# Auto-Optimizing Connection Planning Method for Chain-Type Modular Self-Reconfiguration Robots

Haobo Luo, *Student Member, IEEE, and* Tin Lun Lam, *Senior Member, IEEE*

***Abstract*—Chain-type modular robots are capable of Self-Reconfiguration (SR), where the connection relationship between modules is changed according to the environment and tasks. This article focuses on the connection planning of SR based on Multiple In-degree Single Out-degree (MISO) modules. The goal is to calculate the optimal connection planning solution: the sequence with the fewest detachment and attachment actions. To this end, we propose an auto-optimizing connection planning method that contains a polynomial-time algorithm to calculate near-optimal solutions and an exponential-time algorithm to further optimize the solutions automatically when some CPUs are idle. The method combines rapidity and optimality in the face of an NP-complete problem by using configuration pointers, strings that uniquely specify the robot's configuration. Our polynomial-time algorithm, In-degree Matching (IM) uses the interchangeability of connection points to reduce reconfiguration steps. Our exponential-time algorithm, Tree-based Branch and Bound (TBB) further optimizes the solutions to the optimum by a new branching strategy and stage cost. In the experiments, we verify the feasibility of the auto-optimizing method combining IM and TBB, and demonstrate the superiority of IM over Greedy-CM in the SR of MISO modules and the near-optimality of IM compared to the optimal solutions of TBB.**

***Index Terms*—Modular robots, self-reconfiguration, connection planning, graph matching, computational complexity**

## I. Introduction

A Modular Self-Reconfiguration Robot (MSRR) consists of multiple homogeneous or heterogeneous modules and can transform its overall configuration by changing the connection relationship between modules. MSRR surpasses robots with a fixed configuration in terms of versatility and adaptability [1]. MSRR could perform various tasks such as exploration, grasping, and rescue in unstructured environments such as fires and earthquakes. The action process transforming one configuration of modular robots into a final configuration is defined as Self-Reconfiguration (SR). The significance of the SR process is to build a bridge between configurations suitable for different environments and tasks. SR is a critical competence that reflects the advantages of modular robots.

Existing MSRR hardware can be classified into lattice-type and chain-type [2]. Lattice-type MSRR includes Telecube [3], Crystalline [4], ICubes [5], ATRON [6], Catom [7], Stochastic-3D [8], Miche [9], Vacuubes [10], etc. Chain-type MSRR includes CONRO [11], PolyBot G3 [12], M-TRAN III [13], Molecule [14], SuperBot [15], CKBot [16], Odin [17], YamoR [18], Roombot [19], FreeSN [20], etc. The reconfiguration of lattice-type MSRR [21], [22] is achieved by each module moving along the surface of adjacent modules in the lattice space. The reconfiguration of chain-type MSRR [23], [24] generally aims to realize the detachment and attachment between modules by moving the manipulator composed of modules in a continuous space. Lattice-type MSRR is suitable for composing static structure or dynamic fluids [25], while chain-type MSRR is good at manipulation and locomotion [26]. Lattice-type and chain-type MSRR have different design principles and function advantages, making their reconfiguration algorithms fundamentally different. In this article, we focus on the reconfiguration of chain-type MSRR. Due to the diversity of MSRR hardware modules, we usually generalize an abstract model to increase the applicability of developed algorithms, such as Proteo [27], SlidingCube [28] and RollingSphere [29]. In this article, we study algorithms based on a new abstract module, the Multiple In-degree Single Out-degree (MISO) module, as illustrated in subsection III-A. It can represent the hardware modules of most chain-type modular robots such as FreeBOT [30], SnailBot [31], Ant3DBot [32], FireAnt-3D [33], Cross-ball [34] and Swarm-Bot [35]. Fig. 1 takes FreeBOT and SnailBot as examples to illustrate the physical meaning of MISO:

(1) Multiple interchangeable passive connection points;
(2) Single active connection point.

The self-reconfiguration of chain-type MSRR can be divided into two stages: connection planning and motion planning. The connection planning provides several sub-goals that specify which connections should be deleted and which connections should be created during the SR process. Motion planning

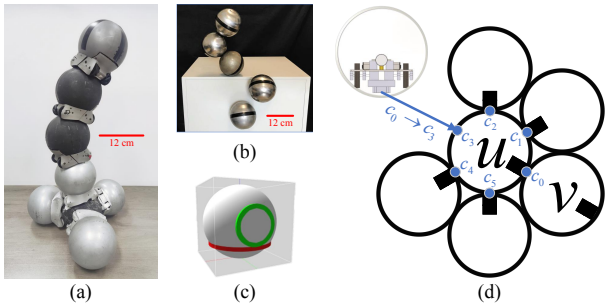



Fig. 1. Meaning of the MISO module. (a)(b) Two hardware implementations of the MISO module, SnailBot and FreeBOT. (c) 3D simulation of the MISO module. The green and red circles represent the module pose. (d) 2D schematic of the MISO module. The rectangle represents an active connection point, such as the magnet inside FreeBOT. The circle represents multiple interchangeable passive connection points, such as FreeBOT's iron shell.

This work was supported by the National Natural Science Foundation of China (62073274), and the funding AC01202101103 from the Shenzhen Institute of Artificial Intelligence and Robotics for Society. *(Corresponding author: Tin Lun Lam)*

H. Luo and T. Lam are with Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS), The Chinese University of Hong Kong, Shenzhen, Guangdong, China. E-mail: haoboluo@link.cuhk.edu.cn, tllam@cuhk.edu.cn.

controls the movement of the chain composed of modules to achieve these sub-goals. Therefore, whether sub-goals are optimal or non-redundant determines the speed of the entire SR process. This article focuses on the theoretical study of the connection planning problem, without considering constraints in motion planning. Optimal connection planning, also called Optimal Reconfiguration Planning (ORP) in [36], is of great significance to the self-reconfiguration of chain-type MSRR. Unfortunately, ORP has been proved to be NP-complete based on SuperBot [24], which means that an algorithm calculating the optimal solution in polynomial time is unlikely to exist. Therefore, the current works' two main research directions are calculating the near-optimal solution in polynomial time and calculating the optimal solution in exponential time, denoted as polynomial-time algorithms and exponential-time algorithms.

Existing works have not yet tried to combine the advantages of polynomial-time algorithms and exponential-time algorithms in one method. Nelson [37] proposed a numerical optimization method for optimal solutions based on permutation matrices, but the computational complexity of the solver [38] is $\mathbb{O}(n!)$ which is worse than the exponential computational complexity. Nelson [37] further proposed a polynomial-time algorithm to solve the near-optimal solution based on principal components analysis and bipartite matching. The algorithm does not exploit graph properties such as interchangeable connection points. Additionally, it cannot handle configurations with root modules whose adjacency matrices are nilpotent [39]. Hou and Shen [36] proposed an exponential-time algorithm and a polynomial-time algorithm, called MDCOP and Greedy-CM, respectively. MDCOP converts ORP into Distributed Constraint Optimization Problem (DCOP) to call the existing algorithms in the DCOP area rather than to develop a new optimal solver for the ORP area from scratch. Greedy-CM can compute the solutions with near-optimality, which is verified by experimental comparison with the optimal solutions calculated by MDCOP. Greedy-CM relies on SuperBot's four non-interchangeable connection points to match sub-graphs greedily.

This article introduces an auto-optimizing connection planning method, which combines the rapidity of polynomial-time algorithms and the optimality of exponential-time algorithms. This method maintains a library where the near-optimal solutions can be read and further optimized by the exponential-time algorithm automatically when some CPUs are idle. The configuration pointer can be used to read the solution after we prove its isomorphic invariance by Theorem 2 in Subsection IV-A. The auto-optimizing method converts computational complexity into space complexity so that MSRR can quickly obtain a further-optimized or optimal solution in actual use. Second, we propose a polynomial-time algorithm, In-degree Matching (IM), which demonstrates that the interchangeability of connection points can be used to reduce reconfiguration steps. Third, we propose an exponential-time algorithm, Tree-based Branch and Bound (TBB), which develops an optimal solver in the ORP area from scratch with a new branching strategy and stage cost. The advantages of TBB are that near-optimal solutions can be used to prune unpromising branches, and further-optimized solutions can be output from time to time before an optimal solution is guaranteed. The contributions of this article are summarized as:

(1) The auto-optimizing method combines the rapidity of polynomial-time algorithms and the optimality of exponential-time algorithms by the reading of configuration pointers.
(2) The IM algorithm improves the optimality of polynomial-time algorithms by interchanging connection points.
(3) The TBB algorithm further optimizes the solution to the optimum by a new branching strategy and stage cost.
(4) The code for all algorithms in this article is shared on Github [1].

The content of this article is organized as follows. Section II provides related works. Section III introduces the MISO module and the definition of connection planning. Section IV describes the contributions of this article: the auto-optimizing method, the IM algorithm, and the TBB algorithm. In Section V, well-designed experiments are used to verify the feasibility of the auto-optimizing method and the superiority of the IM algorithm. In Section VI, case studies are provided to help understand the application of the proposed method.

## II. Related Works

The existing self-reconfiguration algorithms are introduced in the following four categories [1]: bio-inspired approaches, agent-based approaches, control-based approaches and search-based approaches. **Bio-inspired** approaches draw on the morphological change control methods of multicellular organisms, such as the gene regulatory network [40], the distributed control protocol for hormone-like messages [11], etc. Bio-inspired approaches do not pursue the optimal number of self-reconfiguration steps. **Agent-based** approaches train multiple agents who can observe their local environment and act independently. The types of agents include Cellular Automata [41], reinforcement learning [42], game theory [43], etc. For example, Shiba *et al.* [44] apply Q-learning to transform the overall shape of MSRR between line, ring and others. Fitch and Butler [28] adopt the idea of the Markov decision process to distributedly control a large-scale MSRR to achieve locomotion and obstacle-crossing by changing the shape. These agent-based approaches usually output local optimal policies with a certain degree of randomness. **Control-based** approaches simulate the feedback control loop to navigate the sequence of configurations converging to a final configuration. In the gradient-based [45] method, non-source modules follow the steepest descent computed by the density of neighbors' attractors, which are broadcast by a source module. Furthermore, a distance-based method, a heat-based method and a hybrid method are developed in [27]. The distance-based control method is easily trapped by local minimums, such as a long-chain configuration. The heat-based method is subject to the slow broadcast speed of temperature, but it can jump out of some local minimums. The hybrid method combines the advantages of the above two methods. None of these three methods can guarantee convergence. In general, control-based methods are most suitable for lattice-type MSRR whose

[1] https://github.com/FreeformRobotics/MISO_connection

modules can move independently. For example, an obstacle-crossing strategy is proposed in [29] to control multiple modules in parallel to flow on the surface of obstacles by splicing SR processes in the lattice space.

The algorithms proposed in this article can be roughly classified into the search-based approach. **Search-based** approaches generally perform deterministic or heuristic searches in the configuration space. One of the popular heuristic searching methods is the A* search, which estimates the path distances between initial, current, and final configuration based on some metrics, such as the minimal number of steps, to search for the shortest path. For example, Asadpour *et al.* [46] propose the graph signature and the graph edit distance metric. The graph signature is an isomorphic invariant code that can be used to avoid repeated searches when searching for an SR path. The graph edit distance metric can be used as a heuristic function to evaluate the distance between the current configuration and the final configuration. Further, Asadpour *et al.* [47] add the graph edit distance metric from the initial configuration to the current configuration in the heuristic function. Search algorithms may fall into inadequate local optimal solutions and are subject to the exponential growth of the configuration space. Eden [48] gives the upper and lower bounds of the number of exponential reproduction of $n$ biological cells on the 2D lattice as $2^{n-3} < S(n) < \frac{1}{n}2^{2n-1}$. The configuration space of chain-type MSRR also increases exponentially with the total number of modules. When the configuration space is large enough, finding a tractable search direction is difficult and computationally burdensome. For another example, genetic algorithms can be used to search for the task-oriented optimal configuration [49] and self-reconfiguration steps. The genetic algorithm can jump out of the local minimum and gradually approach the global optimal solution as long as time permits. However, the range of feasible descendants of the genetic algorithm increases exponentially with the total number of modules.

We further analyze some self-reconfiguration algorithms that can be divided into connection planning and motion planning stages. Motion planning algorithms are beyond the scope of this article, which involves completely different theorems and concepts.

Hou and Shen [36] propose two connection planning algorithms for SuperBot: MDCOP and Greedy-CM as presented in Section I. The calculations of these two algorithms rely on the four different connection points of SuperBot. Liu and Yim [50] propose a distributed configuration recognition algorithm by solving graph isomorphism problems through maximum subgraph matching and dynamic programming. Further, Liu *et al.* [26] propose a self-reconfiguration algorithm for SMORES-EP based on iterative configuration decomposition of rooted trees. The algorithm replaces maximum common sub-configuration with virtual modules and repeats the process for the remaining sub-configurations. These two algorithms are customized for SMORES-EP. Yim *et al.* [51] propose a graph-based connection planning algorithm for PolyBot. The algorithm uses a straight-line configuration as an intermediate configuration. Similar are LineUp [52] and MorphLine [53]. These algorithms serve as the upper bound of the least reconfiguration steps [53]. There are also some general algorithms based on abstract modules. Casal and Yim [23] consider general chain-type modular robots and propose a divide-and-conquer connection planning algorithm, but they do not seek the optimal solution. Hou and Shen [53] propose the concept of Connection Number (CN) and propose the corresponding reconfiguration algorithm. This algorithm contains redundant actions and is unsuitable for configurations that have circuits or do not maintain overall connectivity.

There are some related topics, including NP-completeness and configuration recognition. Hou and Shen [24] prove that the ORP is NP-complete based on SuperBot and provide the lower bound of the least reconfiguration steps. Gorbenko and Popov [54] provide another way to prove the NP-completeness of ORP. The configuration recognition task means matching a new configuration to a library of known configurations. Park *et al.* [55] provide two general configuration recognition methods based on graph isomorphism algorithm $nauty$ and spectral decomposition respectively, as well as a customized configuration recognition method for CKBot. Most existing configuration recognition methods require isomorphism detection between the new configuration and each known configuration in the library without skipping. Thus, there is room for improvement in the computational complexity of existing configuration recognition methods.

TABLE I
MEANING OF THE MAIN SYMBOLS

| Symbol | Meaning |
|---|---|
| $n$ | Total number of modules |
| $X^I$ | Adjacency matrix of the initial configuration |
| $X^F$ | Adjacency matrix of the final configuration |
| $M_u^I$ | Module in the initial configuration whose ID is $u$ |
| $M_v^F$ | Module in the final configuration whose ID is $v$ |
| $u$, $v$, $p$, $q$ | Different module IDs |
| $c_0@M_u^I$ | Active connection point of $M_u^I$ |
| $c_i@M_u^I$ | The $i$-th passive connection point of $M_u^I$ |
| $c_j@M_v^F$ | The $j$-th passive connection point of $M_v^F$ |
| $i$,$j$, $k$, $l$ | Different indexes of passive connection point |
| $Mc_i@M_u^I$ | Module connected to $c_i@M_u^I$ |
| $Nc_i@M_u^I$ | Number of modules in the sub-tree at $c_i@M_u^I$ |
| $Pc_i@M_u^I$ | Configuration pointer of the sub-tree at $c_i@M_u^I$ |
| $y$.cost | A variable of the node $y$ in the TBB algorithm |
| $d$ | Maximum in-degree of the MISO module |
| $P^{M^I_{\text{root}}}_{M^F_{[:]}}$ | A root module is matched with each vacancy |
| $P^{Mc_{[:]}@M_u^I}_{Mc_{[:]}@M_v^F}$ | Match one by one in different permutations |

## III. PROBLEM FORMATION

This section explains the method to abstract non-lattice modular robots as MISO modules, the definition of connection planning, and the expression of graphs. Table I summarizes the meaning of the main symbols used in this article.

### A. Abstract as MISO Modules

The hardware modules in different chain-type modular robots have different connection characteristics and drive mechanisms, such as FreeBOT [30], Cross-Ball [34], FireAnt-3D [33] and so on. This article proposes to abstract these

hardware modules as Multiple In-degree Single Out-degree (MISO) modules.

Take FreeBOT as an example. FreeBOT consists of a rough metal spherical shell and an internal trolley with two differential wheels and a permanent magnet, as shown in Fig. 1(b)(d). The permanent magnet can be actively connected to the spherical shell of another module, representing a single active connection point. The spherical shell can be connected by multiple other modules, representing multiple interchangeable passive connection points. In graph theory, the number of passive connection points and the number of active connection points of the module are denoted as the in-degree and out-degree of the module. Other hardware modules can be abstracted as MISO modules by designating a unique connection point as an active connection point and other connection points as passive connection points, such as Roombot, Molecube and ClicBot.

The abstracted MISO module has one active connection point and multiple interchangeable passive connection points, denoted as $c_0@M_u$ and $c_1@M_u, c_2@M_u, \cdots, c_5@M_u$ as shown in Fig. 1(d) where $c_i@M_u$ is abbreviated as $c_i$. Abstracting as MISO modules can induce a hardware-forced or artificial-forced direction to each edge in the connectivity graph of the configuration and make it easier to express the configuration as a tree. Second, the interchangeability of the passive connection points of MISO modules can reduce redundant steps for satisfying connection type restrictions. However, if the hardware modules do not support interchangeability of the connection points, the optimal solution calculated based on MISO modules may not meet the hardware implementation requirements.

### B. Connection Planning

In this article, we define *configuration* as the connection relationship between modules. This definition of configuration does not involve the relative poses between modules, since motion planning is not considered in this article. We further assume that the underlying chain-type MSRR can be unfolded by the Carpenter's Rule Theorem [56] and is singularity-free [57] between any two modules. In other words, motion planning is assumed to be able to accomplish all detachment or attachment actions required by the result of connection planning.

Configuration can be expressed by adjacency matrices as shown in Equation (1). In the adjacency matrix, the element $x_{uv} = 1$ means that the active connection point of the module $M_u$ is connected to any passive connection point of the module $M_v$, denoted by $M_u \rightarrow M_v$ in Equation (1). An SR instance is represented by the adjacency matrix of the initial configuration $X^I$ and the adjacency matrix of the final configuration $X^F$. The element-wise subtraction between $X^F$ and $X^I$ generates a difference matrix $D$ [37], as shown in Equation (2). In the $D$ matrix as shown in Equation (3), $d_{uv} = 1$ means that the active connection point of the module $M_u$ in the initial configuration is to be connected to one passive connection point of the module $M_v$. $d_{uv} = -1$ means that the connection between the module $M_u$ and the module $M_v$ in the initial configuration is to be broken. Thus, one non-zero element in the $D$ matrix is defined as one reconfiguration step. The number of non-zero elements in the $D$ matrix is also regarded as the cost of the solution output by connection planning algorithms.

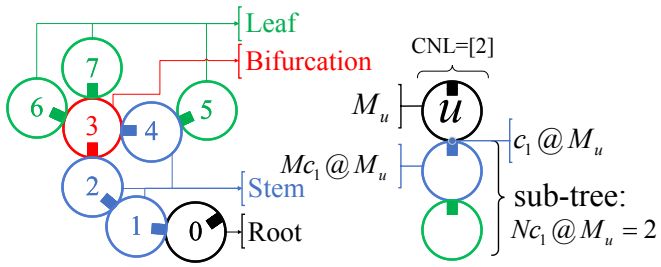



Fig. 2. A graph $\mathbb{G}$ may contain multiple Connected Components (CCs). Each CC may contain four types of modules.

$$X : x_{uv} = \begin{cases} 1 & M_u \rightarrow M_v \\ 0 & \text{otherwise} \end{cases} \quad u, v = 1, \cdots, n \tag{1}$$

$$D = X^F - X^I \tag{2}$$

$$D : d_{uv} = \begin{cases} -1 & \text{detach } M_u \nRightarrow M_v \\ 1 & \text{attach } M_u \Rightarrow M_v \\ 0 & \text{otherwise} \end{cases} \quad u, v = 1, \cdots, n \tag{3}$$

Different adjacency matrices can represent the same configuration according to different ID assignments. Let's assume that each module in the initial configuration and the final configuration has been assigned a unique ID [58] to generate $X^I$ and $X^F$ for the input of connection planning algorithms. The problem of optimal connection planning can be defined as selecting an appropriate ID assignment for modules in the final configuration, so that the $D$ matrix calculated by Equation (2) contains the fewest non-zero elements. The computational complexity of traversing all possible ID assignments of the final configuration is $\mathbb{O}(n!)$. In order to reduce the computational complexity, numerical optimization or graph matching methods are generally used in the connection planning algorithm to adjust the ID assignment in the final configuration. The above connection planning model was first introduced by Nelson [37].

### C. Notations and Expressions of Graphs

We can draw the input adjacency matrix as a simple directed graph $\mathbb{G} = (V, E)$. Each graph $\mathbb{G}$ may contain more than one Connected Component (CC), where vertices are linked to each other by paths, such as the two CCs in Fig. 2. There may be four types of modules in each CC. They are defined as:

(1) Out-degree $= 0 \rightarrow$ *root* module;
(2) Out-degree $= 1$ and In-degree $= 0 \rightarrow$ *leaf* module;
(3) Out-degree $= 1$ and In-degree $= 1 \rightarrow$ *stem* module;
(4) Out-degree $= 1$ and In-degree $\geq 2 \rightarrow$ *bifurcation* module.

The positive direction of a module is defined as the direction of its active connection point, such as the N pole of the

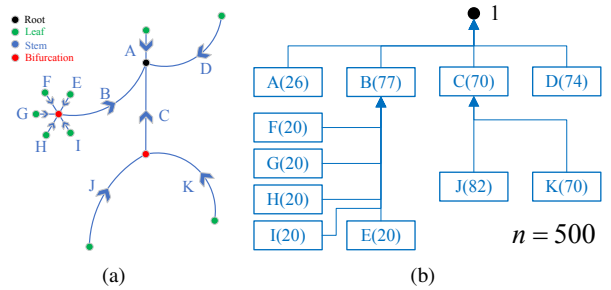



Fig. 3. A Connected Component (CC) that contains a unique root module is itself a rooted tree. (a) The graph of the CC. (b) The tree structure diagram of the CC. The numbers in parentheses represent the number of modules included in the chain.

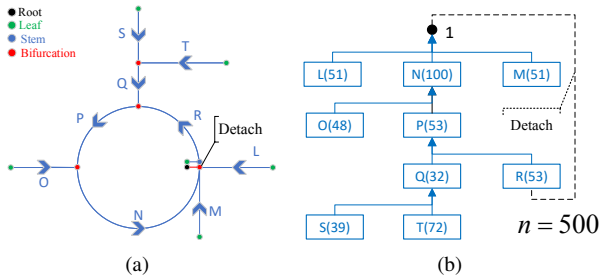



Fig. 4. A Connected Component (CC) without a root module can be spanned into a rooted tree by one detachment. (a) The graph of the CC. (b) The tree structure diagram by spanning the CC. The numbers in parentheses represent the number of modules included in the chain.

permanent magnet in FreeBOT. A chain is defined as one leaf or bifurcation module and the possible stem modules in front of it. For example, $M_{[6]}$,$M_{[5,4]}$ and $M_{[3,2,1]}$ in Fig. 2 are three different chains.

An example explaining the meaning of some symbols is introduced in the right of Fig. 2. In the example, the root module with an ID $u$ is denoted as $M_u$. $M_u$ has a passive connection point, $c_1@M_u$. The module with unknown ID connected to the passive connection point $c_1@M_u$ is denoted as $Mc_1@M_u$. $Mc_1@M_u$ is also the root module of a sub-tree connected to $c_1@M_u$, and the number of modules contained in this sub-tree is denoted as $Nc_1@M_u$. $Nc_i@M_u$ is called the Connection Number (CN) [53], the number of modules in the sub-tree whose root module is connected to the passive connection point $c_i@M_u$. The Connection Number List (CNL) of a module $M_u$ is defined as $[Nc_1@M_u, \cdots, Nc_i@M_u, \cdots, Nc_d@M_u]$, where $d$ represents the maximum number of coexisting passive connection points of the module $M_u$.

Figures 3 and 4 introduce two CCs for further examples. To simplify the expression, Fig. 3(a) and Fig. 4(a) use a line segment to represent the chain, and each chain is distinguished by a letter. The positive direction of the chain is indicated by an arrow. Different types of modules are highlighted in different colors. Figure 3(b) and Fig. 4(b) shows the total number of modules included in each chain. For example, $A(26)$ in Fig. 3(b) represents that the chain $A$ in Fig. 3(a) contains a total of 26 modules, including 25 stem modules and 1 leaf modules.

The CC in Fig. 3(a) has one unique root module and is a tree itself. Starting from the unique root module, we can organize

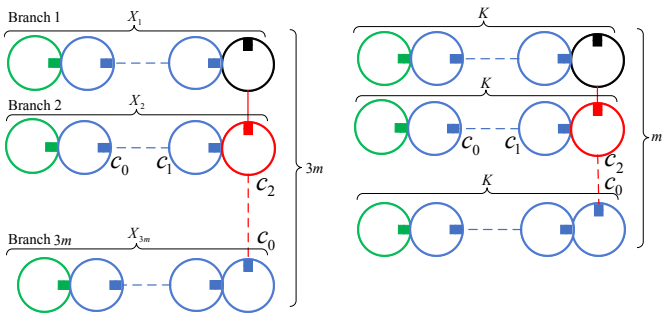



Fig. 5. An instance of the optimal reconfiguration problem constructed for reducing a 3-PARTITION problem to an ORP.

TABLE II
NUMBER OF CONNECTIONS OF EACH TYPE IN TWO CONFIGURATIONS

| | $c_0 \to c_1$ | $c_0 \to c_2$ |
|---|---|---|
| Initial configuration | $n-3m$ | $3m-1$ |
| Final configuration | $mK-m$ | $m-1$ |
| Differences | $2m$ | $-2m$ |

the chains hierarchically in a tree structure as shown in Fig. 3(b). The root module is the first level of the tree structure. The root module may be connected by multiple chains like bifurcation modules. The chains after each bifurcation module are located at the next level of the hierarchy diagram. And so on, until the emergence of leaf modules.

The CC in Fig. 4(a) does not have a root module, but it has one circuit composed of 3 chains $P$-$R$-$N$. This type of CC can be spanned as a tree by disconnecting at any module in the circuit. For example, in Fig. 4(a), we choose to disconnect the active connection point of the red module in the circuit to make it a root module, while the blue module originally connected by the red module becomes a leaf module.

## IV. Connection Planning Method

This section introduces the connection planning method and its main components.

### A. Method Flowchart

Figure 6 shows the flowchart of the connection planning method, which is motivated by Theorem 1 below.

**Theorem 1.** *The Optimal Reconfiguration Problem (ORP) of MSRR composed of MISO modules is also NP-complete, but the interchangeability of passive connection points of MISO modules reduces the lower bound of the least reconfiguration steps.*

*Proof.* The ORP based on modules with non-interchangeable connection points is proven to be NP-complete in [36]. In the following, we show that the ORP based on MISO modules with interchangeable passive connection points is also NP-complete, but the lower bound of the least reconfiguration steps is reduced from $6m-2$ [36] to $4m$.

The NP-completeness of the ORP can be proved by the equivalent proposition that the known NP-complete problem, 3-PARTITION, is polynomial reducible to the acyclic

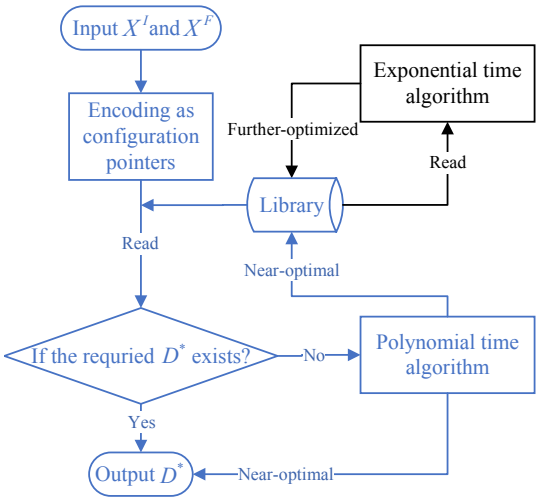



Fig. 6. Method flowchart. The auto-optimizing connection planning method combines the advantages of polynomial-time algorithms and exponential-time algorithms.

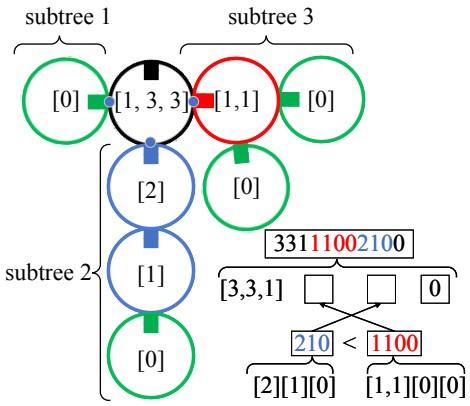



Fig. 7. The configuration pointer uniquely encodes a configuration in the configuration space.

ORP. We construct a SR instance based on MISO modules in Fig. 5 to represent any given 3-PARTITION problem $S = X_1, \cdots, X_{3m}$ as an acyclic ORP. In Fig. 5, the initial configuration contains $3m$ branches, and each branch $i$ has $X_i$ modules connected by the edge $c_0 \rightarrow c_1$. The final configuration contains $m$ equal-length branches, and each branch has $K = \frac{n}{m}$ modules. The rightmost module of each branch in both configurations is connected by $c_0 \rightarrow c_2$.

Table II summarizes the number of connections of each type in the two configurations of the constructed SR instance. Thus the lower bound of the least reconfiguration steps of the constructed SR instance is calculated to be $4m = 2m + \parallel -2m \parallel$. Interchanging the two columns $c_0 \rightarrow c_1$ and $c_0 \rightarrow c_2$ in Table II does not change the lower bound $4m$. By substituting the SR instance and lower bound into the proof in [36], it can verify that the interchangeability of passive connection points of MISO modules does not affect the NP-completeness. □

The NP-completeness of ORP in Theorem 1 indicates that the optimal solution of connection planning is probably not solvable in polynomial time. Therefore, algorithms for solving NP-complete problems can be divided into polynomial-time algorithms for finding near-optimal solutions and exponential-time algorithms for finding optimal solutions. The method flowchart in Fig. 6 uses a polynomial-time algorithm and an exponential-time algorithm in different CPU states to solve the same connection planning problem of frequently used SR instances. This provides a solution quickly, while allowing further optimization and detection when the solution is not yet globally optimal.

The method flowchart is based on two concepts, functional isomorphic configuration [55] and isomorphic invariant signature [47]. Functional isomorphic configurations do not have the same connection relationship but can complete the same task or have similar properties to adapt to the environment. These functional isomorphic configurations can be replaced by a typical configuration that is frequently used. As the number of uses of MSRR increases, so does the number of self-reconfiguration between typical configurations. Further optimizing the connection planning results of frequently used SR instances can statistically increase the self-reconfiguration speed of MSRR in actual use. This further optimization can be done by exponential-time algorithms when MSRR has idle computing resources, but an ability to store reachable configurations and their cost is required. The second paragraph below will introduce an isomorphic invariant signature of the configuration composed of MISO modules, named *configuration pointer*, which is a special case of the configuration string from [53] for MISO modules as shown in Fig. 7. The configuration pointers for isomorphic configurations do not change as modules exchange IDs as the adjacency matrix does. Therefore, the solution of any SR instance can be saved and read by the configuration pointers of the two configurations.

The flowchart of the connection planning method mainly contains a polynomial-time algorithm, an exponential-time algorithm, and a library. The library is a piece of computer memory that records three compressed matrices of each frequently used SR instance: $X^I$, $X^F$ and the optimal solution $D^*$. As shown in Fig. 6, the method first encodes the input $X^I$ and $X^F$ into two configuration pointers. If a $D^*$ matrix can be read from the library based on these two configuration pointers, the $D^*$ is output directly. If no $D^*$ matrix can be read from the library, the IM algorithm in Subsection IV-B is called to solve the near-optimal solution in polynomial time. When MSRR has idle computing resources, the connection planning results in the library will be read using configuration pointers and further optimized by an exponential-time algorithm, TBB in Subsection IV-C. TBB uses the cost of the near-optimal solution as an upper bound and outputs further-optimized solutions to update the library until an optimal solution is guaranteed.

Reading in the library requires the configuration pointer to be isomorphic invariant. The isomorphic invariance of configuration pointers can be guaranteed by the following three encoding rules. Rule (1): stack the CNLs of the modules on each chain in the positive direction of the chain. For example, in Fig. 7, the configuration pointer generated by the subtree 2 is $210 \leftarrow [2][1][0]$. Rule (2): when the CNL of the bifurcation module contains equal CN values, these CN values

are sorted by their corresponding configuration pointers. For example, in Fig. 7, the CNL of the black module contains two equal CN values $Nc_2@M_u = Nc_3@M_u = 3$, and their corresponding configuration pointers generated from the sub-tree 2 and the sub-tree 3 are 210 and 1100. Because of $1100 > 210$, 1100 is written in front of 210 in the final configuration pointer. Rule (3): the CNL of the bifurcation module is sorted from largest to smallest, and the configuration pointers corresponding to CNs are stacked in the CNs' order. For example, in Fig. 7, the sorted CNL of the black module is $[3, 3, 1]$. Thus the configuration pointer generated by the sub-tree 1 corresponding to $Nc_1@M_u = 1$ is stacked at the end of the final configuration pointer. Based on these rules, the generated configuration pointer is isomorphic invariant as explained in Theorem 2 below, which is proved in the appendix.

**Theorem 2.** *Configuration pointers are the unique digital signatures of non-isomorphic configurations in the configuration space, and their length growth rate is* $\mathbb{O}(n \log_{10} n)$.

The benefit of configuration pointers is to read solutions of SR instances from the library quickly. The reading task is to match a newly encountered SR instance to a known SR instance in the library [55]. In our library, the $D^*$ of each SR instance is saved in the memory table as $Library[p(X^I)][p(X^F)]$, where the row index (or the column index) $p(X^I)$ (or $p(X^F)$) represents the configuration pointer generated from $X^I$ (or $X^F$). When a newly encountered SR instance represented by $X^I_{new}$ and $X^F_{new}$ needs to be read from the library, $p(X^I_{new})$ (or $p(X^F_{new})$) is compared with all row indexes (or all column indexes) of the library bit by bit. For example, if $p(X^I_{new}) = 331100210$ is compared with $p(X^I) = 33111002100$, their 5th digits from the left are different. Thus $X^I_{new}$ and $X^I$ are determined to be different configurations, and the comparison with the next row index starts immediately. The computational complexity of the comparison with all row indexes (or all column indexes) is $\mathbb{O}(n \log_{10} n)$, according to Theorem 2.

**Algorithm 1** In-degree Matching (IM) algorithm

**Input:** $X_I, X_F$ matrices
**if** Any configuration contains multiple CCs **then**
  Add a virtual root module to be connected by CCs
**end if**
Initialize $D = X_F - X_I$
**for** Each (virtual) root/bifurcation module $M^I_u$ **do**
  **for** Each (virtual) root/bifurcation vacancy $M^F_v$ **do**
    Initialize $D'$ to zero matrix
    **if** $M^I_u$ ($M^F_v$) is not the root module (vacancy) **then**
      Add two pre-reconfiguration steps to $D'$
    **end if**
    **while** Number of matched modules $< n$ **do**
      Call the PAIRING($\langle M^I_u, M^F_v \rangle$) function
      Package the $u$-$v$ rooted CSG
    **end while**
    Update $D = D'$ if $\| D' \|_F < \| D \|_F$
  **end for**
**end for**
**Output:** near-optimal $D$
**function** PAIRING($\langle M^I_u, M^F_v \rangle$)
  **if** $M^I_u$ or $M^F_v$ is a leaf module **then**
    **return**
  **end if**
  Initialize a bipartite graph
  **for** each child module $Mc_i@M^I_u$ **do**
    **for** each child vacancy $Mc_j@M^F_v$ **do**
      Add two vertices: $Mc_i@M^I_u$ and $Mc_j@M^F_v$
      Add an edge with the weight: $\| Pc_i@M^I_u - Pc_j@M^F_v \|$
    **end for**
  **end for**
  Minimum weight full matching the bipartite graph
  **for** each matching result ($Mc_k@M^I_u$, $Mc_l@M^F_v$) **do**
    Add a child node, $\langle Mc_k@M^I_u, Mc_l@M^F_v \rangle$
    PAIRING($\langle Mc_k@M^I_u, Mc_l@M^F_v \rangle$)
  **end for**
**end function**

### *B. In-Degree Matching Algorithm*

In this subsection, we introduce a connection planning algorithm that can solve near-optimal solutions in polynomial time, the In-degree Matching (IM) algorithm, as summarized in Alg. 1. The IM algorithm assumes that the applied hardware modules have been abstracted as MISO modules which have the property as shown in Theorem 3 below.

**Theorem 3.** *A connected component composed of single out-degree modules contains exactly one root module or one circuit, but not both.*

*Proof.* Let us assume that there are $x$ root modules in a CC that meets two conditions: (1) the entire CC with $n \geq 2$ modules maintains connectivity; (2) each module has a single active connection point.

Because of condition (1), at least one passive connection point of any root module must be connected by one module. Simple directed graphs can be drawn starting from root modules. There must be a common module of any two graphs to maintain the connectivity of the entire CC. The common module belongs to two different chains of two graphs simultaneously. Thus the common module should have two different output directions, which requires two active connection points in a module. Contradiction with the condition (2) exists, thus $x \leq 1$.

If there are $y$ circuits in the initial graph of CC, any circuit can be converted to a line by disconnecting the active connection point of any module $M_u$ in the circuit. Thus, the module $M_u$ will become a root module. All chains outside circuits remain connected to the converted lines, which means the new graph generated by the disconnections of all circuits still satisfies the condition (1). The number of generated root modules in the new graph equals the number of circuits in the initial graph. Because of $x \leq 1$, there is only $y \leq 1$ circuits in the initial graph.

When a CC contains $x = 0$ root modules and $y = 1$

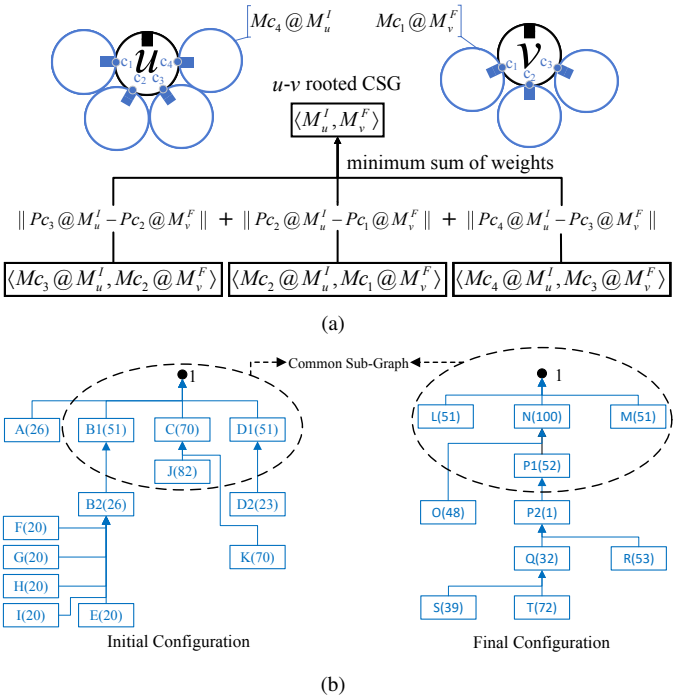



Fig. 8. The In-degree Matching algorithm. (a) The Pairing function performs bipartite matching with the sub-tree dissimilarity as the weight and generates the child nodes with the minimum sum of weights. (b) The Common Sub-Graph (CSG) can be calculated by iteratively calling the Pairing function.

circuits, it can be converted to a CC containing $x = 1$ root modules and $y = 0$ circuits through one detachment. The mutual conversion between the case $(x = 0, y = 1)$ and the case $(x = 1, y = 0)$ through one detachment or attachment is complete and exclusive. Thus the other two cases, $(x = 0, y = 0)$ and $(x = 1, y = 1)$, are impossible. □

Theorem 3 shows that the CCs in the configuration composed of MISO modules are either trees themselves or can be converted as trees through one detachment, such as the CC in Fig. 4. In the IM algorithm, the CCs in the configuration are first converted to rooted trees and connected to a virtual root module. Thus the entire configuration can be expressed as a rooted tree composed of $n + 1$ modules (Alg. 1 Line 2-4).

The core of the IM algorithm is to calculate $u$-$v$ rooted Common Sub-Graph (CSG) whose starting node is $\langle M_u^I, M_v^F \rangle$ representing that the module $M_u^I$ and the vacancy $M_v^F$ are matched. For the sake of distinction, we call the member of the initial configuration as module and the member of the final configuration as vacancy. The selection range of $M_u^I$ ($M_v^F$) is the root and bifurcation modules (vacancies) of the configuration (Alg. 1 Line 6-7). Different $u$-$v$ combinations will produce different $D'$ matrices, and the best one of them is kept (Alg. 1 Line 16). The best one may be obtained by computing the $u$-$v$ rooted CSG starting from a node that matches a bifurcation module and a bifurcation vacancy. When $M_u^I$ ($M_v^F$) is not the root module (vacancy) in the configuration, the configuration is first pre-reconfigured to make $M_u^I$ ($M_v^F$) the root module (vacancy). Pre-reconfiguration is completed by an attachment action and a detachment action (Alg. 1 Line 9-11). Take the two configurations shown in Fig. 9 as an example.

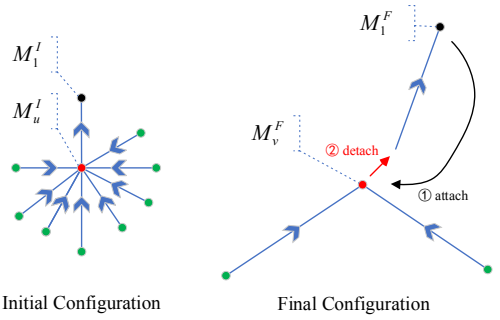



Fig. 9. Pre-reconfiguration. When $M_u^I$ (or $M_v^F$) is not the root module (or vacancy), pre-reconfigure the configuration to make $M_u^I$ (or $M_v^F$) become the root module (or vacancy).

$M_v^F$ is not the root vacancy. The pre-reconfiguration actions of the final configuration are attaching the original root vacancy $M_1^F$ to a passive connection point of $M_v^F$, and detaching the active connection point of $M_v^F$. However, if neither $M_u^I$ nor $M_v^F$ have redundant passive connection points, the pre-reconfiguration will be canceled.

The calculation of $u$-$v$ rooted CSG is done by recursively calling the Pairing function (Alg. 1 Line 13). Take the two configurations shown in Fig. 8(b) as an example. The starting node in $u$-$v$ rooted CSG is $\langle M_u^I, M_v^F \rangle$, as shown in Fig. 8(a). We represent the four child modules of $M_u^I$ as $Mc_1@M_u^I$, $Mc_2@M_u^I$, etc. The same representation method is followed by the three child vacancies of $M_v^F$ in Fig. 8(a). The Pairing($\langle M_u^I, M_v^F \rangle$) function creates a bipartite graph with all child modules of $M_u^I$ on one side and all child vacancies of $M_v^F$ on the other side (Alg. 1 Line 24-27). In the bipartite graph, any two vertices from different sides, such as $Mc_i@M_u^I$ and $Mc_j@M_v^F$, are connected by an edge whose weight is the sub-tree dissimilarity $\| Pc_i@M_u^I - Pc_j@M_v^F \|$ (Alg. 8 Line 28). The sub-tree dissimilarity is defined as the absolute difference between two configuration pointers to measure the difference between two sub-trees connected at $c_i@M_u^I$ and $c_j@M_v^F$. The closer the dissimilarity of two subtrees is to zero, the more similar two subtrees are. A dissimilarity of zero means the subtrees are identical. The sub-tree dissimilarity can be normalized in the code implementation using bit-packed configuration pointers. Based on the above weights, the minimum weight full bipartite matching algorithm [59] can calculate the optimal matches in the bipartite graph (Alg. 1 Line 31). For example, in Fig. 8(a), a $4 \times 3$ bipartite matching problem is solved to obtain the minimum sum of weights, $\| Pc_3@M_u^I - Pc_2@M_v^F \| + \| Pc_2@M_u^I - Pc_1@M_v^F \| + \| Pc_4@M_u^I - Pc_3@M_v^F \|$, which is used to create three corresponding child nodes (Alg. 1 Line 32-33). These three child nodes are recursively fed into the Pairing function to match more modules (Alg. 1 Line 34).

The final $u$-$v$ rooted CSG is shown in the dashed box in Fig. 8(b), where the chains B, D and P are divided into two parts to identify $u$-$v$ rooted CSG. The module and vacancy matched in each node $\langle M_u^I, M_v^F \rangle$ do not need to change the connection relationship. Next, we package all the matched modules (vacancies) into a virtual root module (vacancy)

which is to be connected by the remaining branches (Alg. 1 Line 14). The purpose of packaging is to transform the configuration into a rooted tree with a much smaller number of modules or vacancies. Thus, the CSG with a starting node matching the virtual root module and the virtual root vacancy will be solved repeatedly based on new rooted trees until all modules are matched.

It can be verified that the maximum number of bifurcation modules in a configuration is $\lceil \frac{n}{2} \rceil$. The computational complexity caused by matching different $u$ and $v$ is $\mathbb{O}(\lceil \frac{n}{2} \rceil \times \lceil \frac{n}{2} \rceil)$. The computational complexity of calculating $u$-$v$ rooted CSG is $\mathbb{O}(n)$, where the constant computational complexity of bipartite matching [59] is eliminated because the number of passive connection points to be matched is at most 11 in the face centered cubic packing [27]. Thus the total computational complexity of the IM algorithm is $\mathbb{O}(n^3)$.

### C. Tree-based Branch and Bound Algorithm

This subsection introduces a Tree-based Branch and Bound (TBB) algorithm that computes the global optimal solution in exponential time, as summarized in Alg. 2. The TBB algorithm proposes a new branching strategy and stage cost based on the tree-like properties of the configuration composed of MISO modules.

The architecture of the TBB algorithm is built on the loop of branching and pruning operations. An OPEN list commonly used in shortest path problems [60] is created to store nodes for further inspection. Initially, the OPEN list contains nodes generated by $P^{M^I_{\text{root}}}_{M^F_{[:]}}$, which represents that a root module $M^I_{\text{root}}$ is matched with $n$ vacancies $M^F_{[:]}$ separately (Alg. 2 Line 2). Next, the TBB algorithm initializes a variable UPPER by the number of reconfiguration steps of the near-optimal solution, and initializes a variable LOWER by the lower bound of the least reconfiguration steps [24] (Alg. 2 Line 3). While OPEN is not empty and UPPER $>$ LOWER, branching operations and bound-based pruning operations will continue (Alg. 2 Line 4). The branching operation takes a node $x$ popped from OPEN as the parent node, and calls the BRANCHING function to generate child nodes (Alg. 2 Line 5-6). Each child node $y$ has a cost, $y$.cost, which is the sum of the cost of $x$ and the stage cost $c_{xy}$ calculated by the STAGE_COST function (Alg. 2 Line 7). If $y$.cost $\geq$ UPPER, the node $y$ and its descendants are pruned (Alg. 2 Line 9). The node $y$ has two more variables, $y$.matches which contains the matches of the current node, and $y$.all which contains all matches of the current node and its ancestors, as shown in Fig. 11(c). If $y$.cost $<$ UPPER and the length of $y$.all $< n$, the node $y$ is judged as promising and appended to the OPEN list for further inspection (Alg. 2 Line 11-12). If $y$.cost $<$ UPPER and the length of $y$.all $= n$, a further-optimized solution is obtained and UPPER is lowered to $y$.cost (Alg. 2 Line 14-15). The loop of branching and pruning operations continuously reduces the value of UPPER and the length of OPEN until a global optimal solution is obtained (Alg. 2 Line 19-20).

The BRANCHING function consists three operations to ensure that all possible child nodes are generated. First, for each match $(M^I_u, M^F_v)$ in $x$.matches, if the number of child

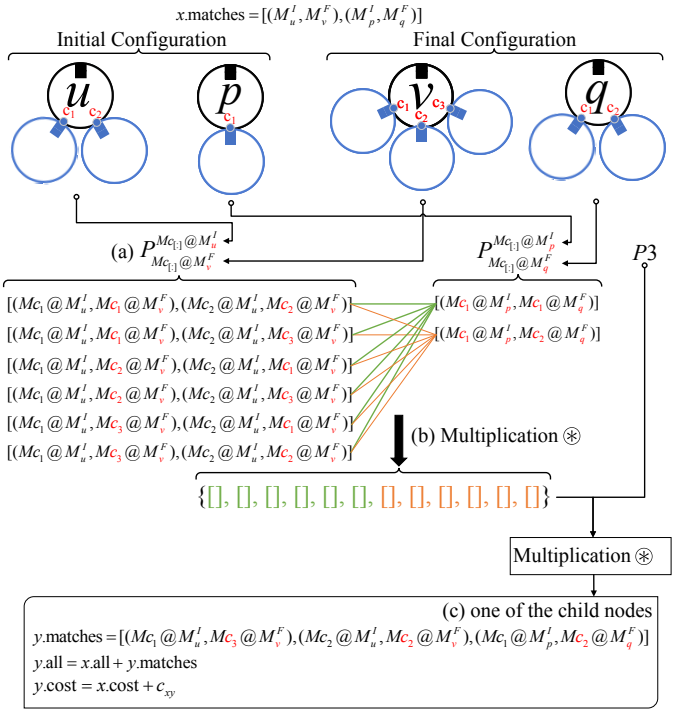



Fig. 10. The branching strategy of the TBB algorithm defines two types of operations, (a) permutation and (b) multiplication, to generate (c) child nodes.

modules of $M^I_u$ is less than the number of child vacancies of $M^F_v$, denoted as $a < b$, all ordered arrangements of $a$ distinct elements selected from the set of child vacancies of $M^F_v$ will be element-wise matched with an ordered arrangement of child modules of $M^I_u$. The opposite of $a > b$ is similar. This type of operations is denoted as permutation $P^{Mc_{[:]}@M^I_u}_{Mc_{[:]}@M^F_v}$ (Alg. 2 Line 23-25). For example, in Fig. 10(a), the two child modules of $M^I_u$ are element-wise matched with six ($A^2_3 = 6$) different arrangements of two of the three child vacancies of $M^F_v$. For another match $(M^I_p, M^F_q)$ in $x$.matches in Fig. 10, $P^{Mc_{[:]}@M^I_p}_{Mc_{[:]}@M^F_q}$ generates two ($A^1_2 = 2$) child nodes. Second, we define a type of multiplication, denoted as $P^{Mc_{[:]}@M^I_u}_{Mc_{[:]}@M^F_v} \circledast P^{Mc_{[:]}@M^I_p}_{Mc_{[:]}@M^F_q}$, that fuses each child node in one set with each child node in the other set (Alg. 2 Line 26). For example, the 12 green or orange lines in Fig. 10(b) indicate that each of the $A^2_3$ child nodes is fused with each of the $A^1_2 = 2$ child nodes, respectively. Among them, the last orange line indicates the fusion of $[(Mc_1@M^I_u, Mc_3@M^F_v), (Mc_2@M^I_u, Mc_2@M^F_v)]$ and $[(Mc_1@M^I_p, Mc_2@M^F_q)]$, which generates $y$.matches in Fig. 10(c). This multiplication operation is associative. For example, if $x$.matches contains three matches, child nodes can be generated by $(P1 \circledast P2) \circledast P3)$ as shown in Fig. 10. Third, if the number of child nodes generated after the above two operations is zero but the length of $x$.all $< n$, a root module in the unmatched sub-configurations $M^I_{\text{root}\notin x.\text{all}}$ will be matched with all the unmatched vacancies $M^F_{[:]-x.\text{all}}$ respectively (Alg. 2 Line 27-29).

The stage cost defined by the STAGE_COST function is stage-accumulative and fast-computable. The input of the STAGE_COST function is the node $y$ generated by

**Algorithm 2** Tree-based Branch and Bound (TBB) algorithm

1: **Input:** $X_I, X_F$ matrices and the near-optimal solution $D$
2: Initialize the OPEN list by $P^{M^I_{\text{root}}}_{M^F_{[:]}}$
3: Initialize UPPER by the cost of $D$ and LOWER by [24]
4: **while** OPEN is not empty and UPPER $>$ LOWER **do**
5: Pop a node $x$ from OPEN
6: **for** Each child $y$ of $x$ from BRANCHING($x$) **do**
7: Set $y$.cost= $x$.cost + STAGE_COST($y$)
8: **if** $y$.cost$\geq$UPPER **then**
9: Prune the $y$ node and its descendants
10: **else**
11: **if** len($y$.all)$<$ $n$ **then**
12: Append a node $y$ to OPEN
13: **else**
14: Obtain a further-optimized $\hat{D}$
15: Set UPPER to be $y$.cost
16: **end if**
17: **end if**
18: **end for**
19: **end while**
20: Obtain a global optimal $D^*$
21: **Output:** an optimal $D^*$
22: **function** BRANCHING($x$)
23: **for** each match $(M^I_u, M^F_v)$ in $x$.matches **do**
24: Generate child nodes by $P^{Mc_{[:]}@M^I_u}_{Mc_{[:]}@M^F_v}$
25: **end for**
26: Multiplications
27: **if** len(child nodes)$=0$ and len($x$.all)$<$ $n$ **then**
28: Generate child nodes by $P^{M^I_{\text{root}\notin x.\text{all}}}_{M^F_{[:]-x.\text{all}}}$
29: **end if**
30: **return** child nodes
31: **end function**
32: **function** STAGE_COST($y$)
33: $c_{xy} = 0$
34: **for** each match $(M^I_u, M^F_v)$ in $y$.matches **do**
35: $c_{xy}$ += Difference of unmatched out-degree
36: $c_{xy}$ += Difference of unmatched in-degree
37: **end for**
38: **return:** $c_{xy}$
39: **end function**

the BRANCHING function. For each match $(M^I_u, M^F_v)$ in $y$.matches, $c_{xy}$ adds the difference of unmatched out-degree between $M^I_u$ and $M^F_v$ (Alg. 2 Line 33-35). The unmatched out-degree refers to the number of unmatched child modules of $M^I_u$ or unmatched child vacancies of $M^F_v$ distinguished by $y$.all. In-degree is handled similarly (Alg. 2 Line 36).

Figure 11 introduces an example to illustrate the process by which TBB computes the optimal solution. The near-optimal solution calculated by the IM algorithm for the reconfiguration instance shown in Fig. 11(a) consists of 4 steps, namely $M^I_8 \nRightarrow M^I_1, M^I_8 \Rightarrow M^I_0$ and $M^I_9 \nRightarrow M^I_2, M^I_9 \Rightarrow M^I_3$. An optimal solution calculated by TBB consists of 2 steps, namely $M^I_2 \nRightarrow M^I_1, M^I_0 \Rightarrow M^I_3$. Initially, the root module $M^I_0$ is assigned to 12 vacancies respectively to generate 12 nodes,

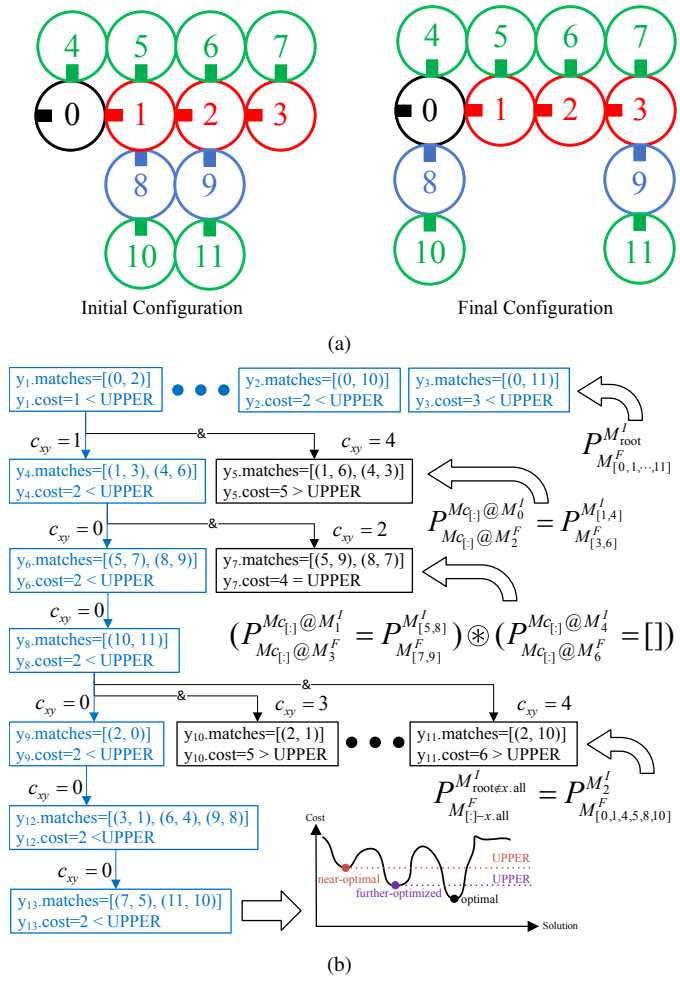



Fig. 11. The TBB algorithm is applied to (a) a reconfiguration instance and solves the optimal solution through (b) a process of cyclic branching and pruning operations.

namely $P^{M^I_{\text{root}}}_{M^F_{[0,\ 1,\cdots,11]}}$. Among them, $y_1$.matches= $[(0,2)]$ contains one of the matches in the optimal solution. Starting at $y_1$, $P^{Mc_{[:]}@M^I_0}_{Mc_{[:]}@M^F_2} = P^{M^I_{[1,4]}}_{M^F_{[3,6]}}$ generates 2 child nodes $y_5$ and $y_4$. Since $y_5$.cost=$y_1$.cost+$c_{xy} = 5 >$UPPER, $y_5$ and its descendants are pruned. Since $y_4$.matches contains two matches, a multiplication operation $(P^{Mc_{[:]}@M^I_1}_{Mc_{[:]}@M^F_3} = P^{M^I_{[5,8]}}_{M^F_{[7,9]}}) \circledast (P^{Mc_{[:]}@M^I_4}_{Mc_{[:]}@M^F_6} = [])$ is involved to generate child nodes. When TBB reaches $y_8$ whose matches only contains leaf modules or vacancies, $P^{M^I_{\text{root}\notin x.\text{all}}}_{M^F_{[:]-x.\text{all}}} = P^{M^I_2}_{M^F_{[0,1,4,5,8,10]}}$ is used instead to generate child nodes. These branching and pruning operations continue to reach an optimal solution, such as the one at the node $y_{13}$. In general, TBB may first obtain further-optimized solutions whose cost can be used as a new upper bound, as depicted in the lower right corner of Fig. 11. If we check all nodes in the OPEN list, an optimal solution can be guaranteed. Finally, we prove the completeness and computational complexity of the TBB algorithm as follows.

**Theorem 4.** *The TBB algorithm is guaranteed to reach an optimal solution with a computational complexity* $\mathbb{O}(n) = n(d!)^{\frac{n}{d}}$*, where* $d$ *is the maximum in-degree of the module.*

*Proof.* Since the number of different solutions with a cost less than any UPPER is limited, the TBB algorithm can terminate. Suppose an optimal solution with minimum cost is obtained in node $y_t$. We will show that the branching strategy of the TBB algorithm can generate all matches in $y_t$.all without being pruned.

Initially, the TBB algorithm uses $P^{M^I_{\text{root}}}_{M^F_{[:]}}$ to generate $n$ nodes, one of which must satisfy $y_1$.matches $\in y_t$.all. An example is $y_1$.matches=$[(M^I_0, M^F_2)]$ shown in Fig. 11(b). Starting from $y_1$, if the BRANCHING function can generate all possible child nodes, due to the tree-like properties of the configuration composed of MISO modules, there must be a child node of $y_1$ that satisfies $y_2$.matches $\in y_t$.matches. Similarly, there must be a child node of $y_2$ that satisfies $y_3$.matches $\in y_t$.matches. And so on, $t$ consecutive nodes $(y_1, y_2, \cdots, y_t)$ can reach an optimal solution.

The next proposition is that the BRANCHING function can generate all possible child nodes. First, the number of child nodes generated by $P^{Mc_{[:]}@M^I_u}_{Mc_{[:]}@M^F_v}$ in the BRANCHING function is exactly the number of ways we can rearrange the child vacancies into a new order that matches element-wise with child modules. Second, the multiplication defined in the BRANCHING function $P^{Mc_{[:]}@M^I_u}_{Mc_{[:]}@M^F_v} \circledast P^{Mc_{[:]}@M^I_p}_{Mc_{[:]}@M^F_q}$ makes a complete bipartite graph of child nodes of two sets. Third, when the number of child nodes output by the above two operations is zero but not all modules have been matched, $P^{M^I_{\text{root}\notin x.\text{all}}}_{M^F_{[:]-x.\text{all}}}$ generates all child nodes to initiate the matching process of the remaining modules and vacancies. Therefore, the BRANCHING function can generate $t$ nodes $(y_1, y_2, \cdots, y_t)$, all of which have a cost less than UPPER.

Since TBB is a sequential algorithm, we can only calculate the worst-case computational complexity with full enumeration. We consider two complete trees with maximum in-degree $d$. Initially, the TBB algorithm creates $n$ nodes whose $y$.matches contains 1 element, denoted as Layer 0. Starting from each node in Layer 0, the BRANCHING function generates $d!$ nodes whose $y$.matches contains $d$ elements, denoted as Layer 1. By analogy, starting from each node in Layer $L-1$, the BRANCHING function generates $(d!)^{d^{L-1}}$ nodes whose $y$.matches contains $d^L$ elements, denoted as Layer $L$. The number of matched modules and the number of generated nodes in all layers can be calculated by Equation (4) and Equation (5) respectively. After substituting $L = \log_d(1-(1-d)n)$ from Equation (4) into Equation (5), the number of generated nodes represented by $n$ is reduced as $\mathbb{O}(n) = n(d!)^{\frac{n}{d}}$.

$$n = 1 + d + \cdots + d^L = \frac{1-d^L}{1-d} \tag{4}$$

$$n \times (d!) \times \cdots \times (d!)^{d^{L-1}} = n(d!)^{\frac{1-d^{L-1}}{1-d}} \tag{5}$$

□

## V. EXPERIMENTS

This section introduces the controlled experiments of various algorithms and the feasibility experiments of the auto-optimizing method.

**Algorithm 3** The Pairing function in Modified Greedy-CM

```
function PAIRING(⟨M_u^I, M_v^F⟩)
    if M_u^I or M_v^F is a leaf module then
        return
    end if
    for each child module Mc_i@M_u^I do
        if a child vacancy is connected to c_i@M_v^F then
            Add a child node, ⟨Mc_i@M_u^I, Mc_i@M_v^F⟩
        end if
    end for
    for each child node do
        PAIRING(child node)
    end for
end function
```

### A. Experiment Setting

We conduct all experiments in a server with 48 cores clocked at 2.6GHz and 256G of memory. In the experiments, the algorithms used for comparison include local-procrustes [61], global-procrustes [61], Greedy-CM [36] and MDCOP [36].

The local-procrustes algorithm uses a permutation matrix to adjust the adjacency matrix of the final configuration so that the adjusted adjacency matrix is as similar as possible to the adjacency matrix of the initial configuration, as shown in Equation (6). In Equation (6), $P$ represents a permutation matrix, which has only one 1 in each row or column with the remaining entries as 0's. Meng [61] implemented an iterative optimization algorithm for solving the local minimum of Equation (6), denoted as local-procrustes. The global optimal solution can be obtained by traversing all permutation matrices, denoted as the global-procrustes algorithm. The calculation complexity of the global-procrustes algorithm is $\mathbb{O}(n!)$, which is only acceptable for problems with $n < 13$.

$$D^* = \min_{P\in\{0,1\}^{n\times n}, P^TP=I} \parallel P^T X_F P - X_I \parallel_F \tag{6}$$

Greedy-CM and MDCOP are algorithms proposed in [36] to compute near-optimal and optimal solutions of connection planning problems for configurations composed of SuperBots, whose four connection points are different from each other. The idea of Greedy-CM is to match the modules connected to $M^I_u$ and the vacancies connected to $M^F_v$ according to whether they are connected on the same type of connection point. Since the original Greedy-CM did not use Theorem 3 to simplify the calculation, for fairness, we borrow the outer structure of the IM algorithm but use the core idea of Greedy-CM to calculate the $u$-$v$ rooted CSG, as written in Alg. 3. In Alg. 3, the passive connection points of each MISO module are randomly marked as $c_1, c_2, \cdots, c_{11}$ to simulate different connection types. This variant algorithm is denoted as Modified Greedy-CM. MDCOP models the reconfiguration planning problem as a Distributed Constraint Optimization Problem (DCOP). MDCOP reduces the domain size of DCOP by limiting the set of candidate mates to only include modules with at least one edge of the same type. When MDCOP is applied to MISO modules without the different types of edges,

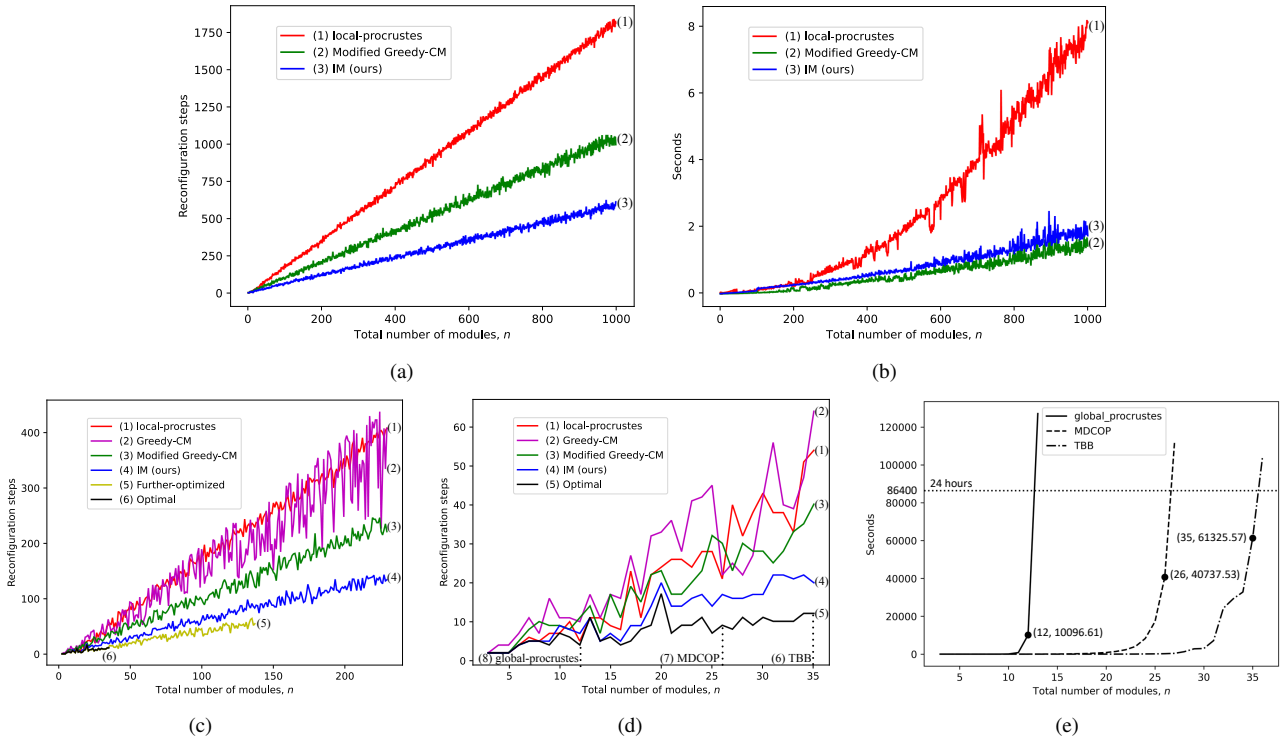



Fig. 12. Experimental results of inputting the first group of matrices to algorithms. (a)(b) The IM algorithm consumes a little more time in exchange for much fewer reconfiguration steps than Modified Greedy-CM. (c)(d)(e) The solutions computed by the IM algorithm are statistically near-optimal. Optimal solutions with scales up to $n \leq 12$, $n \leq 26$ and $n \leq 35$ were computed in 24 hours by global-procrustes, MDCOP and TBB, respectively, as shown by the dotted lines in (d) and the labeled points in (e).

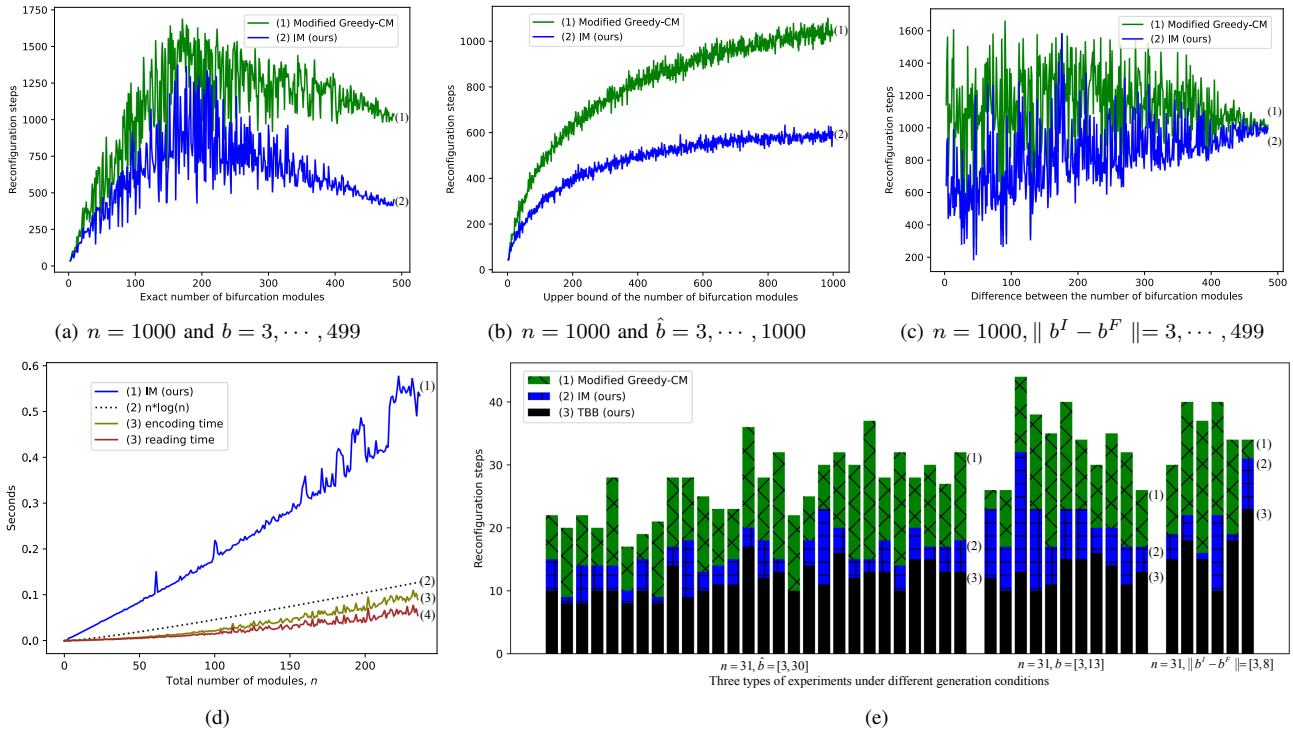
the constraint graph in DCOP is fully connected and the set of candidate mates contains all modules. This greatly degrades the performance of MDCOP [36] when $n > 26$.

We input four groups of random adjacency matrices $X^I$ and $X^F$ to the connection planning algorithms to reveal the influencing factors. The first group of matrices is used to reveal the influence of the total number of modules $n$ on the reconfiguration steps. For each $n = 3, \cdots, 1000$, we generate two adjacency matrices with dimensions $n$ satisfying the conditions that each row has at most one 1, and each column has at most eleven 1's [27]. The remaining three groups of matrices are used to reveal the influence of the number of bifurcation modules when the total number of modules remains unchanged ($n = 1000$). We denote $b$ as the exact number of bifurcation modules in the configuration, $\hat{b}$ as the upper bound of $b$ and $\| b^I - b^F \|$ as the difference between $b$ of the initial configuration and $b$ of the final configuration. The second group of matrices are 496 pairs of $X^I$ and $X^F$ with generation conditions, $n = 1000$ and $b = [3, 499]$. The third group of matrices are 997 pairs of $X^I$ and $X^F$ with generation conditions, $n = 1000$ and $\hat{b} = [3, 1000]$. The fourth group of matrices are 496 pairs of $X^I$ and $X^F$ with generation conditions, $n = 1000$ and $\| b^I - b^F \| = [3, 499]$. $b$ cannot be greater than $\lceil \frac{n}{2} \rceil$, but $\hat{b}$ can. $\hat{b}$ only sets the upper bound of $b$. $b^I$ and $b^F$ are randomly generated separately. Therefore, the third group of matrices can be used to analyze the effects of bounded $b^I$ and $b^F$, and the fourth group of matrices directly controls $\| b^I - b^F \|$ to identify any trends.

## B. Experiment Results

Figures 12 and 13 show the results of inputting the first group of matrices and the remaining three groups of matrices to various connection planning algorithms.

*1) Factor 1: Number of Modules:* Figure 12(a) shows that the IM algorithm outperforms local-procrustes and Modified Greedy-CM on the number of reconfiguration steps. There are two reasons for this result. Firstly, local-procrustes does not make use of the properties of Theorem 3. Their results are bound to be worse than graph-based algorithms. In fact, the loss landscape of the connection planning problem is very bumpy in dimension $n$. For a certain initial value, the result calculated by the local-procrustes will inevitably be limited to a local minimum, where the gradient disappears. Secondly, when the passive connection points can be interchanged, the lower bound of the least reconfiguration steps will be smaller according to Theorem 1, which means that the number of modules that can be matched by the Pairing function will be bigger than that of the idea of Greedy-CM. Thus the IM algorithm performs better when applied to a configuration composed of modules with interchangeable connection points. Figure 12(b) shows that the computation time of the IM algorithm is much less than local-procrustes but slightly more than Modified Greedy-CM. The reason why the IM algorithm consumes a little more time than Modified Greedy-CM is that the Pairing function used in the IM algorithm calls a bipartite matching algorithm [59] with a constant computational complexity. Experimentally, the above Pairing function is slower than the Pairing function of Modified Greedy-CM shown in




(a) $n = 1000$ and $b = 3, \cdots, 499$ (b) $n = 1000$ and $\hat{b} = 3, \cdots, 1000$ (c) $n = 1000, \parallel b^I - b^F \parallel = 3, \cdots, 499$

(d) (e)

Fig. 13. Experimental results of inputting the remaining three groups of matrices to algorithms and the feasibility experiment of the auto-optimizing method. (a)(b)(c) Three groups of experiments reveal the effect of changing the number of bifurcation modules on the performance of the IM algorithm. (d)(e) The combination of configuration pointer, IM and TBB realize read capability and automatic optimization capability.

Alg. 3. The IM algorithm consumes a little more time in exchange for much fewer reconfiguration steps than Modified Greedy-CM.

Figure 12(c) reduces the scale to $n < 230$ and adds the results of Greedy-CM and TBB. The direct application of Greedy-CM to the configurations composed of MISO modules produces a large variance of results. This is because the configuration considered by Greedy-CM is a Connected Component (CC) that maintains connectivity, whereas the configuration we consider contains multiple CCs of diverse sizes. Greedy-CM may select two CCs with different sizes to start matching. Theorem 3 makes the results of Modified Greedy-CM less biased. The results of the TBB algorithm are divided into the optimal solutions for $n \in [3, 35]$ and the further-optimized solutions for $n \in [36, 138]$. Since the TBB algorithm struggles to complete its computation within 24 hours for $n > 35$, we use further-optimized solutions to show the trend for $n > 35$. Figure 12(d) reduces the scale further to $n < 35$ and adds the results of MDCOP and global-procrustes. MDCOP and global-procrustes can only handle scales of $n = 26$ and $n = 12$ respectively in 24 hours of computation time, as shown in Fig. 12(e), for reasons explained in Subsection V-A. As can be seen from Fig. 12(d), the cost of the optimal solution given by the TBB algorithm is the same as that of MDCOP or global-procrustes, although the specific reconfiguration steps may be different. Figure 12(c)(d)(e) also show that the solutions computed by the IM algorithm are statistically near-optimal. However, it is speculated that the gap between the solution of the IM algorithm and the optimal solution will increase as $n$ increases. We suggest that ultra-large MSRR with $n \geq 1000$ may not pursue the optimal solution.

*2) Factor 2: Number of Bifurcation Modules:* Figure 13(a) shows the results based on the second group of matrices with fixed $n = 1000$ and different $b$. As $b$ increases, the number of reconfiguration steps first increases, then decreases, and reaches the peak at around $b = 165$. The upward trend is because the increase of $b$ makes the configuration more complex. The downward trend is because the configuration contains more modules with an out-degree of 2, when $b$ gradually approaches the limit $\lceil \frac{n}{2} \rceil$. These modules and the leaf modules connected to them can be mostly matched at the same time.

Figure 13(b) shows the results based on the third group of matrices. As $\hat{b}$ increases, the number of reconfiguration steps increases due to the upward trend in Fig. 13(a). But the rising slope in Fig. 13(b) gradually drops due to the downward trend in Fig. 13(a). It can also be seen from Fig. 13(b) that when $\hat{b}$ in the configuration is small, the number of reconfiguration steps calculated by the IM algorithm is much smaller than the average number. In Subsection VI-A, we introduce an example of this phenomenon. In that example, the total number of modules is $n = 500$, and the number of bifurcation modules is 4. The number of reconfiguration steps is 23, which is much smaller than the average number 250 shown in Fig. 12(a).

Figure 13(c) shows the results based on the fourth group of matrices. The advantage of the IM algorithm is more pronounced when $\parallel b^I - b^F \parallel$ is small. If two configurations have similar numbers of bifurcation modules, the two configurations are more likely to be similar, and thus more steps can be reduced. As $\parallel b^I - b^F \parallel$ increases, the irreducible difference

between the two configurations gradually increases.

*3) Experiments for Method Feasibility:* Two experiments are conducted to verify the feasibility of the auto-optimizing connection planning method introduced in Subsection IV-A.

First, the near-optimal $D$ matrices are read from the library to test the effectiveness of configuration pointers. Figure 13(d) shows the time for each SR instance to be encoded as two configuration pointers and the time to read the $D$ matrix. In Fig. 13(d), curves of the encoding time and reading time with respect to the number of modules are slightly skewed. Theoretically, the complexity of reading based on the configuration pointer should be $\mathbb{O}(n\log_{10} n)$. Thus we draw a dashed line of $n\log n$ with a scaling factor 4500 in Fig. 13(d) to corroborate the trend of the data. The gap between the reading time and the calculation time of the IM algorithm also increases with the increase of $n$. The rationality of this gap can be verified by comparing their computational complexity. Compared with other graph signatures or configuration recognition methods, the computational complexity of the configuration pointer is preferred. Asadpour *et al.* [46] propose the graph signature to do the isomorphism detection between different configurations with a complexity of $\mathbb{O}(n^2+n\times s^n)$, and Asadpour *et al.* [47] improve the algorithm to get a complexity of $\mathbb{O}(n^2\times s)$, where $s$ is the symmetry factor of the module. The configuration pointer can also be used in the configuration recognition task. Liu and Yim [50] propose a distributed configuration recognition algorithm with a complexity $\mathbb{O}(n^2)$. The computational complexity of the configuration pointer is $\mathbb{O}(n\log_{10} n)$, which is relatively satisfactory. The advantage of the configuration pointer for the configuration recognition task is that there is no need to perform isomorphism detection with all known configurations. The mismatch of any digit in the configuration pointer can skip many known configurations.

Second, the TBB algorithm is invoked to further optimize $D$ matrices read from the library. The results in Fig. 13(e) are obtained by inputting 45 pairs of $X^I$ and $X^F$ with the same dimension $n = 31$ to the method flowchart in Subsection IV-A. The generation conditions of these 45 pairs of matrices are $\hat{b} = [3, 30]$, $b = [3, 13]$ and $\| b^I - b^F \| = [3, 8]$. As shown in Fig. 13(e), the optimal solutions are better than the near-optimal solutions with a margin of $4\% - 9\%$. The margin is obtained by counting the average reduction of reconfiguration steps relative to $2n$. Since near-optimal solutions are close to optimal solutions in reconfiguration steps, it is reasonable that the margin for further optimization is only $4\%-9\%$. This slight improvement is significant for MSRR in the rescue scenarios such as fire and earthquake. The reduction of reconfiguration steps can significantly save the time spent on the actual motion of the SR process.

## VI. Applications

This section provides examples to supplement the discussion about non-zero elements in the $D^*$ matrix and the reduction of reconfiguration steps.

### A. Non-zero elements in the $D^*$ matrix

This subsection explains the actions represented by non-zero elements in the $D^*$ matrix and how each reconfiguration step

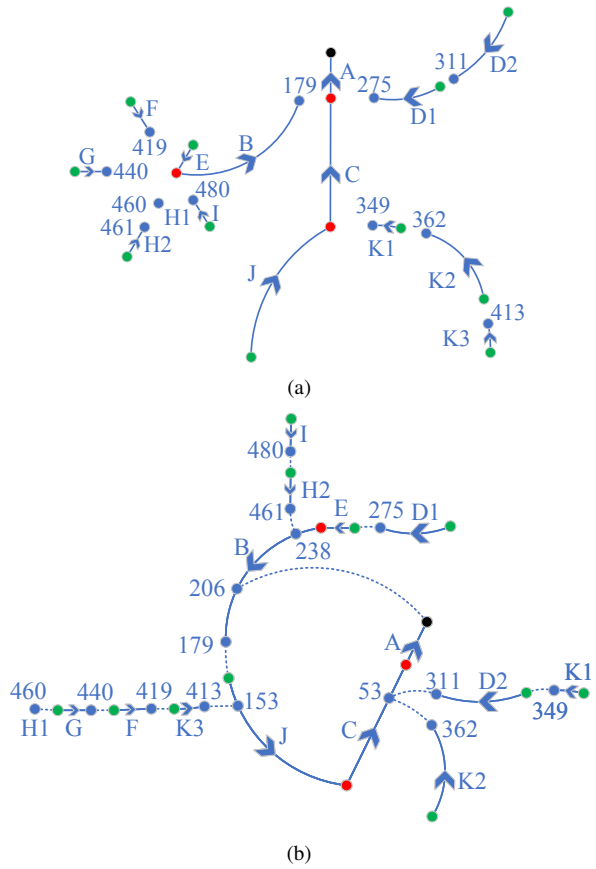



Fig. 14. An example to illustrate the role of non-zero elements in the $D^*$ matrix. (a) All detachment actions corresponding to $d_{uv} = -1$ decompose the initial configuration into several sub-graphs. (b) All attachment actions corresponding to $d_{uv} = 1$ reconnect sub-graphs as the final configuration.

TABLE III
The Difference Matrix Corresponding to Fig. 14

| Row | Attachment or detachment (column) |
|---|---|
| 0 | $0,\cdots,1(206),\cdots,0$ |
| 179 | $0,\cdots,-1(26),\cdots,1(178),\cdots,0$ |
| 275 | $0,\cdots,-1(26),\cdots,1(274),\cdots,0$ |
| 311 | $0,\cdots,1(53),\cdots,-1(310),\cdots,0$ |
| 349 | $0,\cdots,-1(96),\cdots,1(348),\cdots,0$ |
| 362 | $0,\cdots,1(53),\cdots,-1(361),\cdots,0$ |
| 413 | $0,\cdots,1(153),\cdots,-1(412),\cdots,0$ |
| 419 | $0,\cdots,-1(254),\cdots,1(418),\cdots,0$ |
| 440 | $0,\cdots,-1(254),\cdots,1(439),\cdots,0$ |
| 460 | $0,\cdots,-1(254),\cdots,1(459),\cdots,0$ |
| 461 | $0,\cdots,1(238),\cdots,-1(460),\cdots,0$ |
| 480 | $0,\cdots,-1(254),\cdots,1(479),\cdots,0$ |

can be completed. Different hardware implementations have different motion planning methods. The following only takes FreeBOT as an example.

Each FreeBOT has power supply, wireless communication and positioning capabilities, eliminating the need to maintain overall connectivity during self-reconfiguration. Therefore, the first motion planning method is to perform all detachments at once and then complete attachments by serpentine motion on the ground. This distributed method is suitable for large-scale self-reconfiguration with an assumption that each chain is supported by media such as ground and water to avoid collapse. Take the reconfiguration process from the configuration in Fig.

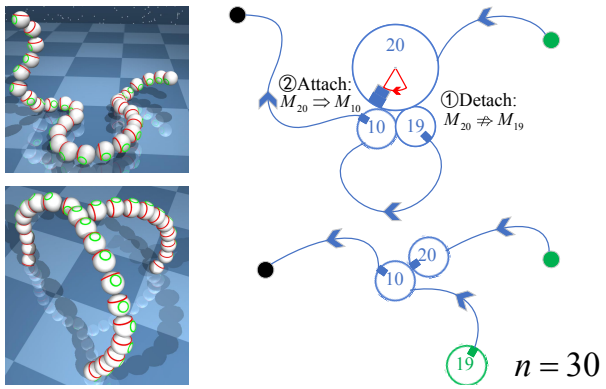



Fig. 15. The reconfiguration steps transforming a snake-shaped robot to a three-legged robot. The red and green circles on the module in the left images represent the module pose. A video animating these steps in zero gravity can be found in the multimedia extensions.

3 to the configuration in Fig. 4 as an example. The $D$ matrix calculated by the IM algorithm is summarized in Table III. As explained in Subsection V-B2, the IM algorithm can calculate the $D$ matrix with large dimensions in polynomial time, and $D$ is very sparse if there are few bifurcation modules in the configuration. In this example, the total number of modules is $n = 500$, and the exact number of bifurcation modules in the final configuration is $b = 4$. The corresponding detachment and attachment actions are drawn in Fig. 14. Figure 14(a) shows the effect of all detachments defined by the non-zero elements $d_{uv} = -1$ in the $D$ matrix. These detachments decompose the entire graph into many sub-graphs. The non-zero elements $d_{uv} = 1$ in the $D$ matrix means that the sub-graph with the module $M_u^I$ as the root module needs to be moved and attached to $M_v^I$. Figure 14(b) shows the effect of all attachments. In Fig. 14(b), we adjust the positions of decomposed sub-graphs and use dotted lines to indicate the required attachments. It can be seen from Fig. 14(b) that the attachments defined in the $D$ matrix can indeed form the final configuration shown in Fig. 4.

The second motion planning method is commanding chains composed of multiple modules to execute motions. This parallel method is suitable for situations where overall connectivity needs to be maintained and is an alternative to the first method when some chains may fall off such as in a bridge configuration. For example, Fig. 15 demonstrates the process of reconfiguration from a snake-shaped robot to a three-legged robot. The chain composed of $M_{10-20}$ is curled up to touch $M_{20}$ and $M_{19}$ with $M_{10}$ following the kinematics of the robotic arm. Then the detachment $M_{20} \nRightarrow M_{19}$ and the attachment $M_{20} \Rightarrow M_{10}$ are completed by the quick rotation of the inner trolley of FreeBOT. The quick rotation of the inner trolley can be replaced by the parallel movement of other hardware modules such as cross-ball [34].

### B. Reduction of Reconfiguration Steps

This subsection shows how the IM algorithm reduces reconfiguration steps by interchanging connection points and the superiority of the IM algorithm in large-scale self-reconfiguration.

Figure 16 illustrates how the solution of the IM algorithm reduces the reconfiguration steps compared to the solution of the Modified Greedy-CM algorithm. The Connection Number List (CNL) of $M_{10}^I$ in Fig. 16(b) is $[Nc_1@M_{10}^I = 6, Nc_2@M_{10}^I = 5, Nc_3@M_{10}^I = 6]$. Suppose the CNL of the bifurcation module in the final configuration is $[Nc_1@M_{10}^F = 5, Nc_2@M_{10}^F = 6, Nc_3@M_{10}^F = 6]$. The steps common to the two solutions of IM and Modified Greedy-CM are shown in Fig. 16(a). The root module $M_0^I$ is connected to $M_6^I$ through a robotic arm composed of $M_{[0,1,\cdots,6]}^I$. Then the detachment $M_3^I \nRightarrow M_2^I$ breaks a circuit into two branches. Figure 16(b) shows the extra steps in the solution of Modified Greedy-CM. Modified Greedy-CM does not consider the interchangeability of connection points. Because of $Nc_1@M_{10}^I - Nc_1@M_{10}^F = 6 - 5 = 1$ and $Nc_2@M_{10}^I - Nc_2@M_{10}^F = 5 - 6 = -1$, the sub-tree connected to $c1@M_{10}^I$ needs to give up a module $M_{16}^I$ to the sub-tree connected to $c2@M_{10}^I$. In Fig. 16(b), the chain consisting of modules on the path between $M_{25}^I$ and $M_{15}^I$ is curled up to touch $M_{25}^I$ and $M_{15}^I$ with $M_{16}^I$. Then the inner trolley of $M_{16}^I$ changes the connection quickly. These two steps can be reduced by interchanging $c_1@M_{10}^I$ and $c_2@M_{10}^I$, which results in $Nc_1@M_{10}^I - Nc_2@M_{10}^F = 6 - 6 = 0$ and $Nc_2@M_{10}^I - Nc_1@M_{10}^F = 5 - 5 = 0$. In the same way, the IM algorithm can reduce many other redundant steps caused by the different orders of CNs in CNL.

Figure 17 shows a more complex example of large-scale self-reconfiguration. In this example, the total number of modules is $n = 568$, and the exact number of bifurcation modules in the final configuration is $b = 28$. MSRR can be packaged into a compact configuration [62] that is easy to store and transport, such as the cube composed of six chains in Fig. 17(a). When performing tasks such as manipulation and exploration, the cube configuration can be reconfigured to a humanoid configuration. The 2D unfolded diagram in Fig. 17(b) shows the positions of detachments in the cube configuration. In Fig. 17(b), the positions of detachments are represented by crosses, and different colors are used to distinguish the results of the IM algorithm and the Modified Greedy-CM algorithm. The golden crosses represent the common detachments of the two algorithms. Figure 17(c) shows the positions of attachments in the humanoid configuration. In Fig. 17(c), the crosses represent the positions of attachments, and the branches around bifurcation modules are separated by dotted lines to make room for crosses of different colors. The distribution of crosses in Fig. 17(b)(c) shows that connection planning algorithms can keep most modules in the original connection relationship, and shows that the IM algorithm reduces some reconfiguration steps compared to the Modified Greedy-CM algorithm. In this example, the number of reconfiguration steps calculated by the IM algorithm is 91, and the number of reconfiguration steps calculated by the Modified Greedy-CM algorithm is 99. For instance, four green crosses outside the bifurcation modules in Fig. 17(c) are required by the Modified Greedy-CM algorithm but reduced by the IM algorithm. The underlying logic to reduce reconfiguration steps conforms to the example shown in Fig. 16. The superiority of the IM algorithm can be strengthened as the number of

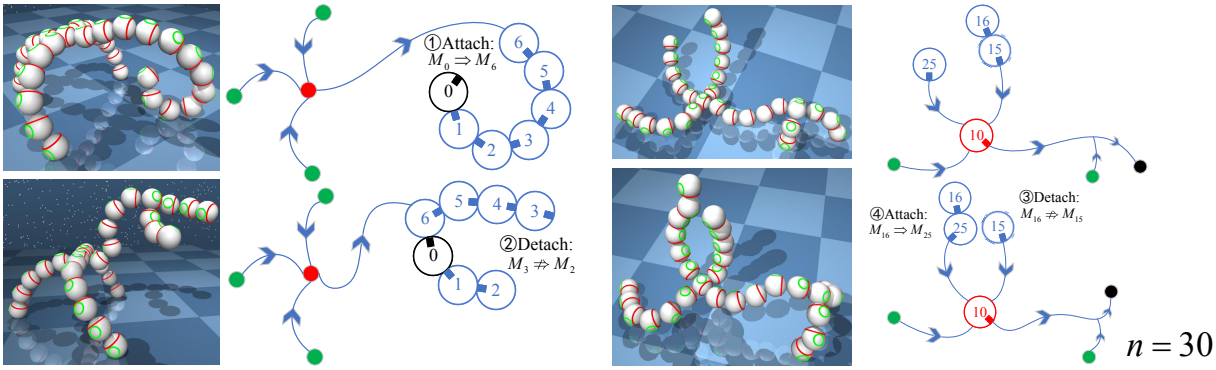



Fig. 16. The reduced reconfiguration steps. (a) The steps common to the two algorithms. (b) The steps defined by the Modified Greedy-CM algorithm but reduced by the IM algorithm. A video animating these steps in zero gravity can be found in the multimedia extensions.

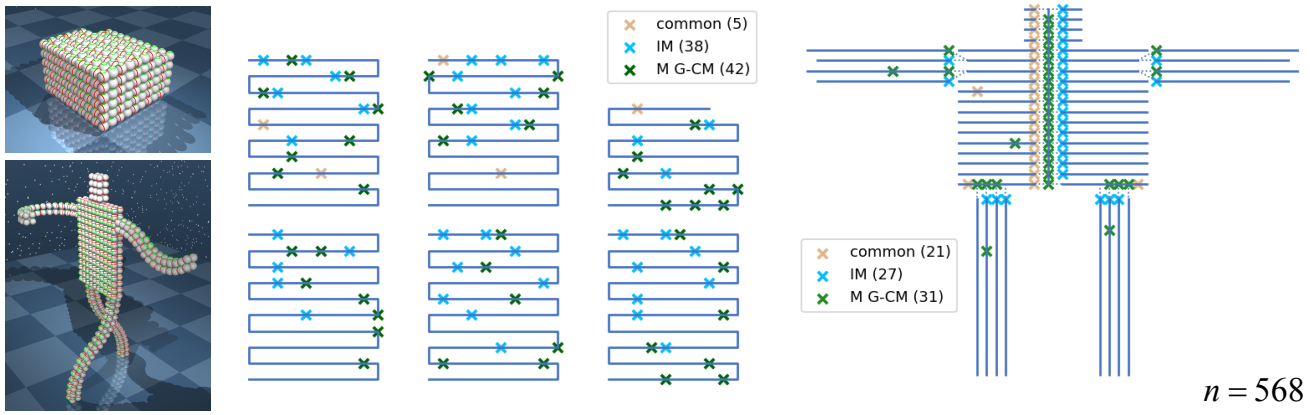



Fig. 17. Large-scale self-reconfiguration from the cube configuration to the humanoid configuration. (a) The two configurations, named cube and humanoid, respectively. (b) 2D unfolded diagram of the cube configuration. (c) 2D unfolded diagram of the humanoid configuration. The numbers in the legend indicate the number of detachments or attachments. The positions of attachments are mainly around the bifurcation modules.

bifurcation modules increases, as shown in the experimental data in Fig. 13.

## VII. Conclusions and Discussion

This article proposes an auto-optimizing connection planning method that combines a polynomial-time algorithm and an exponential-time algorithm through configuration pointers. The polynomial-time algorithm is used to calculate the near-optimal solutions of SR instances. The exponential-time algorithm is called to further optimize the solutions of frequently used SR instances in the library when some CPUs are idle. This method combines the rapidity of the polynomial-time algorithm and the optimality of the exponential-time algorithm for the first time. Second, our polynomial-time algorithm, IM, calculates near-optimal solutions better than local-procrustes and Modified Greedy-CM, which are state-of-the-art in the numerical optimization approach and the graph matching approach separately. This superiority is verified in experiments and based on theoretical proofs. Theorem 1 in this article shows that the interchangeability of the module's connection points can reduce the lower bound of the least reconfiguration steps. Third, our exponential-time algorithm, TBB, proposes a new branching strategy and stage cost, which provides a reference for the subsequent development of optimal solvers in the field of ORP. TBB can use the cost of the near-optimal solution as an upper bound, and output further-optimized solutions anytime to update the library until an optimal solution is guaranteed. These characteristics of our TBB algorithm are exactly in line with the requirements of the auto-optimizing method.

Previous studies have proposed some useful concepts or algorithms for reference. For example, Hou and Shen [36] propose MDCOP and Greedy-CM. Khodr *et al.* [63] propose an isomorphic invariant signature for Roombot to avoid repeated searches in the configuration space. Daudelin *et al.* [64] use the concept of the library to save some frequently used configurations and their behavioral characteristics. The auto-optimizing connection planning method proposed in this article integrates these works and carries out innovations. The IM algorithm surpasses other polynomial-time algorithms in terms of the optimality of solutions theoretically and experimentally. The configuration pointer has the smallest computational complexity among the existing configuration recognition algorithms. The TBB algorithm pioneered the idea of using branching and pruning operations to gradually approach the optimal solution of connection planning. The significant perspective of

our auto-optimizing method is to promote the combination of polynomial-time algorithms and exponential-time algorithms facing NP-complete problems. For example, both SMORES and Roombot have their own configuration recognition algorithms and polynomial-time algorithms, which can be used to implement their own auto-optimizing connection planning method.

However, some limitations are worth noting. The TBB algorithm further optimizes near-optimal solutions to the optimum with a computational complexity of $\mathbb{O}(n) = n(d!)^{\frac{n}{d}}$. This computational complexity is evaluated by the full enumeration without regard to decreasing upper bounds. The effective branching factor does not exceed one-sixth of the maximum in-degree $d$. However, the TBB algorithm still takes more than 24 hours to guarantee the optimal solution when $n > 35$. The computation time is still fairly high preventing the use in an online system when $n > 35$. There is still room for improvement in the computational complexity of these exponential-time algorithms. Another limitation of this article is that the main theorems and experiments are based on the settings of the MISO module. Other hardware modules that do not completely match the settings of the MISO module need to customize their own configuration recognition algorithm, polynomial-time algorithm, or exponential-time algorithm. Moreover, some chain-type MSRR, which cannot satisfy the unfoldable [56] and singularity-free [57] assumptions in Subsection III-B, needs to select or adjust the reconfiguration steps output by connection planning according to the hardware motion constraints.

Motion planning is our future work. In motion planning, considerations include motion constraints, drive characteristics, dynamics, gravity stability, collision detection, etc. We will study motion planning for free-form MSRR such as FreeBOT. 'Free-form' means that the attachment actions do not require precise alignment. The advantage of free-form MSRR is that motion planning can complete all detachment or attachment actions defined in the $D^*$ matrix. Therefore, connection planning does not need to avoid impossible actions caused by gimbal locks or unaligned positions. This allows connection planning to pursue optimality without additional worries. The optimal connection planning and free motion planning together will make self-reconfiguration fast and smooth.

## APPENDIX

In this appendix, we prove Theorem 2 by introducing an enumeration method of configuration spaces, which takes into account the interchangeability of the passive connection points of the MISO module. Since multiple Connected Components (CCs) can be connected to a virtual root module to become a rooted tree composed of $n+1$ modules. Thus we can only consider the configuration containing one CC that has been converted as a rooted tree.

In the following, $S(n)$ represents the number of all non-isomorphic configurations composed of $n$ modules, namely the volume of the configuration space. Figure 18 shows the configuration spaces when $n = 3, 4, 5$. In Fig. 18, we can observe that there is an inclusion relationship between the configuration space of $n$ and the configuration space of $n-1$. For example, the four configurations classified by CNL$=[4]$ in the configuration space of $n = 5$ are generated by connecting the four configurations in the configuration space of $n = 4$ to a new root module. In general, each sub-tree containing $Nc_i@M_u$ modules can be any one of $S(Nc_i@M_u)$ non-isomorphic configurations in the configuration space of $n = Nc_i@M_u$. Therefore, if the passive connection points cannot be interchanged, the number of enumerated configurations corresponding to one CNL is the product $S(Nc_1@M_u)S(Nc_2@M_u)\cdots S(Nc_t@M_u), t \leq d$. The sum of these products corresponding to all possible CNLs is the volume of the configuration space, $S(n)$. Calculating all possible CNLs and enumerating non-isomorphic configurations corresponding to each CNL are the two main steps of the enumeration method of configuration spaces. In particular, these two steps need to be customized to exclude isomorphic configurations caused by the interchangeability of the passive connection points of the MISO module, as explained by Lemma 1 and 2 in the following proof.

*Proof.* Firstly, CNLs with only the difference in the arrangement order of CNs can be represented by one typical CNL for enumeration, denoted as a category CNL. For example, in Fig. 18, when enumerating the configuration space of $n = 4$, the CNL=$[1, 2]$ of the root module $M_u$ is considered, and the CNL=$[2, 1]$ is excluded. All category CNLs with a certain length $t$ can be obtained by solving the constrained linear indeterminate equation in Equation (7). The method to solve the linear indeterminate equation follows [65]. Table IV shows all category CNLs solved according to different values of $t$ when enumerating the configuration space of $n = 8 < d$. For another example, in Fig. 18, when $n = 5$, the set of all category CNLs of the root module is $\{[4], [1, 3], [2, 2], [1, 1, 2], [1, 1, 1, 1]\}$. The volume of the configuration space is $S(5) = S(4) + S(1)S(3)+S(2)S(2)+S(1)S(1)S(2)+S(1)S(1)S(1)S(1) = 9$. Rule (3) in Subsection IV-A specifies that the CNs in the CNL are sorted from largest to smallest, thereby excluding other CNLs that only have a difference in the arrangement order of CNs. Lemma 1 summarizes the above analysis.

$$s.t. \begin{cases} \sum_{i=1}^{t} Nc_i@M_u = n-1, t \leq \min\{11, n-1\} \\ 1 \leq Nc_1@M_u \leq Nc_2@M_u \leq \cdots \leq Nc_t@M_u \\ Nc_1@M_u, Nc_2@M_u, \cdots, Nc_t@M_u \in \mathbb{Z} \end{cases} \tag{7}$$

**Lemma 1.** *The encoding rule (3) of the configuration pointer excludes isomorphic configurations caused by CNLs with a different arrangement of CNs.*

Secondly, if the category CNL contains $r$ equal CNs, interchanging the corresponding $r$ passive connection points will generate isomorphic configurations. Take the $[1, 3, 3]$ in Tale IV as an example, there are $r = 2$ passive connection points

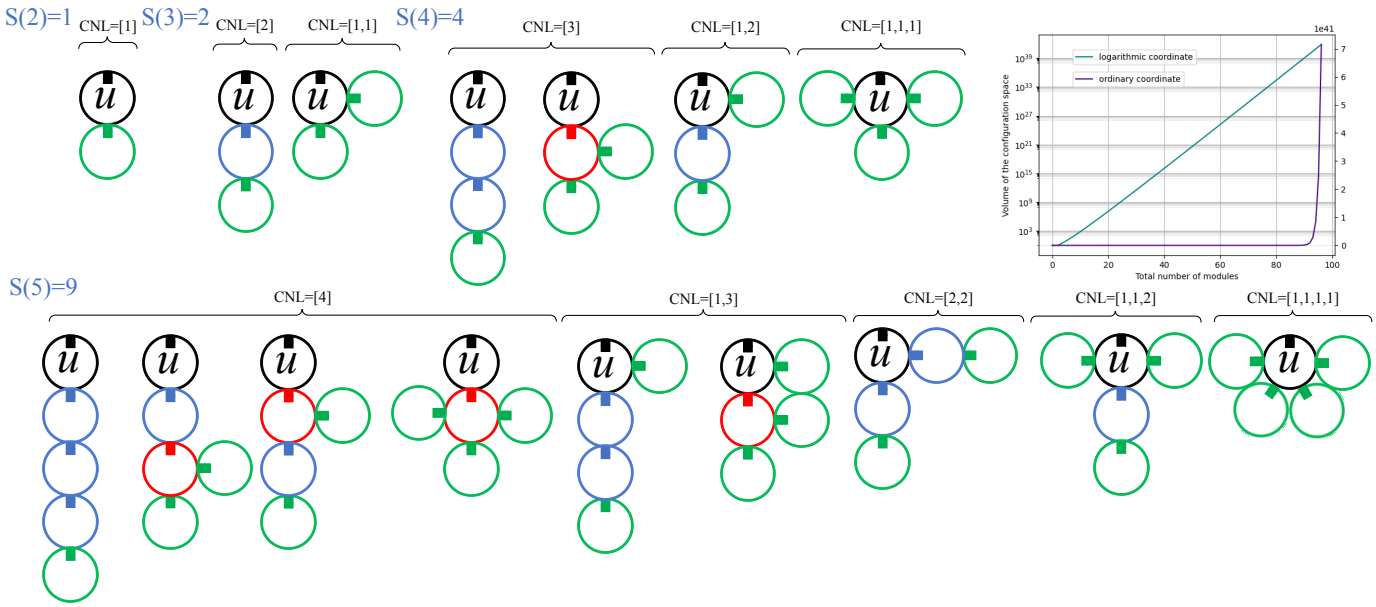



Fig. 18. The enumeration method of configuration spaces, which enumerates all possible CNLs and their corresponding non-isomorphic configurations.

TABLE IV
CATEGORY CNLS WITH A LENGTH $t$ WHEN $n = 8$

| t | category CNLs | t | category CNLs |
|---|---|---|---|
| 1 | [7] | 4 | [1, 1, 1, 4] |
| 2 | [1, 6] | | [1, 1, 2, 3] |
| | [2, 5] | | [1, 2, 2, 2] |
| | [3, 4] | 5 | [1, 1, 1, 1, 3] |
| 3 | [1, 1, 5] | | [1, 1, 1, 2, 2] |
| | [1, 2, 4] | 6 | [1, 1, 1, 1, 1, 2] |
| | [1, 3, 3] | 7 | [1, 1, 1, 1, 1, 1, 1] |
| | [2, 2, 3] | / | / |

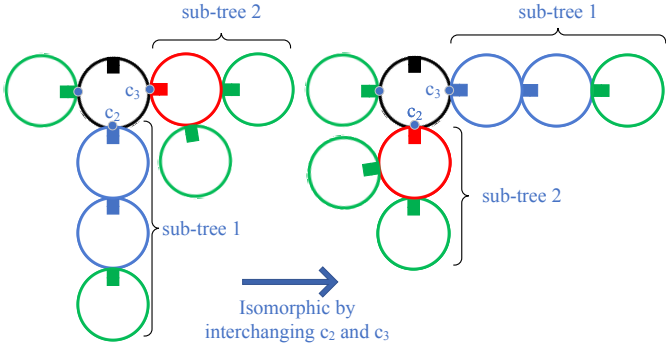



Fig. 19. The isomorphic configuration produced by interchanging passive connection points.

connected by sub-trees composed of 3 modules, and each sub-tree can be one of $S(3) = 2$ different configurations in the configuration space of $n = 3$. Let's label these two different configurations as the sub-tree 1 and the sub-tree 2 in Fig. 19. The number of different configurations enumerated by connecting the two sub-trees at the two passive connection points in turn is not $S(3)S(3) = 4$, but should be replaced with $C^r_{S(3)+r-1} = C^2_{2+2-1} = 3$. This replacement excludes the isomorphic configurations created by interchanging passive connection points. As shown in Fig. 19, the configuration created by connecting the sub-tree 1 and the sub-tree 2 to $c_2@M_u$ and $c_3@M_u$ is isomorphic with the configuration created by connecting the sub-tree 1 and the sub-tree 2 to $c_3@M_u$ and $c_2@M_u$. Therefore, the number of enumerated non-isomorphic configurations corresponding to the category CNL $[1, 3, 3]$ should be $S(1) * C^2_{2+2-1}$. In general, if $Nc_i@M_u, \cdots, Nc_j@M_u$ are $r$ equal CNs, $S(Nc_i@M_u) \cdots S(Nc_j@M_u)$ will be replaced by $C^r_{S(Nc_j@M_u)+r-1}$, which represents the number of combinations of $r$ repeatable configurations selected from $S(Nc_j@M_u)$ configurations. Rule (2) in Subsection IV-A specifies that the order of repeated CNs is determined by the value of the configuration pointer of the corresponding sub-tree. Take the configurations in Fig. 7 as an example. Although $Nc_2@M_u = Nc_3@M_u = 3$, due to $1100 > 210$, two isomorphic configurations produced by interchanging $c_2$ and $c_3$ as shown in Fig. 19 will generate the same configuration pointer. Thus we have Lemma 2.

**Lemma 2.** *The encoding rule (2) of the configuration pointer excludes isomorphic configurations caused by CNLs with repeated CNs.*

In summary, the three encoding rules of the configuration pointer can make any two isomorphic configurations generate the same configuration pointer, so non-isomorphic configurations in the configuration space have different configuration pointers.

Finally, we prove that the length growth rate of the configuration pointer is $O(n \log_{10} n)$. Suppose the configuration to be encoded is a chain. The length of the configuration pointer can be written as Equation (8), where $CN$ represents the CN value of a sub-tree containing $CN \in [1, n]$ modules in the chain. The logarithm of $CN$ is the number of digits of the CN value. The length of the configuration pointer is the sum of the number of digits of each CN value. Equation (8) is

then simplified by the limitation calculated in Equation (9) as $f(n) = n \log_{10} n$.

$$f(n) \approx \sum_{CN=1}^{n} \log_{10} CN = \log_{10}(n!) \tag{8}$$

$$\lim_{n\to\infty} \log_{10}(n!) = n \log_{10} n \tag{9}$$

□

## REFERENCES


[1] H. Ahmadzadeh and E. Masehian, "Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization," *Artificial Intelligence*, vol. 223, pp. 27–64, 2015.

[2] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[3] J. W. Suh, S. B. Homans, and M. Yim, "Telecubes: Mechanical design of a module for self-reconfigurable robotics," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 4. IEEE, 2002, pp. 4095–4101.

[4] R. Fitch, Z. J. Butler, and D. Rus, "The crystal robot: Implementation and demonstration." in *AAAI Mobile Robot Competition*, 2002, pp. 65–71.

[5] C. Unsal and P. K. Khosla, "A multi-layered planner for self-reconfiguration of a uniform group of i-cube modules," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, vol. 1. IEEE, 2001, pp. 598–605.

[6] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund, "Modular atron: Modules for a self-reconfigurable robot," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 2. Ieee, 2004, pp. 2068–2073.

[7] B. Kirby, J. Campbell, B. Aksak, P. Pillai, J. Hoburg, T. C. Mowry, and S. C. Goldstein, "Catoms: Moving robots without moving parts," in *Proceedings of the national conference on artificial intelligence*, vol. 20, no. 4. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 1730.

[8] P. White, K. Kopanski, and H. Lipson, "Stochastic self-reconfigurable cellular robotics," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 3. IEEE, 2004, pp. 2888–2893.

[9] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu, "Miche: Modular shape formation by self-disassembly," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 345–372, 2008.

[10] R. F. M. Garcia, J. D. Hiller, K. Stoy, and H. Lipson, "A vacuum-based bonding mechanism for modular robotics," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 876–890, 2011.

[11] W.-M. Shen, B. Salemi, and P. Will, "Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots," *IEEE transactions on Robotics and Automation*, vol. 18, no. 5, pp. 700–712, 2002.

[12] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: a modular reconfigurable robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 514–520.

[13] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata, "Distributed self-reconfiguration of m-tran iii modular robotic system," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 373–386, 2008.

[14] V. Zykov, A. Chan, and H. Lipson, "Molecubes: An open-source modular robotics kit," in *IROS-2007 Self-Reconfigurable Robotics Workshop*. Citeseer, 2007, pp. 3–6.

[15] B. Salemi, M. Moll, and W.-M. Shen, "Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 3636–3641.

[16] M. Park and M. Yim, "Distributed control and communication fault tolerance for the ckbot," in *2009 ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*. IEEE, 2009, pp. 682–688.

[17] A. Lyder, R. F. M. Garcia, and K. Stoy, "Mechanical design of odin, an extendable heterogeneous deformable modular robot," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee, 2008, pp. 883–888.

[18] J. Dietsch, R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. J. Ijspeert, "Exploring adaptive locomotion with yamor, a novel autonomous modular robot with bluetooth interface," *Industrial Robot: An International Journal*, 2006.

[19] A. Spröwitz, S. Pouya, S. Bonardi, J. Van Den Kieboom, R. Möckel, A. Billard, P. Dillenbourg, and A. J. Ijspeert, "Roombots: reconfigurable robots for adaptive furniture," *IEEE Computational Intelligence Magazine*, vol. 5, no. 3, pp. 20–32, 2010.

[20] Y. Tu, G. Liang, and T. L. Lam, "Freesn: A freeform strut-node structured modular self-reconfigurable robot-design and implementation," in *2022 IEEE International Conference on Robotics and Automation (ICRA). IEEE*, 2022.

[21] J. E. Walter, E. M. Tsai, and N. M. Amato, "Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots," *IEEE transactions on Robotics*, vol. 21, no. 4, pp. 621–631, 2005.

[22] H. Kawano, "Linear heterogeneous reconfiguration of cubic modular robots via simultaneous tunneling and permutation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 332–338.

[23] A. Casal and M. H. Yim, "Self-reconfiguration planning for a class of modular robots," in *Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839. International Society for Optics and Photonics, 1999, pp. 246–257.

[24] F. Hou and W.-M. Shen, "On the complexity of optimal reconfiguration planning for modular reconfigurable robots," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2791–2796.

[25] H. Luo and T. L. Lam, "Adaptive flow planning of modular spherical robot considering static gravity stability," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4228–4235, 2022.

[26] C. Liu, M. Whitzer, and M. Yim, "A distributed reconfiguration planning algorithm for modular robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4231–4238, 2019.

[27] M. Yim, Y. Zhang, J. Lamping, and E. Mao, "Distributed control for 3d metamorphosis," *Autonomous Robots*, vol. 10, no. 1, pp. 41–56, 2001.

[28] R. Fitch and Z. Butler, "Million module march: Scalable locomotion for large self-reconfiguring robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 331–343, 2008.

[29] H. Luo, G. Liang, M. Li, H. Qian, and T. Lam, "An obstacle-crossing strategy based on the fast self-reconfiguration for modular sphere robots," in *Proceedings 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020)*. IEEE, 2020.

[30] G. Liang, H. Luo, M. Li, H. Qian, and T. Lam, "Freebot: A freeform modular self-reconfigurable robot with arbitrary connection point - design and implementation," in *Proceedings 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020)*. IEEE, 2020.

[31] D. Zhao and T. L. Lam, "Snailbot: a continuously dockable modular self-reconfigurable robot using rocker-bogie suspension," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 4261–4267.

[32] S. Niu, L. Ye, H. Liu, B. Liang, and Z. Jin, "Ant3dbot: A modular self-reconfigurable robot with multiple configurations," in *International Conference on Intelligent Robotics and Applications*. Springer, 2022, pp. 552–563.

[33] P. Swissler and M. Rubenstein, "Fireant3d: a 3d self-climbing robot towards non-latticed robotic self-assembly," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3340–3347.

[34] Y. Meng, Y. Zhang, A. Sampath, Y. Jin, and B. Sendhoff, "Cross-ball: a new morphogenetic self-reconfigurable modular robot," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 267–272.

[35] F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo, "Swarm-bot: A new distributed robotic concept," *Autonomous robots*, vol. 17, no. 2, pp. 193–221, 2004.

[36] F. Hou and W.-M. Shen, "Graph-based optimal reconfiguration planning for self-reconfigurable robots," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1047–1059, 2014.

[37] C. A. Nelson, "A framework for self-reconfiguration planning for unit-modular robots," Ph.D. dissertation, Purdue University, 2005.

[38] J. J. McGregor, "Backtrack search algorithms and the maximal common subgraph problem," *Software: Practice and Experience*, vol. 12, no. 1, pp. 23–34, 1982.

[39] A. Jain, "Graph theoretic foundations of multibody dynamics," *Multibody system dynamics*, vol. 26, no. 3, pp. 307–333, 2011.

[40] Y. Meng and Y. Jin, "Morphogenetic self-reconfiguration of modular robots," in *Bio-inspired self-organizing robotic systems*. Springer, 2011, pp. 143–171.
[41] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Cellular automata for decentralized control of self-reconfigurable robots," in *Proc. of the ICRA 2001 workshop on modular robots*. Citeseer, 2001, pp. 21–26.
[42] P. Varshavskaya, L. P. Kaelbling, and D. Rus, "Automated design of adaptive controllers for modular robots using reinforcement learning," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 505–526, 2008.
[43] Z. Ramaekers, R. Dasgupta, V. Ufimtsev, S. Hossain, and C. A. Nelson, "Self-reconfiguration in modular robots using coalition games with uncertainty." *Automated Action Planning for Autonomous Mobile Robots*, vol. 11, p. 09, 2011.
[44] S. Shiba, M. Uchida, A. Nozawa, H. Asano, H. Onogaki, T. Mizuno, H. Ide, and S. Yokoyama, "Autonomous reconfiguration of robot shape by using q-learning," *Artificial Life and Robotics*, vol. 14, no. 2, p. 213, 2009.
[45] K. Støy, "Controlling self-reconfiguration using cellular automata and gradients," in *Proceedings of the 8th international conference on intelligent autonomous systems (IAS-8)*, 2004, pp. 693–702.
[46] M. Asadpour, A. Sproewitz, A. Billard, P. Dillenbourg, and A. J. Ijspeert, "Graph signature for self-reconfiguration planning," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 863–869.
[47] M. Asadpour, M. H. Z. Ashtiani, A. Sproewitz, and A. Ijspeert, "Graph signature for self-reconfiguration planning of modules with symmetry," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 5295–5300.
[48] M. Eden, "A two-dimensional growth process," in *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*, vol. 4. University of California Press Berkeley, 1961, pp. 223–239.
[49] I.-M. Chen, "Theory and applications of modular reconfigurable robotic systems," Ph.D. dissertation, California Institute of Technology, 1994.
[50] C. Liu and M. Yim, "Configuration recognition with distributed information for modular robots," in *Robotics Research*. Springer, 2020, pp. 967–983.
[51] M. H. Yim, D. Goldberg, and A. Casal, "Connectivity planning for closed-chain reconfiguration," in *Sensor Fusion and Decentralized Control in Robotic Systems III*, vol. 4196. International Society for Optics and Photonics, 2000, pp. 402–412.
[52] M. Yu, Z. Ye, Y.-J. Liu, Y. He, and C. C. Wang, "Lineup: Computing chain-based physical transformation," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 1, pp. 1–16, 2019.
[53] F. Hou and W.-M. Shen, "Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 3135–3140.
[54] A. Gorbenko and V. Popov, "Programming for modular reconfigurable robots," in *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*, no. 5, 2011.
[55] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 403–421, 2008.
[56] J. Seo, S. Gray, V. Kumar, and M. Yim, "Reconfiguring chain-type modular robots based on the carpenter's rule theorem," in *Algorithmic Foundations of Robotics IX*. Springer, 2010, pp. 105–120.
[57] L. Zong, G. Liang, and T. L. Lam, "Kinematics modeling and control of spherical rolling contact joint and manipulator," *IEEE Transactions on Robotics*, 2022.
[58] J. Assaker, A. Makhoul, J. Bourgeois, and J. Demerjian, "A unique identifier assignment method for distributed modular robots," in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'20). IEEE*, 2020.
[59] R. M. Karp, "An algorithm to solve the m × n assignment problem in expected time o(mn log n)," *Networks*, vol. 10, no. 2, pp. 143–152, 1980.
[60] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1.
[61] F. Meng, "Python library for (generalized) procrustes problems," Website, 2021, https://procrustes.readthedocs.io/en/latest/.
[62] K. C. Cheung, E. D. Demaine, J. R. Bachrach, and S. Griffith, "Programmable assembly with universally foldable strings (moteins)," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 718–729, 2011.
[63] H. Khodr, M. Mutlu, S. Hauser, A. Bernardino, and A. Ijspeert, "An optimal planning framework to deploy self-reconfigurable modular robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4278–4285, 2019.
[64] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell, "An integrated system for perception-driven autonomy with modular robots," *Science Robotics*, vol. 3, no. 23, 2018.
[65] H. L. Keng, "Indeterminate equations," in *Introduction to Number Theory*. Springer, 1982, pp. 276–299.

**Haobo Luo** received the B.Eng. degree in electrical engineering and automation from Wuhan University of Technology, Wuhan, China, in 2017, and the M.Sc. degree in mechanical and automation engineering from the Chinese University of Hong Kong, Hong Kong, in 2019. He is currently working toward the Ph.D. degree in computer and information engineering at the Chinese University of Hong Kong, Shenzhen.

His research interests include modular self-reconfiguration robots, graph theory, numerical optimization and motion planning.

**Tin Lun Lam** (Senior Member, IEEE) received the B.Eng. (First Class Hons.) and Ph.D. degrees in robotics and automation from the Chinese University of Hong Kong, Hong Kong, in 2006 and 2010, respectively.

He is currently an Assistant Professor with the Chinese University of Hong Kong, Shenzhen, China, the Executive Deputy Director of the National-Local Joint Engineering Laboratory of Robotics and Intelligent Manufacturing, and the Director of Center for the Intelligent Robots, Shenzhen Institute of Artificial Intelligence and Robotics for Society. He has authored or coauthored two monographs and more than 50 research papers in top-tier international journals and conference proceedings in robotics and AI [IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON ROBOTICS, Journal of Field Robotics, IEEE/ASME TRANSACTIONS ON MECHATRONICS (T-MECH), IEEE ROBOTICS AND AUTOMATION LETTERS, IEEE International Conference on Robotics and Automation, and IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)]. He holds more than 70 patents. His research interests include multirobot systems, field robotics, and collaborative robotics.

Dr. Lam received an IEEE/ASME T-MECH Best Paper Award in 2011 and the IEEE/RSJ IROS Best Paper Award on Robot Mechanisms and Design in 2020.