can add one of its neighbors if the accessibility condition is satisfied for this neighbor robot. If an accessibility issue is encountered while adding a novel robot to the structure, the procedure disassembles some of the structure's parts until the accessibility issue is solved.

Two cases with accessibility issues can arise during the assembly process of a robot w:

(1) The accessibility issue is caused by w's parent in the process.
(2) The accessibility for w is caused by another robot already added to the ASP.

For the first case, where the accessibility issue for w is caused by w's parent in the process, consider the configuration, as shown in Figure 5(a). As shown in Figure 5(b), in steps 1–2, the ASP visits $F \rightarrow E \rightarrow G$. At the end of step 2, as G has no neighbor to add, the procedure tries adding a new neighbor of G's parent, robot E. At this point, the only undiscovered neighbor of E is robot C. However, C cannot be added to the structure (step 3, as shown in Figures 5(a) and 5(b)) without violating the accessibility condition. Thus, in order to achieve an ASP without accessibility issues, C's parent, robot E, must be deleted from the growing structure; otherwise, the accessibility issue remains for robot C.

Furthermore, as E has added robot G, this implies that G must also be deleted from the growing structure. If G had added any neighbors to the growing structure, those neighbors should also be deleted. In other words, if E is deleted from the structure, all the assembly branches starting at E must be deleted. After the branch is deleted, the search can return to the previously added robot, in this case, F. New neighbors of F can be explored (step 4, as shown in Figure 5(b)). Finally, the correct sequence $F \rightarrow C \rightarrow E \rightarrow G$ can be encountered (steps 4–6, as shown in Figure 5(b)).

For the second case of accessibility issue, where the accessibility issue for a robot w is not caused by w's parent but for another robot already added to the growing structure, consider the structure composed of robot systems I, J, K, and L, as shown in Figure 6(a).

Consider that robot systems I and J are connected to the growing structure in $I \rightarrow J$ order. For the remaining robot systems K and L, there are two possible assembly orders $I \rightarrow J \rightarrow K \rightarrow L$ or $I \rightarrow J \rightarrow L \rightarrow K$. Let's analyze each case:

(i) Sequence $I \rightarrow J \rightarrow K \rightarrow L$ (Figure 6(b)): Due to the accessibility issue, L cannot be added to the structure. Thus, L's parent, robot K must be deleted from the structure. Thus, by the end of this cycle, $I - J$ can be added to the structure as opposed to robots $K - L$.
(ii) Sequence $I \rightarrow J \rightarrow L \rightarrow K$ (Figure 6(c)): Due to the accessibility issue, K cannot be added to the structure. Thus, K's parent, robot L must be deleted from the structure. Thus, by the end of this cycle, $I - J$ can be added to the structure as opposed to robots $K - L$.

This situation occurs because the accessibility issue is not directly caused by the parents of any novel added robots (L or K), but it is caused by robot J. Thus, in order to solve the accessibility issue, J should be deleted from the structure and, eventually, a correct assembly order, as $I - K - L - J$, can be found.

These cases are comprised in the first ASP, Algorithm 1 (AssemblyOrder), which is briefly explained in the next paragraphs. A complete proof that the procedure only assembles configurations in a continuous path and without accessibility issues is presented in Section: Mathematical proofs for the proposed ASPs.

When the assembly process is initialized, there are no robots in the structure, i.e., in terms of graph theory, all vertices are initially marked as undiscovered. A robot s is selected as the starting point for the assembly process.

Then, Algorithm 1 (AssemblyOrder) is called.

Call the procedure for a starting robot s: AssemblyOrder (G, s)

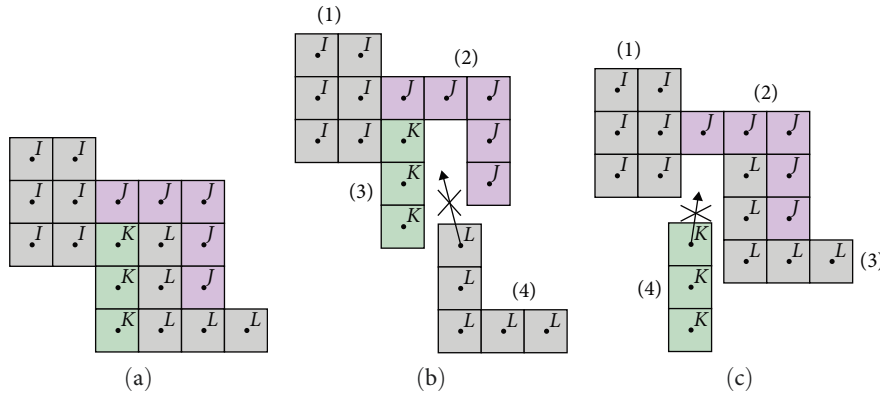

Figure 6: Due to the accessibility issue in (a), robot systems L and K cannot be added to the growing structure in cases (b) and (c), respectively.

```
1: Label s as discovered
2: For each vertex w in G.adj[s] do
3:     If vertex w is labeled as undiscovered then
4:         If w can be added to the structure (addEdge (G, w) is true) then
5:             G.Tree[s] ← w
6:             G.parent(w) = s
7:     Label w as discovered
8:     AssemblyOrder (G, w)
9:     Else
10:        deleteBranch (G, s)
11:        Label s as undiscovered
12:        Return
13: If exists w in G.adj[s] marked as undiscovered then
14:     deleteBranch (G, s)
15:     Label s as undiscovered
16: Return
```

Algorithm 1: AssemblyOrder (G, s).

(1) Line 1: Add s to the structure (label s as discovered).
(2) Lines 2 and 3: Select a neighbor robot w of s that is not in the structure (undiscovered).
(3) Line 4: Check if it is possible to add w (verify accessibility condition, which will be explained by Algorithm addEdge); if it is possible, continue, otherwise, go to line 9.
(4) Lines 5–7: Add w to the structure and mark s as parent of w.
(5) Line 8: Call recursively the procedure AssemblyOrder (G, w), i.e., continue searching "deeper" for new robots (as in a depth-first search).
(6) Lines 9–12: If w cannot be added to the structure, then w's parent, s, cannot be added to the structure.
(7) These lines comprise the first case of accessibility issues, where the accessibility issue for a robot w is caused by w's parent in the process and shown in Figure 5.
(8) Lines 13–16: Given a robot s, when the recursion explores all neighbors of s, if any of these neighbors is still not added to the structure, then s cannot also be added to the structure.
(9) These lines comprise the second case of accessibility issues, where the accessibility issue for a robot w is not direct caused by w's parent in the process and shown in Figure 6.

Finally, the main Algorithm 1 (AssemblyOrder), with the steps explained above, is herein presented. This algorithm contains the necessary steps to plan the sequence in which the robots are added to the structure.

It calls another two auxiliary procedures: Algorithm 2 (addEdge), which determines when a new robot can be added to the growing structure and Algorithm 3 (deleteBranch), which gives a strategy to delete a not allowed robot from the growing structure. These two algorithms are presented in the next subsections.

Next, two subsections present Algorithm 2 (addEdge) and Algorithm 3 (deleteBranch), respectively.

```
1: For each vertex w in G.adj[s] do
2:     If vertex w is labeled as discovered then
3:         Create edge (w, s)
4: Check if any created edges are in conflict
5: If A conflict occurs then
6:     Return false
7: Else
8:     Return true
```

Algorithm 2: addEdge (G, s).

```
1: Remove s from G.Tree[parent(s)]
2: While G.Tree[s] is not empty do
3:     Pop the first element w from G.Tree[s]
4:     Mark w as undiscovered
5:     deleteBranch (G, w)
```

Algorithm 3: deleteBranch (G, s).

2.1.2. Algorithm 2 (addEdge): Conditions to Add a New Robot to the Target Structure. The next algorithm, first presented by Salvi et al. [34], gives a procedure to determine if a new robot can be added to the growing structure. Wherever a new robot s is added to the growing structure:

(1) Construct oriented edges ending in s and starting in the s's already added neighbors.
(2) Check if these edges are in conflict, i.e., if exists north–south or east–west edge pairs.
(3) If no conflict occurs, the new robot can be added to the growing structure. Otherwise, an accessibility issue was encountered.

For example, as shown in Figure 7(a), robot J is added to the configuration composed only of robot K, thus edge $K \to J$ is created. As shown in Figure 7(b), robot L is added to the configuration composed of robots K and L, thus edges $K \to L$ and $J \to L$ are created.

As shown in Figure 7, each edge can receive one to four labels: north, south, east, and west according to the following rule. *An edge from robot A to robot B (edge $A \to B$) is labeled as:*

(1) *North: If adjacent modules $a \in A$ and $b \in B$ exist such that module a is at the north of module b ($a_x = b_x$ and $a_y = b_y + 1$).*
(2) *South: If adjacent modules $a \in A$ and $b \in B$ exist such that module a is at the south of module b ($a_x = b_x$ and $a_y = b_y - 1$).*
(3) *East: If adjacent modules $a \in A$ and $b \in B$ exist such that module a is at the east of module b ($a_y = b_y$ and $a_x = b_x + 1$).*

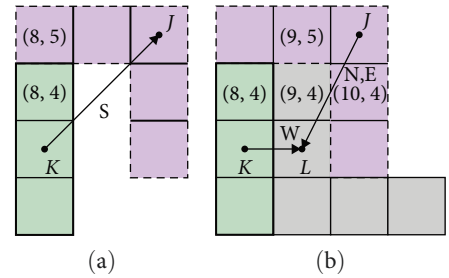

Figure 7: (a) Robot J is added to the configuration composed only by robot K, thus edge $K \to J$ is created. (b) Robot L is added, thus edges $K \to L$ and $J \to L$ are created.

(4) *West: If adjacent modules $a \in A$ and $b \in B$ exist such that module a is at the west of module b ($a_y = b_y$ and $a_x = b_x - 1$).*

Where (a_x, a_y) and (b_x, b_y) are the coordinates of modules a and b, respectively. The horizontal and vertical axes are, respectively, the x- the y-axis.

Thus, in order to determine if a new robot s can be added to the whole structure, Algorithm 2 (addEdge) considers all edges arriving in s. These edges start at s's neighbors already added to the growing structure. addEdge checks if these edges are in conflict, i.e., if exists north–south or east–west edge pairs. If no conflict occurs, the algorithm returns true, signalizing that the new robot can be added to the growing structure. Otherwise, it returns false.

2.1.3. Algorithm 3 (deleteBranch): Deleting a Branch from the Structure. For a better understanding of Algorithm 3, first presented by Salvi et al. [34], consider that an element s must be deleted from the assembly planning. An example of this procedure is shown in Figure 5(b), more precisely in step 3. In that step, robot E tried adding robot C. Due to the accessibility issue, C could not be added to the growing structure. Thus, Algorithm 1 (AssemblyOrder) called the deleteBranch procedure for robot E (C's parent). As E had added robots G, E and G were both deleted from the growing structure. If G had added any neighbors to the growing structure, those neighbors would have been deleted in an analogous process and so on.

Furthermore, in order to delete a robot from the structure, the robot must be marked as undiscovered and deleted from its parent's tree. As shown in Figure 5(b), deleteBranch(E):

(1) Marks E as undiscovered and deletes it from its parent tree, in this case from G.Tree[F].
(2) Marks G, a child of E, as undiscovered and deletes it from G.Tree[E].

The discussion suggests a recursive implementation of the deletion process, marking element's as undiscovered and deleting them from their parent's tree. Algorithm 3 summarizes this idea.

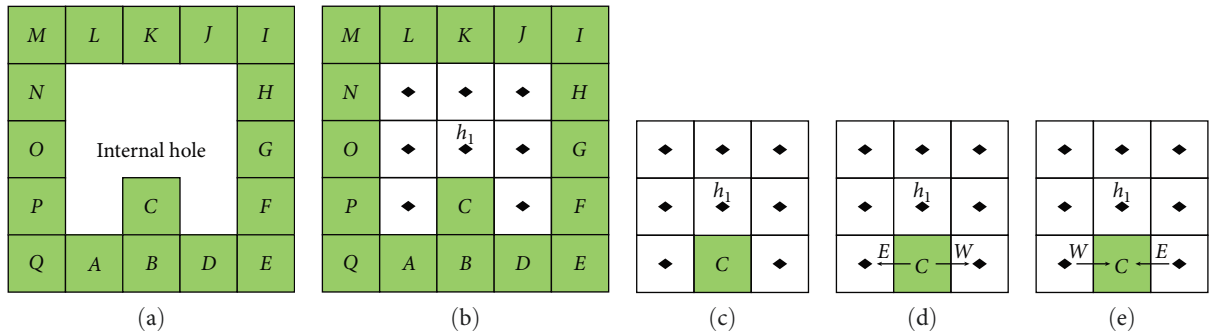

Figure 8: (a) A target structure with an internal hole. (b) If the internal hole is modeled as a singular robot system h_1, the target structure cannot be assembled. In fact, robot system C cannot be assembled in $C \to h_1$ or $h_1 \to C$ order (c). As shown, respectively, in (d) and (e), robot system C induces conflicting edges in h_1 and vice versa.

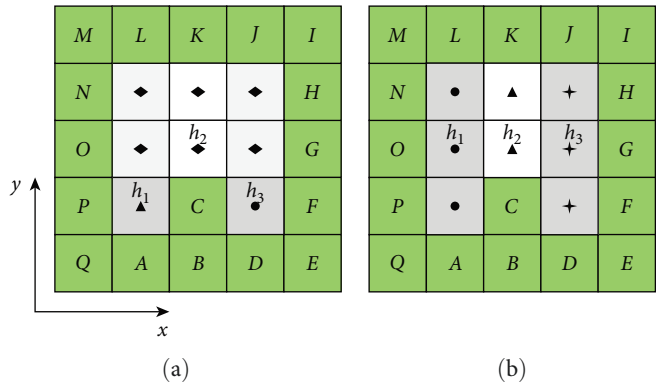

Figure 9: The internal hole, as shown in Figure 9(a), can be modeled as three continuous robot systems parallel, respectively, to the x- or y-axis, as shown in (a) and (b), respectively.

In Section 5, applications of ASP1 for configurations composed of robot systems without internal holes are presented. Also, the algorithm's main properties and mathematical proofs are discussed.

3. ASP2: Novel ASP for Continuous Construct Configurations with Internal Holes

In this section, ASP1 is extended for target structures with internal holes. In order to perform this extension, holes are regarded as one or more robot systems. Furthermore, additional rules, original contributions of this work, are herein introduced.

First, internal holes are regarded as robot systems; otherwise, accessibility issues might not be detected by addEdge. Furthermore, in order to avoid accessibility issues for robots docking to the structure, internal holes are modeled by continuous modular systems parallel to the x- or y-axis. For example, the internal hole, as shown in Figure 8(a), can be modeled as three continuous robot systems parallel, respectively, to the x- or y-axis. This is shown in Figures 9(a) and 9(b), where the holes are labeled as h_1, h_2, and h_3.

An incorrect modeling for the hole in Figure 8(a) is shown in Figure 8(b) where the hole is modeled as a unique robot system h_1. This modeling is considered incorrect because robot system C cannot be assembled in $C \to h_1$ or $h_1 \to C$ order, as shown in Figure 8(c). In fact, as shown, respectively, in Figures 8(d) and 8(e), robot system C induces conflicting edges west–east in h_1 and vice versa. Thus, addEdge would not allow neither the assembly sequence $C \to h_1$ nor the assembly sequence $h_1 \to C$.

In the next paragraphs, the additional rules for the second ASP are discussed. First, let us consider that in order to assembly connected configurations, any robot must connect to at least one other robot already in the target structure. The only exception is the first added robot which is allowed to connect only to previously added holes.

This discussion is summarized in Rule 1.

Rule 1: *If a new robot system s is added to the growing structure, then s must connect with another robot system already in the growing structure. The only exception is if s is the first robot system added to growing structure.*

Furthermore, as holes are regarded as continuous robot empty spaces, parallel, respectively, to the x- or y-axis, holes are allowed to connect with other holes at any point. This discussion is summarized in Rule 2 and an example of its application is shown in Figure 10.

Rule 2: *If a hole h is added to the growing structure, then h is not required to connect with any robot system already present in the growing structure.*

By modeling any internal hole as described above and by applying Rules 1 and 2, ASP1 can be extended successfully to

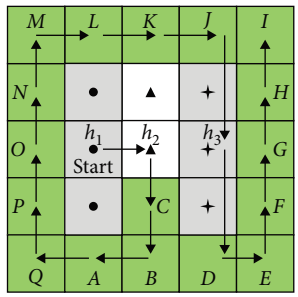

Figure 10: This figure shows an application of Rule 2. When the target configuration is assembled in the order shown by the arrows, hole h_2 does not connect with any robot already in the structure. In fact, h_2 only connects with h_1, which is another hole.

configurations with internal holes obtaining ASP2. Furthermore, if the holes are regarded as robot systems, ASP2 constructs the configuration in a continuous path, i.e., each novel robot is added by the previously added robot or by a previously added hole, which is modeled as a robot system.

Thus, incorporating Rules 1 and 2 into AssemblyOrder, Algorithm 1, a novel algorithm, Algorithm 4, is obtained. In Algorithm 4, any robot system representing a hole is referenced only as a hole, and real robot systems are referenced only as robot systems. If an element s has any holes as neighbors, those holes must be the last elements in s's adjacent list, i.e., in G.adj[s] because the model proposed in this thesis uses the holes as the last resource to transverse the configuration.

(1) Rule 1 is summarized in Lines 5–13 of Algorithm 4.

Suppose that a robot system s is trying to add a new robot system w. If s models a hole and w is not the first added robot system, Rule 1 requires that robot system w connects with another robot system z already present in the structure. Thus, in Line 6, the algorithm searches for such robot system z. If z is found, the parent of w can be the robot system z or the hole s. However, if the hole s is chosen as a parent, the ASP will not reflect the mechanical docking between robots w and z. Thus, in Lines 7 and 8, w is added to $G.Tree[z]$ and z is marked as w's parent, respectively. Then, the algorithm can search for new neighbors of w, i.e., AssemblyOrder (G, w) is called in Line 9. If s is not a hole or w is the first added robot system, the usual procedure of AssemblyOrder must be performed, which is done in Lines 11–13.

(1) Rule 2 is summarized in Lines 11–13 of Algorithm 4.

Suppose that a robot system s is trying to add a hole w. Thus, by Rule 2, w is not required to connect with any robot system already in the growing structure. This implies that the usual procedure of AssemblyOrder must be performed, which is done in Lines 11–13.

ASP2, a novel contribution of this work, is adequate for shape heterogeneous modular robots but it is also able to treat cases of modular construction not treated in literature. Consider, for example, the configurations, as shown in Figure 11. The ASPs of Paulos et al. [29] and Seo et al. [30, 31] are not able to assemble any of these configurations because these configurations have internal holes and narrow corridors. However, except for the configuration, as shown in Figure 12(d), the configurations can be assembled without violating the accessibility condition for any module. In other words, the configurations can be assembled without accessibility issues.

```
1: Label s as discovered
2: For each vertex w in G.adj[s] do
3:    If vertex w is labeled as undiscovered then
4:       If w can be added to the structure (addEdge (G, w) is
         true) then
5:          If s is a hole and w is not a hole or the first added
robot system then
6:             If exists a discovered robot system z ∈ G.adj[w]
               then
7:                G.Tree[z] ← w
8:                G.parent(w) = z
9:             AssemblyOrder (G, w)
10:         Else
11:            G.Tree[s] ← w
12:         G.parent(w) = s
13:         AssemblyOrder (G, w)
14:      Else
15:         deleteBranch (G, s)
16:         Label s as undiscovered
17:         Return
18: If exists w in G.adj[s] marked as undiscovered then
19:    deleteBranch (G, s)
20:    Label s as undiscovered
21:    Return
```

Algorithm 4: AssemblyOrder (G, s) configurations with internal holes.

ASP2 can assemble the first three configurations, as shown in Figure 11, without violating the accessibility condition for any module. Furthermore, it does not return an assembly sequence, as shown in Figure 12(d). This result is correct because this configuration cannot be assembled without violating the accessibility condition for at least one module. The assembly sequences for the first three cases, as shown in Figure 11, are shown by the arrows, as shown in Figure 12(a)–12(c), respectively.

In Section 5, applications of Algorithm 4 for path continuous assembly of configurations with internal holes are presented. Also, the algorithm's main properties and mathematical proofs are discussed.

4. ASP3: Novel ASP for Allowing Path Discontinuous Assembly

In this section, a novel ASP for allowing path discontinuous assembly, ASP3, is obtained from ASP2. Given the importance of this case for modular construction, i.e., for aggregating module by module to assembly the target structure, the

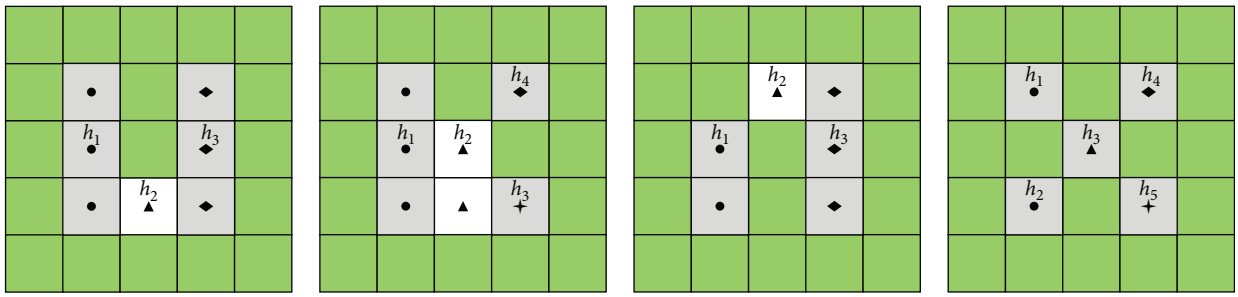

Figure 11: These configurations can be assembled with the proposed ASP without violating the accessibility condition. These configurations cannot be assembled with the ASPs of Paulos et al. [29] and Seo et al. [30, 31].

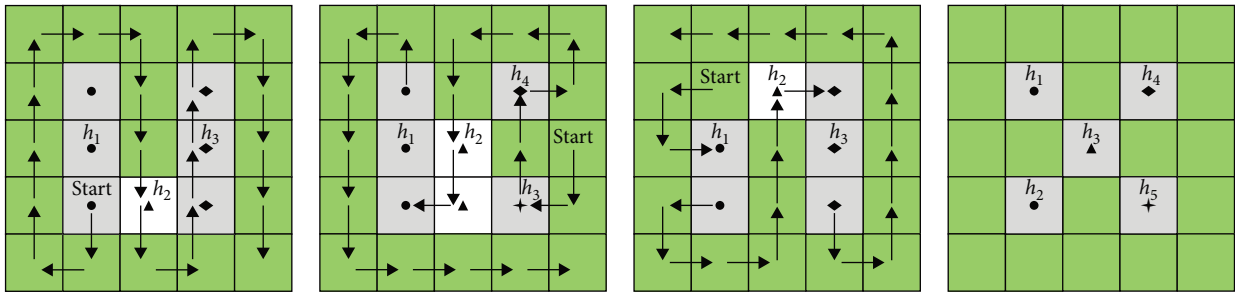

Figure 12: The configurations in the first three cases can be assembled with the proposed ASP, without violating the accessibility condition. Also, the proposed ASP can detect that the configuration in the fourth case cannot be assembled without violating the accessibility condition.

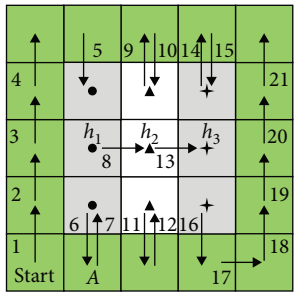

Figure 13: An example of ASP3 with the novel rule for path discontinuous assembly.

main examples herein presented focus on this case. However, the results herein discussed can be also extended to robot systems.

Consider the ASP, as shown in Figure 13. This ASP utilizes the added holes as links from one side to another of the already assembled structure, thus providing a way for assembling the configuration on a noncontinuous path.

Regard that this kind of situation cannot be treated with the first two ASPs.

Thus, in order to allow path discontinuous assemblies, in ASP3, holes can be allowed to add one or more modules to the growing structure if these additions are possible. As shown in Figure 13, in order to include this case, ASP3 recursion must allow the assembly sequence to return to a hole if this hole has already added a module to the growing structure. Then, this hole can try to add another module to the growing structure.

This novel rule is incorporated into Algorithm 4, resulting in Algorithm 5. More specifically, the rule is summarized in Lines 20–24 of Algorithm 5.

The novel lines in Algorithm 5 are modifications of Lines 18–21 of Algorithm 4. However, these lines were first introduced in Algorithm 1 and stated that: *given a robot system s, when the recursion explores all neighbors of s, if any of these neighbors are still not added to the structure, then s cannot also be added to the structure*. In the novel version, if the parent of s is a hole, the algorithm does not delete s and returns to the hole, allowing another robots to be added to the growing structure; this is done in Lines 20–24 of Algorithm 5.

Furthermore, it can be noticed that ASP3 is still able to find path continuous sequences. An example is shown in Figure 14 where another case of modular construction is presented, as this configuration presents internal holes and narrow corridors. Thus, it cannot be assembled with the ASPs of Paulos et al. [29] and Seo et al. [30, 31].

It is important to notice that ASP3 does not guarantee that all possible path discontinuous assemble sequences will be found, but it solves some cases that the main ASPs found in the literature cannot solve. More details and discussions about the results and the limitations of the three proposed ASP will be presented in Section 5.

```
1: Label s as discovered
2: For each vertex w in G.adj[s] do
3:     If vertex w is labeled as undiscovered then
4:         If w can be added to the structure (addEdge (G, w) is true) then
5:             If s is a hole and w is not a hole or the first added robot system then
6:                 If exists a discovered robot system z ∈ G.adj [w] then
7:                     G.Tree[z] ← w
8:                     G.parent(w) = s
9:                 AssemblyOrder (G, w)
10:            Else
11:                G.Tree[s] ← w
12:                G.parent(w) = s
13:            AssemblyOrder (G, w)
14:        Else
15:            deleteBranch (G, s)
16:            Label s as undiscovered
17:            Return
18: If exists w in G.adj[s] marked as undiscovered then
19:     deleteBranch (G, s)
20:     If G.parent(s) is a hole, then
21:         If exists a discovered robot system z ∈ G.adj[w] then
22:             G.Tree[z] ← w
23:             G.parent(w) = s
24:         Return
25:     Else
26:         Label s as undiscovered
27:         Return
```

Algorithm 5: AssemblyOrder (*G*, *s*) configurations with internal holes.

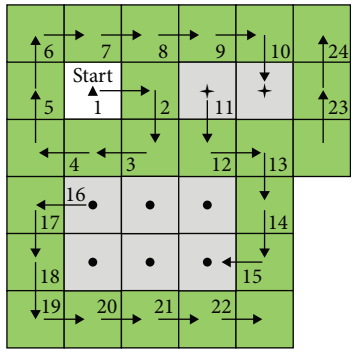

Figure 14: A further example of the third ASP in modular construction.

5. Results and Discussion

In this section, properties and implementation results for the three ASPs are presented. Furthermore, mathematical proofs of the *termination*, *soundness*, *and complexity* of the algorithms composing the three ASPs are discussed.

Table 2: ASP1 computational time for path continuous assembly.

Target configuration	Computational time	Starting point
Figure 2(a)	8 ms on average	Arbitrary

Table 3: ASP2 computational time for path continuous assembly.

Target configuration	ASP result and starting point	Computational time (ms)
Figure 9(a)	See Figure 10	0.3
Figure 3(a)	See Figure 3	1.3

5.1. Implementation Results. The three ASPs proposed in this work were implemented in C++. The software takes as input the sites composing each robot system of the target structure and the graph representing the robot system connections. It returns an assembly plan that specifies a complete order for building the target structure. All computer simulations were performed in a 4-GB, 1.6-GHz machine.

First, the work considers the path continuous assembly cases: configurations without internal holes (ASP1) and configurations with internal holes (ASP2). In these cases, both configurations composed for modular robots or by robot systems are presented. Then, in order to illustrate the path discontinuous case (ASP3), the modular construction is presented. In this section, the main results for both of these cases are discussed.

5.1.1. Path Continuous Case: ASP1 and ASP2 Results. For the path continuous case, configurations composed for robot systems and by modular robots are considered. Algorithms 1 and 4 are the main algorithms for these cases, more specifically, for configurations without and with internal holes, respectively.

First, Algorithm 1 was applied to target structures without holes and composed for robot systems. Considering the configuration, as shown in Figure 2(a), given an arbitrary choice of the starting robot system for the assembling process, the computational time was 8 ms on average for the ASP to compute a correct assembly sequence, as shown in Table 2. However, this algorithm cannot be applied to configurations with internal holes. This result was first shown by Salvi et al. [34].

Then, Algorithm 4 was applied to configurations with internal holes and composed for robot systems. Table 3 presents the average time for the ASP2 to compute a correct assembly sequence, as shown in Figure 9(a) and Figure 3(a). It is important to notice that neither of these configurations can be assembled with the main ASPs in literature [29, 31]. In fact, the first configuration has a narrow corridor and the second one is composed for robot systems; neither of these cases can be solved with the main ASPs in literature. Furthermore, in all the novel ASPs proposed in this work, the

Table 4: ASP3 computational time for a configuration without narrow corridors.

Target configuration	Computational time	Starting point
Figure 13	6 ms on average	Arbitrary

Table 5: ASP3 computational time for the examples with narrow corridors.

Target configuration	ASP result and starting point	Computational time (ms)
Figure 11(a)	See Figure 12(a)	3.706
Figure 11(b)	See Figure 12(b)	5.384
Figure 11(c)	See Figure 12(c)	3.375
Figure 12(d)	It does not exist	29
Figure 14	See Figure 14	8.87

assembly starting point can be freely chosen, which cannot be obtained with the main ASPs in literature either.

There is no guarantee that ASP1 or ASP2 will return all possible assembly sequences for a given configuration.

However, any sequence returned by these ASPs is a valid sequence.

To the best of the author's knowledge, this is the first work that shows, considering the accessibility condition:

(1) How to assemble target structures composed for shape heterogeneous modular robot systems with internal holes, the case without internal holes was treated by Salvi et al. [34].
(2) How to assemble planar target structures with narrow corridors.
(3) How to assemble configurations by choosing the ASP starting point.

5.1.2. ASP for Allowing Path Discontinuous Assembly: ASP3 Results. The assembly planning for allowing path discontinuous assembly is modeled in ASP3. The main algorithm for this case is Algorithm 5. In order to illustrate this ASP for path discontinuous assembly, the modular construction was chosen. It is important to notice that ASP3 can be applied for configurations with and without internal holes.

For the configuration, as shown in Figure 13 and Table 4, the computational time was 6 ms on average for the ASP to compute a correct assembly sequence, including the one, as shown in Figure 13. It is important to notice that this kind of discontinuous assembly is not solved in the literature.

More examples are shown in Table 5, which presents the average time it takes ASP3 to determine a correct assembly sequence. As previously stated for Algorithm 4, for the path continuous case, Algorithm 5 is able to assemble the first three configurations, as shown in Figure 11, and the configuration, as shown in Figure 14, without violating the accessibility condition for any module. These four cases cannot be treated with the main ASPs in literature [29, 31] because they have narrow corridors. Also, Algorithm 5 detected that does not exist an adequate ASP for the configuration, as shown in Figure 12(d).

There is no guarantee that the proposed ASP will return all possible assemble sequences for a given configuration. However, any sequence returned by this ASP does not violate the accessibility condition for any module.

It is important to notice that for configurations without narrow corridors, the expected average time of ASP3, ASP2, or ASP1 is greater than the main ASPs in literature [29, 31]. In fact, the ASPs proposed by Paulos et al. and Seo et al. have polynomial-time complexity. However, with the ASPs by Paulos et al. [29] and Seo et al. [31], a starting point for the assembly process cannot be freely chosen. On the other hand, all the ASPs proposed in this work allow the selection of a starting point for the assembly process.

As it is explained in the next section, where the complexity analysis for the proposed ASPs is presented, the average time depends mostly on the number of sequences evaluated during the assembly process. To the best of the author's knowledge, this is the first work that shows, considering the accessibility condition:

(1) An ASP for modular robots allowing discontinuities in the assembly path.

Furthermore, as future work, the ASPs proposed in this paper can be extended to complex 3D shapes, a first application for planar vertical structures can be found in a study by Salvi et al. [32].

5.2. Mathematical Proofs for the Proposed ASPs. In this section, mathematical proofs about the properties of the three proposed ASPs, original contributions of this work, are introduced. It is shown that the proposed methods are correct and that they finish, however, as, in the worst case scenario, they can return empty sequences and they are not complete. First, the auxiliary procedures addEdge and DeleteBranch are discussed; then, each ASP is analyzed.

5.2.1. Properties of Algorithm 2 (addEdge). An analysis of Algorithm 2 is herein presented. First, let us discuss the *termination* of addEdge by analyzing what occurs in each line of Algorithm 2.

Lines 1–3 end when the cycle *for* is completed for graph G. In the worst scenario, G is a complete and finite graph with n vertices, thus it has $\frac{n(n-1)}{2}$ edges. In this case, as G is presented in the adjacency list form, the cycle *for* ends after $n(n-1)$ operations. Also, considering m as the number of novel edges created in Line 3, then m is limited by the number of edges in G, i.e., $m \leq \frac{n(n-1)}{2}$.

Line 4 ends when all pairs of edges created in Line 3 are tested. If m edges are generated in the previous step, then the number of tests is limited by $\frac{m!}{(m-2)!2!} < m(m-1)$.

If a conflict occurs, then Lines 5 and 6 are performed and the algorithm returns false. Otherwise, Lines 7 and 8 are performed and the algorithm returns true.

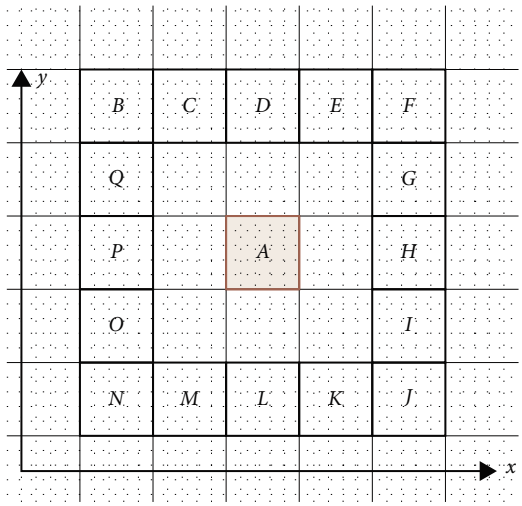

Figure 15: Module A forms a narrow corridor with any of the modules from B to Q.

In the next paragraphs, it is discussed how addEdge detects accessibility issues in the proposed ASP, i.e., its *soundness* in terms of detecting accessibility issues. When Algorithm 1 calls addEdge for a robot s being added to the growing structure, addEdge is either capable of detecting accessibility at the moment that s is added to the growing structure, or, in the worst case scenario, addEdge will detect accessibility issues when the last robot is added by Algorithm 1 to the growing structure.

Consider a configuration C, its graph G, and a robot system s belonging to C. A call of function addEdge is started when AssemblyOrder, Algorithm 1, tries to add a robot system s to the ASP constructing C.

Thus, in Lines 1–3, addEdge creates edges between s and its already assembled neighbors. In Line 4, addEdge checks if there exists a north/south or an east/west pair of edges. If a conflict occurs, addEdge returns false; otherwise, it returns true. Therefore, for each robot system s being added by AssemblyOrder, addEdge guarantees that there are no direct neighbors of robot system s at its north and south or at its east and west. Thus, s is not blocked by north/south or east/west neighbors.

Let us now examine if the north/south or the east/west checking is a sufficient condition in order to detect that a module passes through a narrow corridor during assembly. In order to perform this analysis, the creation of a narrow corridor is discussed.

Consider module A, as shown in Figure 15, and consider a generic configuration C containing A. If any module in the set B–Q, as shown in Figure 15, is added to configuration C, then a narrow corridor between A and the novel module is created.

First, let us analyze the case where the narrow corridor is created in the generic configuration C between module.

A in position (i, j) and one of the modules D, H, L, or P, which are, respectively, in positions $(i-2, j)$, $(i, j+2)$, $(i+2, j)$, and $(i, j-2)$, as shown in Figure 16(a).

Figure 16(b) shows that if configuration C has no internal holes or if they are regarded as auxiliary robots, a novel module is eventually added in position $(i-1, j)$, $(i, j+1)$, $(i+1, j)$, and $(i, j-1)$. Thus, function addEdge will create north/south or west/east edges arriving at the novel module, detecting the accessibility issue.

Second, let us consider the case where a narrow corridor is created in configuration C between module A and the other modules, as shown in Figure 15, i.e., modules B, C, E, F, G, I, J, K, M, N, O, or Q.

In order to perform the analysis, let us consider modules in the narrow corridor created between modules A, which is in position (i, j) and F, which is in position $(i-2, j+2)$; the other cases are similar. An example is shown in Figures 17(a) and 17(b).

It is important to notice that the presence of a narrow corridor, as shown in Figure 17(b), only represents an accessibility issue if a novel robot transverse this narrow corridor and then is added to some position into the structure, as shown in Figure 18.

As shown in Figure 18, if the novel robot transverse the narrow corridor between A and F, it will be surrounded by the modules inside the path or it will close the path, as shown in Figure 17(b). The path between A and F must exist because the configuration is connected; furthermore, because holes are treated as continuous structures parallel to x- or y-axis, the ASP tries to fully cover the remaining space with novel modules. Thus, in the worst scenario, the last module that is added inside the closed path will be blocked by north/south and west/east neighbors implying that addEdge eventually detects the accessibility issue.

Furthermore, as in all ASPs, an accessibility issue is followed by a call to DeleteBranch, and sequential deletions are performed until the first robot system violating the accessibility condition is deleted from the growing structure.

5.2.2. Properties of Algorithm 3 (DeleteBranch). Algorithm 3 is executed in order to delete elements from the ASP created by Algorithm 1. It will be shown that Algorithm 1 always returns connected and finite trees without accessibility issues. Thus, Algorithm 3 is executed on a finite and connected tree.

Suppose that Algorithm 3 is executed on a tree G with n vertices. In the worst scenario, the first call of Algorithm 3 is performed for the tree root, a generic robot system s.

As G is a tree with n elements, it has $n-1$ edges. Also, for each call of Algorithm 3, when Line 3 is performed, a tree edge is eliminated along with its tree leaf. Thus, after $n-1$ calls of Algorithm 3, all the $n-1$ tree edges are eliminated. At this point, each *while* cycle is closed, finishing the entire procedure.

In order to show the *soundness* of Algorithm 3, it is necessary to show that the ASP tree is not disconnected while performing the deletion process. This proof is discussed along with the soundness of Algorithm 1 is discussed.

5.2.3. Properties of Algorithm 1: Main Algorithm for ASP1. In this section, the mathematical properties of Algorithm 1 are discussed.

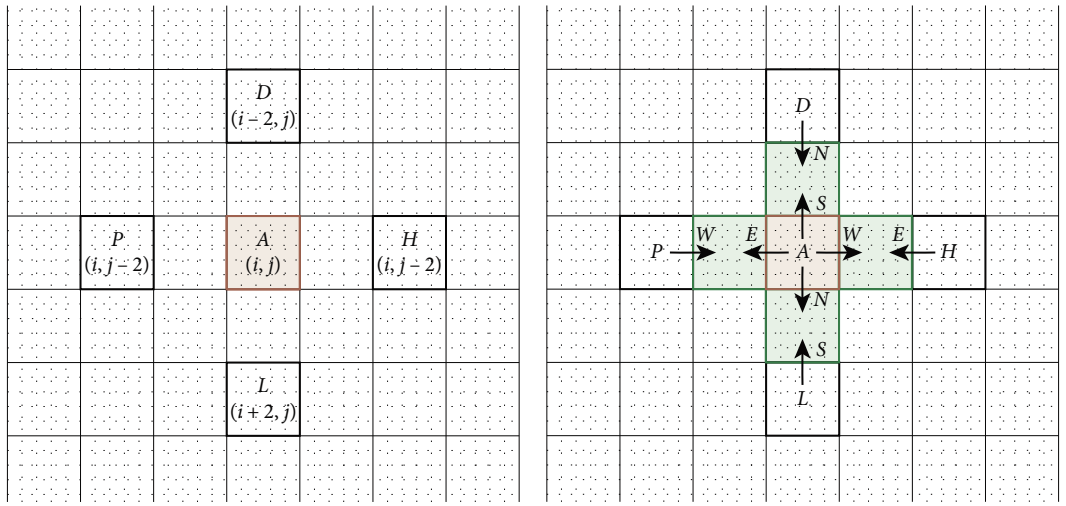

Figure 16: Detection of the narrow corridors between module A and modules D, H, L, and P.

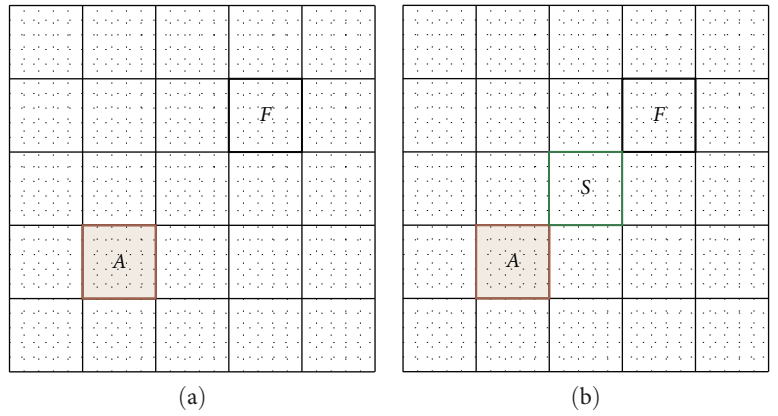

Figure 17: Conditions for obtain a narrow corridor.

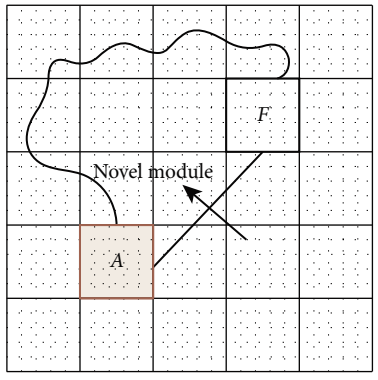

Figure 18: Conditions for obtain a narrow corridor.

First, let us discuss the termination of Algorithm 1. Given a configuration C with n robot systems, Algorithm 1 ends if an ASP with n elements is founded or if all possible sequences are explored.

In the first case, as Algorithm 1 finds an ASP with n elements, all robot systems belonging to configuration C are marked as discovered. Thus, each open call of function AssemblyOrder is closed when the cycle *for* in Line 2 ends. In fact, as there are no undiscovered robot systems, the conditions in Line 3 and in Line 16 are not true. Therefore, for each open call of AssemblyOrder, the cycle *for* in Line 2 ends without further operations thus, the algorithm ends.

In the second case, in the worst scenario, the algorithm ends when all possible vertices combinations are explored, i.e., when all spanning trees are visited. For a complete graph with n vertices, the number of these trees is n^{n-2}. However, for the modular robot case, the configuration graph is usually not complete; therefore, the number of these possibilities is lower and can be determined by the Kirchhoff's matrix tree theorem [57].

In order to show the soundness of Algorithm 1, it can be demonstrated that it always returns connected trees without accessibility issues. Consider an ASP S_n with n elements returned by Algorithm 1. First, the accessibility condition is shown by induction in n.

For $n = 1$, the property is true. In fact, if $n = 1$, when the algorithm starts, Line 1 marks this element as discovered.

Then, as no undiscovered elements remain, the algorithm ends. Thus, for $n = 1$, the algorithm returns a tree with a single element that is connected and does not present accessibility issues.

Assume that for some $n \in N$, the algorithm returns ASPs without accessibility issues for every natural $i < n$. Consider an ASP S_n with n elements, obtained from an ASP S_{n-1} with $n-1$ elements by adding a new element k. By the induction hypothesis, S_{n-1} presents no accessibility issues. Thus, if S_n presents some accessibility issue, it must be caused by element k. Consider s the parent of k in this step of the process. There are two possibilities:

(1) If k causes no accessibility issues, Line 4 of Algorithm 1 is satisfied. Thus, k is added to the structure in Lines 5–7. Then, AssemblyOrder (G, k) is called in Line 8. Because k has no undiscovered neighbors, AssemblyOrder (G, k) ends as all the others AssemblyOrder calls. Therefore, an ASP S_n with n elements presenting no accessibility issues is returned.

(2) If k causes an accessibility issue, deleteBranch (G, s) is called in Line 10. Thus, the ASP branch starting in s, parent of k, is deleted. Then, s is also deleted from the structure in Line 11. At this point, the assembly ASP presents $l < n$ elements; thus, by the induction hypothesis, it presents no accessibility issues.

The algorithm returns to s's parent, an element a. If a has unvisited and unexplored neighbors, they are explored at this point. If, at the end of the neighborhood checking for a, s and all the other a's neighbors are added to the structure, AssemblyOrder (G, a) ends. At this point, the algorithm returns a partial ASP with less than n elements. Thus, by the induction hypothesis, it presents no accessibility issues.

If, at the end of the neighboring checking for a, s or any other a's neighbors are not added to the structure, Lines 13–16 are executed. Thus, s and a are deleted, along with the branch starting in a. As in the previous case, the algorithm returns a partial ASP with less than n elements; thus, by the induction hypothesis, it presents no accessibility issues.

The algorithm then returns to a's parent, an element b, and the process described in the two paragraphs above is performed for b and so on. In the worst scenario, an element j cannot be added to the structure at any step thus, the resulting ASP is empty. In order to discuss the empty case, consider that all possible ASP but one was already tested. Suppose that in the final possible sequence, an element i tries to add j. As j cannot be added to the structure at any point, i and its entire ASP branch are deleted. Then, the parent of i is deleted by the condition in Lines 13–16 and so on, until all elements are deleted. Thus, the resulting ASP is empty.

Therefore, when Algorithm 1 ends, the ASP presents n elements without accessibility issues or it is empty.

In the next paragraphs, it is shown that the ASP is a tree. First, it is important to notice that when an element w with parent s is added to the structure, a single edge is created. In fact, in Line 5, w is added to G.Tree[s], i.e., edge (s, w) is created. Thus, for each added element, only one edge is created. Therefore, if n elements are added, $n-1$ edges are created.

Also, no cycles are created because if a vertex k is incident to two edges, k has two parents x and y in the process. This is an absurd because after k is added to the ASP by an element x, k is marked as discovered. Thus, another element y cannot add k unless it has been remarked as undiscovered. In order to marked k as undiscovered, function deleteBranch (G, x) must be called, which can only occur in Lines 10 or 14. However, after each of these lines, x is marked as undiscovered. Thus, even in this case, k has only one parent, y. Therefore, no cycles are created in the ASP process. As the process creates $n - 1$ edges and no cycles when n elements are added, the ASP constructs a tree.

Furthermore, when function deleteBranch is called, it performs a DFS search marking tree leaves as undiscovered. Therefore, the ASP remains a tree when this function is called. In fact, suppose that deleteBranch is called for some element s. First, in its Line 3, function deleteBranch pops the first element w from G.Tree[s], i.e., edge (w, s) is deleted. Then, in Line 5, deleteBranch (G, w) pops the first element z from G.Tree[w], i.e., edge (z, w) is deleted. This process continues until function deleteBranch reaches an element d whose G.Tree[d] is empty, i.e., d is a leaf of the ASP tree. Then, function deleteBranch returns to d's parent, some element c. In other words, it returns to its open call deleteBranch (G, c). However, this call is only closed when all elements in G.Tree [c] are deleted. Thus, when deleteBranch (G, c) is closed, c is a leaf of the ASP tree. The same occurs for each of function deleteBranch calls. Thus, this recursive deletion process can be regarded as successively deleting tree leaves. Therefore, the ASP remains a tree after all the deletions.

In terms of complexity, in the worst scenario, the algorithm ends when all possible vertices combinations are explored, i.e., when all spanning trees are visited. For a complete graph with n vertices, the number of these trees is n^{n-2} thus, the complexity is $O(n^{n-2})$. However, for the modular robot case, the configuration graph is usually not complete; therefore, the number of these possibilities is lower and can be determined by the Kirchoff's matrix tree theorem [57].

5.2.4. Properties of Algorithm 4—ASP2. In this section, the properties of Algorithm 4 are discussed.

The termination of Algorithm 4 follows directly from the termination of Algorithm 1. In fact, as holes are modeled as auxiliary robot systems, the unique difference between Algorithm 4 and Algorithm 1 is how the robot systems are connected to the holes, thus, and holes connect to each other, thus conditions for termination follows from ASP1, considering that in the worst cases scenario, all possible sequences are visited.

Furthermore, ASP2, as ASP1, returns a tree without accessibility issues or its soundness. As Algorithm 4 is obtained from Algorithm 1 by adding the conditions in Lines 5–14, in order to show that Algorithm 4 is sound, it can be shown that these conditions do not introduce accessibility issues or disconnect the graph.

Consider that a robot system s tries to add a robot system w to a configuration C being assembled by Algorithm 4. Thus, in Line 4, the accessibility condition for w is verified. Therefore, if w is added to the configuration, no accessibility conditions exist. Let us analyze the cases where s is a hole because the other cases are identical to Algorithm 1.

(1) If s is a hole and w is a hole, in Lines 11–12, s adds w to the ASP and only one edge is created between s and w.
(2) If s is a hole and w is not a hole, w must connect to another robot system z in the ASP. If z exists, in Lines 6–8, z adds w to the ASP, and only one edge is created between z and w. Otherwise, s is eliminated from the ASP in Lines 14–17. Thus, this process does not disconnect the ASP or create accessibility issues.

It is important to remark that, in both cases, when w is added to the ASP, only one edge is created, thus the ASP remains a tree without accessibility issues.

Therefore, as the other cases are identical to Algorithm 1, Algorithm 4 returns a tree without accessibility issues.

Furthermore, as stated for Algorithm 1, in the worst scenario, Algorithm 4 ends when all possible vertices combinations are explored, i.e., when all spanning trees are visited, thus both ASPs have the same complexity.

5.2.5. Properties of Algorithm 5—ASP3. Algorithm 5 differs from Algorithm 4 in Lines 20–27. Thus, in order to show that Algorithm 5 ends, the condition in these lines must be analyzed. Consider a configuration C being assembled by Algorithm 5 and consider that a robot system s, added to the ASP by a hole h_1, cannot add some of his neighbors to configuration C. At the end of the neighborhood checking for s, there exist unassembled neighbors, thus the condition in Line 18 is satisfied.

Therefore, all the ASP branch starting at s is deleted in Line 19.

Also, as the step of neighborhood checking for s ends, the function AssemblyOrder (G, s) is closed. Then, Algorithm 5 returns to his previously open call, AssemblyOrder (G, h_1). Thus, function AssemblyOrder (G, s) is closed before returning to AssemblyOrder (G, h_1), where the neighborhood checking for h_1 can be continued. Furthermore, AssemblyOrder (G, h_1) is closed when its cycle *for* in Line 2 is finished.

In the other case, where a robot system s does not have a hole as a parent was analyzed for Algorithm 4. In the worst scenario, the algorithm ends when all possible vertices combinations are explored, i.e., when all spanning trees are visited. For a complete graph with n vertices, the number of these trees is n^{n-2}. However, for the modular robot case, the configuration graph is usually not complete; therefore, the number of these possibilities is lower and can be determined by the Kirchoff's matrix tree theorem [57].

In order to show that Algorithm 5 is sound, i.e., that returns sequences without accessibility issues the same lines as for the termination case must be analyzed. Consider again a configuration C been assembled by Algorithm 5. Consider that a robot system s, added to the ASP by a hole h_1, cannot add some of its neighbors to configuration C. At the end of the neighborhood checking for s, there exists not assembled neighbors thus, the condition in Lines 23–31 is satisfied. Therefore, all the ASP branch starting at s is deleted in Line 24.

Also, as the step of neighborhood checking for s ends, the function AssemblyOrder (G, s) is closed. Then, Algorithm 5 returns to his previously open call, AssemblyOrder (G, h_1).

As AssemblyOrder (G, h_1) already examined the accessibility of s in Line 4, no accessibility issues are found when s is added to the structure. Furthermore, if s is a hole, only one edge is created between s and h_1. If s is not a hole, by the condition in Lines 25–27, s must connect to a robot system z in configuration C. In other words, only one edge is created between s and z. Therefore, the ASP remains a tree because only one edge is created for each new element and it presents no accessibility issues. In the worst case scenario, at some point, deleteBranch is called for hole h_1 and s is then deleted from the structure.

Finally, the complexity analysis is equal to the analysis of the first two ASPs.

6. Conclusions

This work presents three novel ASP for modular robots, which can be successfully applied for different scenarios and that are able to assemble novel classes of configurations [32]. In fact, with respect to the main ASP for modular robots found in literature [29–31], the main goal of this work is to extend the classes of rectangular modular robots configurations, which can be assembled without violating the accessibility condition. More precisely, the main ASPs in literature do not allow configurations with narrow corridors, i.e., corridors which are too narrow for a robot to transverse. Furthermore, these ASPs do not allow preassembled substructures or free selection of the assembly starting point.

In this work, the classes of rectangular modular robots configurations which can be assembled without violating the accessibility condition when compared to the main centralized ASPs in literature for modular robots [29–31] were significantly extended. Furthermore, as the classical ASPs for assembling mechanical parts were not applied for modular robots, the present work can be only directly compared with the works of Paulos et al. [29] and Seo et al. [30, 31]. Examples of configurations whose ASPs can be only be achieved with the methods proposed in this paper are presented in Section 5.

First, the novel ASP for path continuous construction of configurations without internal holes, ASP1, first introduced by the Salvi et al. [34] is discussed. Then, the case with internal holes, ASP2, an original contribution of this work, is presented. Furthermore, application cases and novel mathematical proofs, which are also original contributions of this work, are presented for both ASPs.

(1) These two ASPs are implemented by two main algorithms which consider configurations without and

with internal holes, respectively: Algorithm 1, first introduced by Salvi et al. [34], and Algorithm 4, a novel contribution of this work.

(2) Original auxiliary procedures, first introduced by Salvi et al. [34], are also discussed. Algorithm 2 (addEdge), which determines when a new robot system can be added to the growing structure, and Algorithm 3 (deleteBranch), which gives a strategy to delete a not allowed robot system from the growing structure, are introduced in a study by Salvi et al. [34].

(3) Examples of application and mathematical proofs for these two ASPs are also presented.

Furthermore, a novel ASP for path discontinuous assembly, ASP3, is introduced. This case is illustrated for modular construction, i.e., aggregating module by module to assemble the target structure.

(1) The novel ASP is implemented by its main algorithm: Algorithm 5. It can be applied to configurations with and without internal holes.

(2) Original implementation results and mathematical proofs for ASP3 are also presented. The proposed ASP has a bigger complexity when compared to the literature for modular construction; however, it is able to assembly novel classes of configurations.

To the best of the authors' knowledge, this is the first work that presents, considering the accessibility condition, how to obtain a centralized ASP for assembling planar structures composed of rectangular modules with the following characteristics:

(1) With narrow corridors.

(2) Composed of subsets of preassembled modules and configurations with internal holes.

(3) Choosing the ASP starting point.

(4) Achieving discontinuous assembly paths.

Furthermore, all the ASPs herein proposed satisfy the accessibility condition, i.e., any rectangular module cannot pass through a gap only as large as a side of a module between two physical robots already assembled in the structure. Finally, the complexity of the proposed ASPs is presented: for n robots, the worst case for the number of possible sequences evaluated is n^{n-2} considering a complete graph. However, for the modular robot case, the configuration graph is usually not complete; therefore, the number of these possibilities is lower and can be determined by the Kirchhoff's matrix tree theorem [57]. Thus, the average time depends mostly of the number of sequences evaluated during the assembly process.

6.1. Further Works. The following suggestions are presented for future work:

(1) Extension of the three proposed ASPs for other types of modules and for different types of modules constraints.

(2) Extension of the ASP for modular construction to three dimensional structures. In this case, stability analysis should be integrated to the proposed ASP.

(3) Implementation of the proposed methods, in modular platforms, integrating them with path planning methods and application of statistical methods to measure the success of achieving the assembly sequence in uncertain environments.

Data Availability

No underlying data were collected in this study because the main contribution of this paper is a set of new ASP algorithms. These algorithms and all the mathematical proofs are presented in this work. Many examples are provided along the manuscript.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research has been partially funded by the Brazilian National Council for Scientific and Technological Development (CNPq).

References

[1] Q. Su, "A hierarchical approach on assembly sequence planning and optimal sequences analyzing," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 1, pp. 224–234, 2009.

[2] M. V. A. R. Bahubalendruni and B. B. Biswal, "A review on assembly sequence generation and its automation," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 230, no. 5, pp. 824–838, 2016.

[3] H. G. Lv and C. Lu, "An assembly sequence planning approach with a discrete particle swarm optimization algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 5–8, pp. 761–770, 2010.

[4] S. Ghandi and E. Masehian, "A breakout local search (BLS) method for solving the assembly sequence planning problem," *Engineering Applications of Artificial Intelligence*, vol. 39, pp. 245–266, 2015.

[5] A. Bourjault, *Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires*, PhD thesis, Universite de Franche-Comte, France, 1984.

[6] T. De Fazio and D. Whitney, "Simplified generation of all mechanical assembly sequences," *IEEE Journal on Robotics and Automation*, vol. 3, no. 6, pp. 640–658, 1987.

[7] R. H. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994.

[8] L. S. H. de Mello and A. C. Sanderson, "And/or graph representation of assembly plans," in *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, Fifth

AAAI National Conference on Artificial Intelligence, pp. 1113–1119, AAAI Press, USA, 1986.

[9] S. Lee and Y. G. Shin, "Assembly coplanner: cooperative assembly planner based on subassembly extraction," in *Computer-Aided Mechanical Assembly Planning*, L. S. Homem de Mello and S. Lee, Eds., vol. 148 of *The Springer International Series in Engineering and Computer Science*, pp. 315–339, Springer, Boston, MA, 1991.

[10] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: the motion space approach," *Algorithmica*, vol. 26, no. 3-4, pp. 577–601, 2000.

[11] M. Wu, Y. Zhao, and C. Wang, "Knowledge-based approach to assembly sequence planning for wind-driven generator," *Mathematical Problems in Engineering*, vol. 2013, Article ID 908316, 7 pages, 2013.

[12] N. Zhang, Z. Liu, C. Qiu, and J. Tan, "A novel assembly sequence design mechanism for assembly sequence planning," in *Proceedings of the 2021 8th International Conference on Industrial Engineering and Applications (Europe)*, ICIEA 2021-Europe, pp. 109–114, Association for Computing Machinery, New York, NY, USA, 2021.

[13] J. Yu, L. D. Xu, Z. Bi, and C. Wang, "Extended interference matrices for exploded view of assembly planning," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 279–286, 2014.

[14] V. S. S. V. Prasad, M. Hymavathi, C. S. P. Rao, and M. V. A. R. Bahubalendruni, "A novel computative strategic planning projections algorithm (CSPPA) to generate oblique directional interference matrix for different applications in computer-aided design," *Computers in Industry*, vol. 141, Article ID 103703, 2022.

[15] G. A. Kumar, M. V. A. R. Bahubalendruni, V. S. S. Vara Prasad, D. Ashok, and K. Sankaranarayanasamy, "A novel geometric feasibility method to perform assembly sequence planning through oblique orientations," *Engineering Science and Technology an International Journal*, vol. 26, no. 4, Article ID 100994, 2022.

[16] T. Dong, R. Tong, L. Zhang, and J. Dong, "A knowledge-based approach to assembly sequence planning," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 11-12, pp. 1232–1244, 2007.

[17] Y. Y. Hsu, W. C. Chen, P. H. Tai, and Y. T. Tsai, "A knowledge-based engineering system for assembly sequence planning," in *Proceedings of the 36th International MATADOR Conference*, 36th International MATADOR Conference, pp. 123–126, Springer, London, UK, 2010.

[18] J. Qian, Z. Zhang, C. Shao, H. Gong, and D. Liu, "Assembly sequence planning method based on knowledge and ontostep," *Procedia CIRP*, vol. 97, no. 1, pp. 502–507, 2021.

[19] W.-C. Chen, P.-H. Tai, W.-J. Deng, and L.-F. Hsieh, "A three-stage integrated approach for assembly sequence planning using neural networks," *Expert Systems with Applications*, vol. 34, no. 3, pp. 1777–1786, 2008.

[20] J. F. Wang, J. H. Liu, and Y. F. Zhong, "A novel ant colony algorithm for assembly sequence planning," *The International Journal of Advanced Manufacturing Technology*, vol. 25, no. 11-12, pp. 1137–1143, 2005.

[21] Hao Pan, Wen Jun Hou, and Tie Meng Li, "Genetic algorithm for assembly sequences planning based on heuristic assembly knowledge," in *Applied Mechanics and Materials*, vol. 44–47, pp. 3657–3661, International Conference on Frontiers of Manufacturing and Design Science, Trans Tech Publications, Switzerland, 2011.

[22] I. Ibrahim, Z. Ibrahim, H. Ahmad et al., "An assembly sequence planning approach with a rule-based multi-state gravitational search algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 79, no. 5–8, pp. 1363–1376, 2015.

[23] C. Li and W. Hou, "Assembly sequence planning based on hierarchical model," *Wireless Communications and Mobile Computing*, vol. 2022, Article ID 9461794, 19 pages, 2022.

[24] G. Anil Kuma, M. V. A. Raju Bahubalendruni, I. Anil Kumar, S. S. Vara Prasad Varupala, and K. Sankaranarayanasamy, "A modified cut-set method for mechanical subassembly identification," *Assembly Automation*, vol. 41, no. 6, pp. 659–680, 2021.

[25] A. K. Gulivindala, M. V. A. R. Bahubalendruni, S. S. V. P. Varupala, and K. Sankaranarayanasamy, "A heuristic method with a novel stability concept to perform parallel assembly sequence planning by subassembly detection," *Assembly Automation*, vol. 40, no. 5, pp. 779–787, 2020.

[26] M. V. A. R. Bahubalendruni, A. Gulivindala, M. Kumar, B. B. Biswal, and L. N. Annepu, "A hybrid conjugated method for assembly sequence generation and explode view generation," *Assembly Automation*, vol. 39, no. 1, pp. 211–225, 2019.

[27] M. V. A. R. Bahubalendruni and B. B. Biswal, "An efficient stable subassembly identification method towards assembly sequence generation," *National Academy Science Letters*, vol. 41, no. 6, pp. 375–378, 2018.

[28] M. V. A. R. Bahubalendruni, B. B. Biswal, M. Kumar, and R. Nayak, "Influence of assembly predicate consideration on optimal assembly sequence generation," *Assembly Automation*, vol. 35, no. 4, pp. 309–316, 2015.

[29] J. Paulos, N. Eckenstein, T. Tosun et al., "Automated self-assembly of large maritime structures by a team of robotic boats," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 958–968, 2015.

[30] J. Seo, M. Yim, and V. Kumar, "Assembly planning for planar structures of a brick wall pattern with rectangular modular robots," in *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 1016–1021, Madison, New Jersey, USA, 2013.

[31] J. Seo, M. Yim, and V. Kumar, "Assembly sequence planning for constructing planar structures with rectangular modules," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5477–5482, Stockholm, Sweden, 2016.

[32] A. Z. Salvi, R. Simoni, and H. Simas, *Enumerating and assembling configurations with modular robots*, PhD thesis, Universidade Federal de Santa Catarina, Florianopolis, SC, Brazil, 2018.

[33] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, "A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 254–263, Cambridge, MA, USA, 2016.

[34] A. Z. Salvi, R. Simoni, and H. Simas, "Assembly sequence planning for shape heterogeneous modular robot systems," in *Multibody Mechatronic Systems*, J. Carvalho, D. Martins, R. Simoni, and H. Simas, Eds., vol. 54 of *MuSMe 2017. Mechanisms and Machine Science*, pp. 128–137, Springer, Cham, 2018.

[35] C. Jones and M. J. Mataric, "From local to global behavior in intelligent self-assembly," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 1, pp. 721–726, Taipei, Taiwan, 2003.

[36] R. Fitch, Z. Butler, and D. Rus, "Reconfiguration planning for heterogeneous self-reconfiguring robots," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453)*, vol. 3, pp. 2460–2467, New Jersey, USA, 2003.

[37] D. Bie, I. Sajid, J. Han, J. Zhao, and Y. Zhu, "Natural growth-inspired distributed self-reconfiguration of ubot robots," *Complexity*, vol. 2019, Article ID 2712015, 12 pages, 2019.

[38] T. Tucci, B. Piranda, and J. Bourgeois, "A distributed self-assembly planning algorithm for modular robots," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pp. 550–558, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2018.

[39] T. Kang, J.-B. Yi, D. Song, and S.-J. Yi, "High-speed autonomous robotic assembly using in-hand manipulation and re-grasping," *Applied Sciences*, vol. 11, no. 1, Article ID 37, 2020.

[40] B. Jenett, A. Abdel-Rahman, K. C. Cheung, and N. A. Gershenfeld, "Material–robot system for assembly of discrete cellular structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4019–4026, 2019.

[41] S. Leder, O. S. Oguz, H. Kim et al., "Co-design in architecture: a modular material-robot kinematic construction system," in *International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, 2021.

[42] J. Werfel, Y. Bar-Yam, D. Rus, and R. Nagpal, "Distributed construction by mobile robots with enhanced building blocks," in *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pp. 2787–2794, ICRA 2006, Orlando, FL, USA, 2006.

[43] J. Werfel and R. Nagpal, "Three-dimensional construction with mobile robots and modular blocks," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 463–479, 2008.

[44] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "Pseudo-bacterial potential field based path planner for autonomous mobile robot navigation," *International Journal of Advanced Robotic Systems*, vol. 12, no. 7, Article ID 81, 2015.

[45] U. Orozco-Rosas, K. Picos, J. J. Pantrigo, A. S. Montemayor, and A. Cuesta-Infante, "Mobile robot path planning using a qapf learning algorithm for known and unknown environments," *IEEE Access*, vol. 10, pp. 84648–84663, 2022.

[46] N. Abujabal, R. Fareh, S. Sinan, M. Baziyad, and M. Bettayeb, "A comprehensive review of the latest path planning developments for multi-robot formation systems," *Robotica*, vol. 41, no. 7, pp. 2079–2104, 2023.

[47] S. Lin, A. Liu, J. Wang, and X. Kong, "A review of path-planning approaches for multiple mobile robots," *Machines*, vol. 10, no. 9, Article ID 773, 2022.

[48] J. R. Sánchez-Ibáñez, C. J. Pérez-del-Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: a review," *Sensors*, vol. 21, no. 23, Article ID 7898, 2021.

[49] B. K. Patle, G. Babu L, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: on path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.

[50] Ryan Henderson Kelly, *Algorithms for planning and executing multi-roboat shapeshifting*, PhD thesis, Massachusetts Institute of Technology, 2019.

[51] B. Gheneti, S. Park, R. Kelly et al., "Trajectory planning for the shapeshifting of autonomous surface vessels," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 76–82, IEEE, 2019.

[52] B. Ding, Z.-X. Yang, X. Xiao, and G. Zhang, "Design of reconfigurable planar micro-positioning stages based on function modules," *IEEE Access*, vol. 7, pp. 15102–15112, 2019.

[53] Z. Wu and Q. Xu, "Survey on recent designs of compliant micro-/nano-positioning stages," *Actuators*, vol. 7, no. 1, Article ID 5, 2018.

[54] B. Ding, Z. Yang, and Y. Li, "Design of flexure-based modular architecture micro-positioning stage," *Microsystem Technologies*, vol. 26, no. 9, pp. 2893–2901, 2020.

[55] S. Liao, B. Ding, and Y. Li, "Design, assembly, and simulation of flexure-based modular micro-positioning stages," *Machines*, vol. 10, no. 6, 2022.

[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Computer Science, McGraw-Hill, New York, 2009, https://books.google.com.br/books?id=aefUBQAAQBAJ, 3rd edition.

[57] M. M. John Harris and J. L. Hirst, *Combinatorics and Graph Theory*, Undergraduate Texts in Mathematics (UTM), Springer-Verlag, New York, 2nd edition, 2008.