Robotics and Autonomous Systems 62 (2014) 1047–1059

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

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

Graph-based optimal reconfiguration planning for self-reconfigurable robots

Feili Hou *, Wei-Min Shen

Information Sciences Institute, University of Southern California, United States

HIGHLIGHTS

- On finding the least (dis)connect actions to reconfigure between arbitrary shapes.
- A proof that the optimal reconfiguration planning problem is NP-complete.
- An algorithm which can generate the optimal reconfiguration sequence.
- An algorithm that finds the near-optimal reconfiguration sequence in polynomial time.

ARTICLE INFO

Article history:
Available online 12 September 2013

Keywords:
Modular robots
Optimal reconfiguration planning
Computational complexity

ABSTRACT

The goal of optimal reconfiguration planning (ORP) is to find a shortest reconfiguration sequence to transform a modular and reconfigurable robot from an arbitrary configuration into another. This paper investigates this challenging problem for chain-type robots based on graph representations and presents a series of theoretical results: (1) a formal proof that this is an NP-complete problem, (2) a reconfiguration planning algorithm called MDCOP which generates the optimal graph-based reconfiguration plan, and (3) another algorithm called GreedyCM which can find a near-optimal solution in polynomial time. Experimental and statistical results demonstrate that the solutions found by GreedyCM are indeed near-optimal and the approach is computationally feasible for large-scale robots.

1. Introduction

Different from conventional robots, self-reconfigurable robots can adapt their own configurations and offer more versatile capabilities for challenging tasks in different environments. In applications such as reconnaissance, rescue missions, and space applications where the task and the environment are not always fully known in advance, the ability to adapt shape may be critical for a robot to accomplish its tasks, maximize its potential, and recover from unexpected damage. Self-reconfigurable robots with modular architecture would be ideal for such situations. Thus, realizing the ability of self-reconfiguration with a large number of independent modules has been one of the most important and challenging topics in the field of modular and reconfigurable robots.

Based on the design of modules, self-reconfigurable robots can be classified into two main categories: lattice-type and chain-type. In lattice-type robots, modules are arranged in a 2D or 3D lattice space of cells, and reconfiguration is achieved by a module to detach from its current lattice location, move along the surface of other modules, and then dock to a module at an adjacent lattice cell. Examples of such robots include 3D Fracta [1], Molecule [2], Telecube [3], Crystalline [4,5], ICubes [6], ATRON [7], Catom [8], the Programmable Parts [9], Stochastic-3D [10], Miche [11], Vacuubes [12] etc. In chain-type robots, modules can form movable chains and loops of any graphical topology, and the reconfiguration is achieved through "connect" and "disconnect" operations between modules along with the joint motion of chains. Hardware implementations of this class of robots include CONRO [13,14], PolyBot G3 [15], M-TRAN III [16], Molecubes [17], SuperBot [18–20], CKBot [21], Odin [22], YamoR [23], Roombot [24], Motein [25] etc. The different geometric arrangement of modules between lattice-type and chain-type modular robots makes their reconfiguration planning mechanisms fundamentally different.

This paper is mainly focused on the reconfiguration planning of chain-typed robots. The objective of self-reconfiguration planning is to figure out the reconfiguration steps for changing connectivity among the modules so as to transform the robot from the current configuration into a goal configuration. In the distributed fashion, it is the question of how the modules coordinate with others to figure out the necessary connections and disconnections given that they can only communicate and sense locally. Currently, only a few works were published on chain-type reconfiguration.

Casal [26] first tackled the problem of chain reconfiguration and presented a divide-and-conquer approach to the problem. This

* Corresponding author.
E-mail addresses: fhou@usc.edu (F. Hou), shen@isi.usc.edu (W.-M. Shen).

0921-8890/$ – see front matter
http://dx.doi.org/10.1016/j.robot.2013.06.014

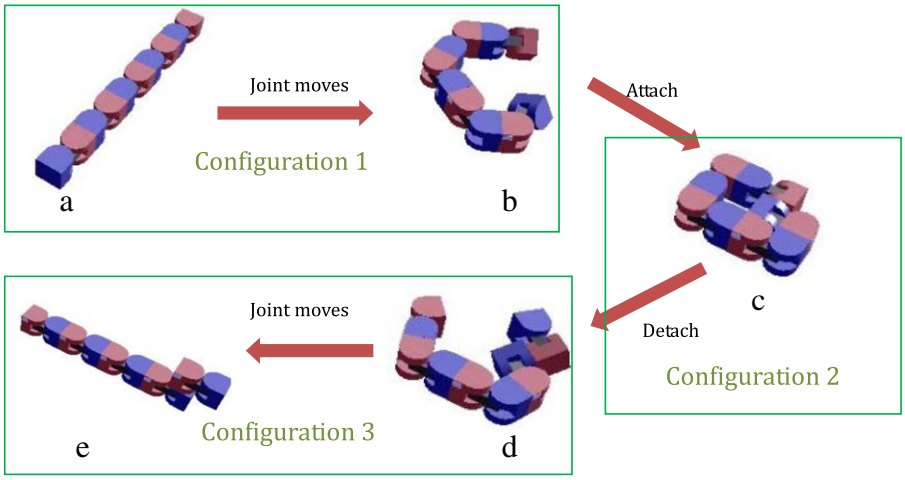


Fig. 1. An example of reconfiguration process.

algorithm is further developed by Yim et al. and theoretical results regarding its running time and the number of moves needed to reconfigure are given in [27]. Nelson [28] used the principal component analysis in conjunction with standard weighed bipartite graph matching theory to compare the initial and goal configurations and generate the reconfiguration steps. Payne [29] used a hormone inspired distributed controller and implemented the reconfiguration from "I" shape to "T" shape on CONRO. Gay [30] used the modules called DOF-Box II to units to build furniture that could change shape. Asadpour [31] has developed a self-reconfiguration planning method based on heuristic search, where the graph signatures method is used to test isomorphism between configurations, and the maximum common graph between configurations is used as a heuristic function to guide the search.

Aside from the research on reconfiguration planning, some other work on configurations is also indirectly related to our problem. I-Ming Chen [32] mentioned that the number of configurations is exponential, and even enumerating all the configurations is difficult. Castano [14] and Park [33] provide ways of finding the functionally identical configurations using graph isomorphism and configuration matching techniques. Chirikjian [34] established lower and upper bounds for the minimum number of moves needed for lattice-type reconfiguration, although it may not be applicable for chain-type reconfiguration.

The existing literature has used different techniques for finding an efficient reconfiguration sequence. However, the optimal solution with the least reconfiguration steps has never been reached. It is commonly believed that this problem is computationally intractable, but concrete evidence for this belief is still lacking. Some key research questions still remain open in this area. For example, how hard is it to find the least number of reconfiguration steps? Can an optimal solution be found efficiently? How to find the optimal solutions?

The goal of this paper is to investigate the optimal reconfiguration planning problem, i.e. finding the least number of reconfiguration steps to transform from one arbitrary configuration to another. During physical execution, some additional reconfiguration steps might be needed to compensate hardware limitation, which depend on many robot-dependent factors like the modules' internal degree of freedom, self-collision, gravity and others, and vary considerably from one module design to another. To gain the intrinsic theoretical insight of this problem, we exclude the hardware concerns for now, and work on the high-level connectivity planning in terms of graph representation in this paper.

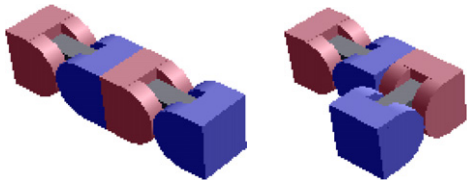
We first analyze the complexity of the optimal reconfiguration planning problem by rephrasing the optimal reconfiguration problem into the configuration matching problem. Based on our previous work [35], we provide a theoretical proof that the optimal reconfiguration planning problem of finding the least number of reconfiguration steps to transform between two configurations is NP-complete, i.e. a polynomial algorithm is unlikely to exist. Next, two different reconfiguration planning methods are proposed for different needs. The first one is called MDCOP, which has a theoretical guarantee to find the optimal graph-based reconfiguration plan. MDCOP works by converting a reconfiguration problem into a distributed constraint optimization problem (DCOP), and then solve it through existing DCOP algorithms. The second algorithm is called GreedyCM and it can find extremely near-optimal solutions in time that is polynomial to the size of the robot. Since the control of modular robots is inherently distributed, both of our methods are developed in distributed fashion, where the robot can efficiently identify the reconfiguration steps in a multi-modular-coordination way.

In the rest of the paper, Section 2 defines the problem of self-reconfiguration planning and Section 3 maps the problem into a configuration-matching problem. Section 4 analyzes the computational complexity of the optimal reconfiguration planning. Section 5 presents the two algorithms for reconfiguration planning in the graph-based representations. Experimental results are demonstrated in Section 6, and conclusions and future work discussion are given in Section 7.

2. Problem statement

2.1. The optimal reconfiguration planning problem

A modular robot is composed of a set of modules, and its configuration is an arrangement of the connectivity of the modules. The two elementary reconfiguration actions for rearranging connectivity are: (1) making some new connections, also called *attach* or *connect* actions, which are usually executed along with the motion of chains, and (2) disconnecting some current connections, also called *detach* actions or *disconnect* actions. We say that two configurations are *adjacent* if one can be transformed into the other by one reconfiguration action. Fig. 1 shows an example of the reconfiguration process using SuperBot [18–20]. The robot transform from configuration 1 to configuration 2 by an attach action, and then to configuration 3 through a detach action. Please note that only attach or detach actions, rather than the joint movements, will change the configuration. So, in our example, (a) and (b) have the same configuration even though they look different. So do (d) and (e).



(a) Snake robot. (b) L-shape robot.

Fig. 2. Example of complex configuration structure: two robots may have the same graphic topology but very different internal structures.

The self-reconfiguration planning problem is defined as: Given an arbitrary initial configuration and an arbitrary goal configuration, what are the reconfiguration actions to reconfigure from the initial to the goal? Specifically, what connections to make and what connections to disconnect to rearrange modules' connectivity? In the distributed fashion, it is the question of how the modules find out answers to the above questions in a multi-module-coordination way.

During the reconfiguration process, the reconfiguration actions consume the most time and energy, so it is a common practice to minimize the number of reconfiguration steps, i.e. the number of "connects" plus the number of "disconnects". Therefore, the optimal reconfiguration planning problem is to find the least number of reconfiguration steps to transform from the initial configuration into the goal configuration.

2.2. Research challenge

If we can represent all configurations as vertices of a graph and have an edge between two vertices if the corresponding configurations are adjacent, then the optimal reconfiguration problem could simply be formulated as a shortest path problem on such a graph. However, this is computationally intractable since the state space (configuration space) and action space (reconfiguring action space) grow exponentially with the number of modules.

In reality, there are many challenges for finding the minimal number of reconfiguration steps. The challenges to be addressed in this paper include:

Exponential configuration space: The configuration space is the set of all distinct configurations that a given number of modules may form. A robot with n modules can form arbitrary graph topology, so the number of configurations grows exponentially with the number of modules in the robot. It may not be possible to enumerate all configurations in an efficient way.

The complex structure of configuration: A robot's configuration is determined by not only the topological structure but also the connectors and the orientation of the connections. Thus, two configurations may be topologically the same, but because modules may be connected via different connectors they are in fact two very different configurations with different functions. For example, using two SuperBot modules, one can build a snake robot or an L-shape robot as shown in Fig. 2. These two robots have the same topological structure (if we view each module as a node and each connection as an edge) but they are in fact two very different configurations in function. Thus, the complex internal structure of a configuration presents a technical challenge for both formalization and analysis.

High dimension of reconfiguration action: Since a robot can bend its body through joint movements, any two modules in the robot that have free connectors may potentially be aligned and docked with each other. Any existing connections in the robot can also be potentially detached for reconfiguration. This high dimension of reconfiguration actions makes chain-type robots very flexible, but computationally very complex to represent and analyze.

Distributed coordination and local information only: Self-reconfigurable modular robots are distributed in nature. Each module has its own control unit, and they can only sense locally and communicate with their immediate neighbors. Due to this inherently distributed nature, it is not efficient and impractical for any single module to have a global view/control of all other modules. Thus, it is yet another challenge how these modules coordinate their local actions based on local information and communication to accomplish the final configuration.

When executing a reconfiguration plan, the connect actions require the alignment of the corresponding modules and their connectors, which involves the motion of joints of many modules. Feasibility of the physical execution of a reconfiguration step depends on the kinematics and flexibility of the modules, such as degree of freedom, motor forces, and so on. These may vary considerably from one hardware design to another. A reconfiguration plan that is executable on one type of module may require some extra "repair" steps on another type due to its kinematical insufficiency. To understand the inherent theoretical features of the optimal reconfiguration planning problem in general, we address the optimal reconfiguration planning problem as the connectivity rearrangement planning in terms of graph representation without consideration of the hardware limitations.

It is easy to know the number of modules in the initial configuration and the goal configuration. If the initial configuration has more modules than the goal configuration, redundant modules can be dropped. Conversely, the reconfiguration task is not possible due to the lack of required modules for the goal configuration. Without loss of generality, we assume that the number of modules in the initial and goal configuration is the same in the following.

3. Maximum configuration matching

Before presenting the complexity analysis and our reconfiguration planning algorithm, we would first describe our graph representation called C-Graph (Connector-Graph) for the robot's configuration and rephrase the optimal reconfiguration planning problem into the maximum configuration matching problem.

3.1. Graph-based configuration representation

C-Graph is the extension of a regular graph with differentiated connecting points and connection orientations. Modules are nodes and physical connections between modules are edges in C-Graph. Each node has a finite number of ports that are internally labeled corresponding to the connectors of a module. A connection between module u's connector Ci and module v's connector Cj with orientation ori corresponds to an edge $e =< Ci, ori, Cj >$ from module u's point of view, or $e =< Cj, ori, Ci >$ from module v's point of view. The value of edge e represents the connection type between nodes. Fig. 3 shows examples of C-Graph representation for SuperBot. A SuperBot has six genderless connectors ($C0$ through $C5$) on the six surfaces of the two linked cubes, and any connector in one module can connect to any connector of the other module in 4 different 90° rotations or orientations (ori

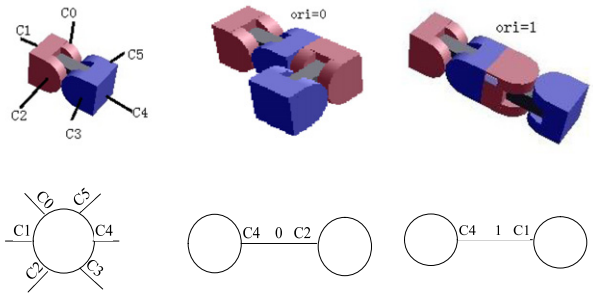


Fig. 3. Examples of C-Graph.

$= [0, 1, 2, 3]$). Throughout this paper, we will call C-Graph or configuration, node or module, edge or connection interchangeably. Without loss of generality, orientation of connections is omitted for the rest of this paper for easy illustration. When needed, we can always add *ori* field back into edge e to compare the connection type.

3.2. Maximum configuration matching problem

Given an arbitrary initial and a goal configuration, the more common parts that can be detected between them, the less change is needed to reconfigure from one to another, and the less reconfiguration steps are needed. Therefore, we rephrase the graph-based optimal reconfiguration planning problem into the *maximum configuration matching* problem.

The maximum configuration matching problem is defined as: Given the initial configuration I and the goal configuration G, how to match the nodes between I and G so that the number of matched (unmatched) edges is maximum (minimum). In a distributed version, it is the question of how each module in I finds its matched node in G, so that the number of matched edges will be maximum. Here, the matched edge is defined as:

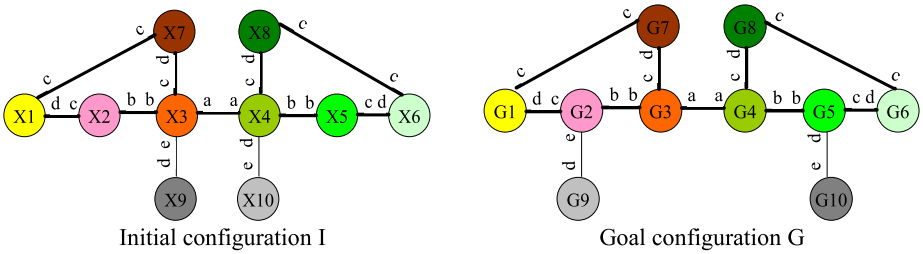
- *Matched edges*: Given the initial configuration I and the goal configuration G, where $u \in I$ and $v \in I$ are connected by edge e, $u' \in G$ and $v' \in G$ is connected by edge e'. Edge e and e' are called *matched edges* if the node matching between I and G is $\{(u, u'), (v, v')\}$, and we have $e = e'$.

For example, in Fig. 4, we use coloring to show the node matching. If the nodes are matched as $\{(X1, G1), (X2, G2), \ldots\}$ as shown in 4(a), then there are 9 matched edges (bold edges), and two unmatched edges (non-bold edges). This looks like a good configuration matching. However if the nodes are matched as $\{(X1, G7), (X2, G3), \ldots, \}$ as shown in 4(b), then there are 10 matched edges and only 1 unmatched edge. Since I and G are not isomorphic (because I has maximum degree of 4 and G has maximum degree of 3), and 4(b) has only 1 unmatched edge between I and G, we can say that the configuration matching in 4(b) achieves the maximum matching.

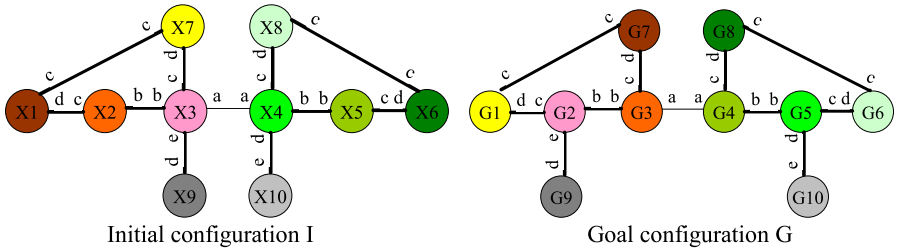
The graph-based reconfiguration plan is straightforward after finding the node matching and unmatched edges between I and G. The reconfiguration actions are: all unmatched edges in the goal configurations are to be connected, and all unmatched edges in the initial configuration are to be disconnected. For example, if we match the above initial and goal configuration as in 4(a), the graph based reconfiguration plan would be attaching $X2(e)$–$X10(d)$ and $X5(d)$–$X9(e)$ (corresponds to the unmatched edges $G2(e)$–$G9(d)$ and $G5(d)$–$G10(e)$ in G) and detaching $X4(d)$–$X10(e)$ and $X3(e)$–$X9(d)$ (corresponds to the two unmatched edges in I). If we match as in 4(b), the graph-based reconfiguration plan is: attach $X2$ and $X5$ by $X2(a)$–$X5(a)$ (corresponds to the unmatched edge $G3(a)$–$G4(a)$ in G) and detach $X3(a)$–$X4(a)$ (corresponds to the unmatched edge $X3(a)$–$X4(a)$ in I). In distributed implementation, after informed the configuration matching, each module can look at all of its connections and identify the unmatched edges as to be disconnected, and also find out the unmatched edges of its matched node as to be connected. Obviously, the number of reconfiguration steps is equal to the total number of unmatched edges in I and G. Therefore, achieving the maximum matched edges (or minimum unmatched edges) will generates an optimal graph-based reconfiguration plan with the least reconfiguration steps.

3.3. Relationship with graph similarity

Basically, configuration matching is to find the edges to be added and the edges to be deleted to transform from I to G. This drives us to relate the graph-based reconfiguration planning problem to the graph edit distance problem in the area of graph similarity. The aim of graph edit distance is to find the minimum



(a) An instance of configuration matching that is not maximum.



(b) An instance of maximum configuration matching.

Fig. 4. Two instances of configuration matching.

number of prescribed edit operations (nodes and edge additions, deletions and substitutions) that are required to transform one graph into the other. It has been widely studied with different techniques, such as the exact algorithm of A* tree search, and the approximation algorithms of continuous optimization, spectral neutral network, quadratic programming, etc. [36].

Although the graph edit-distance problem is similar to the optimal reconfiguration planning problem at first glance, it is challenging to apply its techniques here due to two reasons. (1) As we discussed before, a modular robot's configuration is decided by the connection type as well as the graph topology, and a regular graph is not enough to represent the configuration. This makes the comparison of initial and goal configurations more complicated. (2) In graph edit distance, a new node can be inserted freely, or be changed freely into another type of node. On the contrary, in our problem, a robot module cannot "jump" or "teleport" into another location in the robot, or become another type of module magically. So, our focus is to rearrange the connectivity of a robot's structure, while graph edit distance is focused on inserting/deleting/replacing the mismatched nodes.

4. Computational complexity of optimal reconfiguration planning

It is commonly agreed that optimal reconfiguration planning is computationally intractable, but concrete support for this belief is lacking. One widely used explanation [26,30] is that the configuration space is exponential, but that alone is not enough to show that the problem cannot be solved efficiently in polynomial time because many optimization problems can actually be solved efficiently even if their search space is exponential. For example, the shortest-path problem for graphs without negative cycles can be solved efficiently, while the number of paths between two nodes can grow exponentially with the number of nodes in the graph. Here, we investigate the computational complexity of the optimal reconfiguration planning problem in depth and provide a formal proof that the problem is indeed NP-complete.

4.1. Short review of NP-completeness

A problem X is defined as NP-complete if:

- X is in NP: X can be shown to be in NP by demonstrating that a candidate solution to X can be verified in polynomial time.
- X is NP-hard: X is NP-hard if there is an already known NP-complete problem Y such that Y is *polynomial reducible* to X, and we write $Y <_p X$. $Y <_p X$ means that if we have a black box capable of solving X, then an arbitrary instance of problem Y could be solved by first reducing to X using a polynomial number of standard steps, and then by a polynomial number of calls to that black box that solves X. So, X is at least as hard as Y with respect to polynomial time.

Usually, if X is an optimization problem, it is always reformulated into a decision problem to explore its reducibility from Y. An optimization problem is NP-complete if its corresponding decision problem is NP-complete [37]. The general strategy to prove $Y <_p X$ is: given an arbitrary instance S_y of Y, and show how to construct, in polynomial time, an instance S_x of problem X, such that the answer to the question whether S_x is a "yes" instance of X if and only if the answer to the question whether S_y is a "yes" instance of Y.

4.2. NP-completeness of optimal reconfiguration planning problem

Intuitively, less reconfiguration steps are needed when the differences between the initial and goal configurations are low, and more reconfiguration steps are needed when the difference is high. This drives us to relate the reconfiguration planning problem to the graph similarity and matching problem. As we know, many graph similarity problems, such as maximum common subgraph isomorphism etc., are NP-complete, but they become polynomial-time solvable when the underlying graphs are acyclic. This leads us to an expectation that the optimal reconfiguration planning may also be solved efficiently when the initial and goal configurations are acyclic. We refer to this problem as the *Acyclic Optimal Reconfiguration* problem. Unfortunately, based on our study, we find that even for acyclic configurations, the optimal reconfiguration problem is still NP-complete.

Lemma 1. *Even when the initial and goal configurations are acyclic, optimal reconfiguration planning is still NP-complete.*

Note that when both the initial configuration and goal configuration are acyclic, the number of "connect" actions must be equal to that of "disconnect" actions, otherwise the initial and goal configurations will have different number of edges, which is not true given that they have the same number of modules. Thus, the decision version of the *Acyclic Optimal Reconfiguration* problem is reformulated as:

> *Given acyclic configurations I and G, and a given integer n, whether there exists a reconfiguration plan with at most 2n reconfiguration steps, i.e. n "connect" and n "disconnect" actions?*

The proof that this is an NP-complete problem is as follows:

Step 1: Show that it is in NP. Given any reconfiguration plan with at most $2n$ steps, it is obvious that we could check in polynomial time whether I can be transformed into G.

Step 2: Prove that it is NP-hard by reducing from the 3-PARTITION problem, and show that 3-PARTITION $<_p$ ACYCLIC OPTIMAL RECONFIGURATION.

The 3-PARTITION problem is defined as:

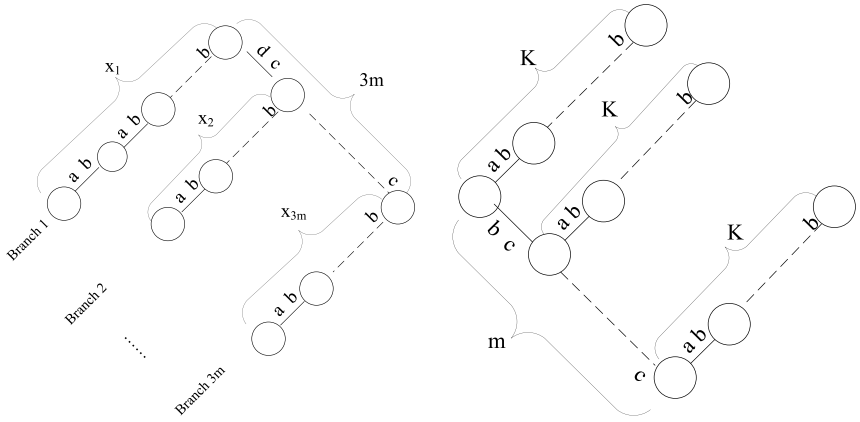
> *3-PARTITION: Given a set of positive integers with* $3m$ *elements,* $S = \{X_1, \ldots, X_{3m}\}$, *where* $\sum_{X_i \in S} X_i = mK$, *and each element* X_i *satisfy* $K/4 < X_i < K/2$ $(i = 1, \ldots, 3m)$. *Can S be partitioned into m disjoint subsets* $S_1, \ldots, S_m$ *such that the sum of the numbers in each subset is the same and equal to K?*

For an arbitrary given instance $S = \{X_1, \ldots, X_{3m}\}$ in a 3-PARTITION problem, we construct an initial configuration I and a goal configuration G as shown in Fig. 5. The connectors are labeled alphabetically as *abcd*. In the initial configuration I, we start with $3m$ branches where each branch i has X_i number of nodes connected in a line by edge $\langle a, b\rangle$. Then, we connect these line branches consecutively by their rightmost nodes via edge $\langle d, c\rangle$. In the goal configuration G, there are m equal-length branches, where each branch has $K(K = \frac{\sum_{X_i \in S} X_i}{m})$ nodes connected in a line by edge $\langle a, b\rangle$. These m branches are connected consecutively by their leftmost nodes via edge $\langle b, c\rangle$.

To be precise, it actually takes pseudo-polynomial instead of polynomial time to construct I and G as above, since the configuration of I and G has size $\sum_{X_i \in S} X_i$, and this is polynomial in the magnitude of the numbers in S, but not polynomial in the size of the representation of S. However, this does not affect our proof of NP-hardness, because the 3-PARTITION problem is NP-complete in the strong sense in that it is NP-complete even when all of the integers in S are bounded by a polynomial in the size of S [37].

Now, to prove 3-PARTITION $<_p$ Acyclic Optimal Reconfiguration, we will show that an arbitrary instance of set S is solvable for the 3-PARTITION problem if and only if the correspondingly constructed I can be transformed to G in at most $6m - 2$ steps.

Soundness. Let S be an arbitrary instance in the 3-PARTITION problem, and initial configuration I and goal configurations G are constructed as above. If the 3-PARTITION problem with instance S has a solution, then I can be transformed to G in at most $6m - 2$ steps.



(a) Initial configuration I. (b) Goal configuration G.

Fig. 5. Constructed instance of initial and goal configurations.

Table 1
The number of edges of each type in I and G.

	# of edge $\langle a, b\rangle$	# of edge $\langle b, c\rangle$	# of edge $\langle d, c\rangle$
Configuration I	$\sum_1^{3m} X_i - 3m$	0	$3m-1$
Configuration G	$mK - m$	$m-1$	0
Configuration G − Configuration I	$(mK-m)-(\sum_1^{3m} X_i - 3m) = 2m$	$(m-1)-0 = m-1$	$0-(3m-1) = -(3m-1)$

Proof. It has been proved that if the 3-PARTITION problem with instance S has a solution, then S can partitioned into m disjoint subsets $S_1, \ldots, S_m$, and each subset S_j must be a triple of exactly three elements with sum K [37]. With a given solution to the 3-PARTITION, we have configuration matching as: for each triple $S_i = \{X_r, X_s, X_t\}$, since $X_r + X_s + X_t = K$, we match the corresponding branch r, s and t on I consecutively to any available length K branch in G, i.e. out of all the nodes in the matched length-K branch of G, the first X_r nodes are matched to branch r in I, and the next X_s nodes are matched to branch s in I, and the last X_t nodes are matched to branch t in I. Using this configuration matching, every length K branch in G can be matched, and every such branch has exactly 2 unmatched edges (i.e. the edge between the length X_r chain and the next length X_j chain, and the edge between the length X_j chain and the length X_r chain), so there are $2m$ unmatched edges of $\langle a, b\rangle$ in total. In addition, the $m-1$ edges $\langle b, c\rangle$ in G are all unmatched, so the total unmatched edges are $3m-1$ in G. In I, the $3m-1$ edges of $\langle d, c\rangle$ are all unmatched. In other words, I can be transformed into G by connecting all the $3m-1$ unmatched edges in G and disconnecting all the $3m-1$ unmatched edges in I, that is $6m-2$ reconfiguration steps in total. This completes the proof of soundness.

Completeness. Let S be an arbitrary instance in the 3-PARTITION problem, and the initial configuration I and goal configurations G are constructed as above. If I can be transformed to G in at most $6m-2$ steps, then the 3-PARTITION problem with instance S has a solution.

Proof. The main idea of proving completeness is: if it can be shown that the reconfiguration process must consist of connecting the $3m$ line branches in I into m length-K branches without breaking any edges inside the $3m$ line branches in I, then for each "connecting" action between a branch i and a branch j, we put integer X_i and X_j into the same subset. This defines a partition of set S into m subsets with equal sum K.

Let us first compare the number of edges in each type between I and G. Take the edge of type $\langle a, b\rangle$ as an example. In I, each branch i has $X_i - 1$ edges of $\langle a, b\rangle$, and thus $\sum_1^{3m} X_i - 3m$ edges of $\langle a, b\rangle$ in total for $3m$ line branches. Similarly, we can get the number of edges of all types in I and G, and their difference as shown in Table 1.

Based on the values in the third row of Table 1, it can be seen that to reconfigure I into G, it is a must-have to make $2m$ new connections of $\langle a, b\rangle$, $m-1$ new connections of $\langle b, c\rangle$, and disconnect $3m-1$ existing connections of $\langle d, c\rangle$. This is $(2m)+(m-1)+(3m-1) = 6m-2$ reconfiguration steps in total. Therefore, if there is a reconfiguration plan from I to G with at most $6m-2$ steps, it must be the actions stated above. By examining the edges in I, we can find that the disconnect action of $\langle d, c\rangle$ will disconnect I into $3m$ line branches without breaking the $\langle a, b\rangle$ edges within any one of the $3m$ line branches. Also, since the two-end modules in each line branch are the only ones that have free connector a or b, the connect action of $\langle a, b\rangle$ must be that of connecting the two branches into one without interfering with the inner modules. Since the goal configuration has m branches, where each branch has K nodes connected in a line by $\langle a, b\rangle$, the $2m$ connect actions must produce m length-K branches. The goal configuration is thus reached by connecting these m branches consecutively by the connect actions of $\langle b, c\rangle$.

For each connect action $\langle a, b\rangle$ between branch i and branch j, we put the integer X_i and X_j into the same subsets. Corresponding to the m length-K branches in G, we end up with m subsets, with the sum of the numbers in each set equal to K. Namely, the 3-PARTITION problem with instance S has a solution. This completes the proof of completeness.

Lemma 2. *Optimal reconfiguration planning of finding the least number of reconfiguration steps for chain-type modular robots is NP-complete.*

Since the Acyclic Optimal Reconfiguration problem is a special case of the optimal reconfiguration planning problem for all configurations, Lemma 2 is straightforward.

5. Reconfiguration planning algorithms

Based on the above complexity analysis, we know that a polynomial algorithm for optimal reconfiguration planning is unlikely to exist. In other words, we have a tradeoff between maximum

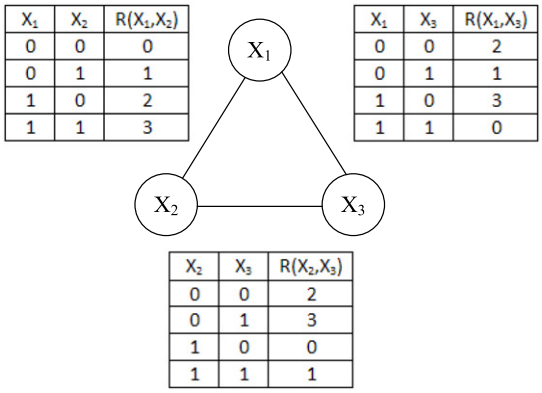


Fig. 6. An instance of DCOP.

configuration matching and the polynomial running time. To satisfy the needs in different situations, we proposed two configuration matching algorithms. One is called MDCOP, by which modules can coordinate and find the maximum matched edges. The other is called GreedyCM, which settles with near-optimal results but runs in polynomial time. In both algorithms, C-Graph of the goal configuration is informed to every module in the initial configuration.

5.1. MDCOP

In classic graph theory, the majority of graph matching techniques are based on the backtracking paradigm, and many backtracking optimization algorithms are also done in the context of constraint optimization problems. This commonality inspired us to reduce the distributed configuration matching problem to a distributed constraint optimization problems (DCOP), and we call it the Mapping_to_DCOP (MDCOP) algorithm. By converting our problem into DCOP, we gain direct access to the rich research findings in the DCOP area instead of inventing new algorithms from scratch.

5.1.1. Overview of DCOP

A DCOP is a problem in which a group of agents must distributedly choose values such that the cost of a set of constraints over the variables is minimized. It can be defined as a $\langle X, D, R\rangle$ where

- $X = \{X_1, \ldots, X_n\}$ is a set of agents.
- $D = \{D_1, \ldots, D_n\}$ is a set of finite domains, where D_i denotes the set of possible values of X_i.
- $R = \{R_1, \ldots, R_m\}$ is a set of binary constraints $R_k : D_i \times D_j \to N$ defines the cost for each possible combination of values (X_i, X_j).

Each agent X_i has control of its value and knowledge of its domain D_i. The objective of DCOP is to have every agent X_i assigned a value so that the total cost R is minimized. A DCOP problem is always visualized as a *constraint graph* where the vertices are the agents and the edges are the constraints. The cost is described in tables, and we call it the *constraint table* or *cost table*. Fig. 6 shows an example DCOP with three agents (X_1, X_2, X_3). All agents have the same domain $D = \{0, 1\}$. The solutions to it are $X_1 = 0, X_2 = 1, X_3 = 0$, or $X_1 = 0, X_2 = 1, X_3 = 1$, with minimized cost $R(X_1, X_2) + R(X_2, X_3) + R(X_1, X_3) = 3$.

5.1.2. MDCOP algorithm

The analogy of distributed configuration matching to DCOP is straightforward if we view each module X_i in I as an agent and its value is its matched node in G. Each module's domain is the set of all the nodes in G. The cost between two connected modules is 0 if the edge between them is a matched edge, otherwise 1. So minimizing the total cost in DCOP is equal to minimizing the number of unmatched edges in I, i.e. the number of detach actions in the reconfiguration. Since

# of attach actions

$$= \#\text{ of edges } (G) - \#\text{ of edges } (I) + \#\text{ of detach actions},$$

we can see that the number of attach actions is also minimized automatically. Therefore, the DOCP solution will generate the least reconfiguration steps.

To ensure that no two modules select the same node in G, a binary constraint is needed between any pair of modules. This makes the constraint graph in DCOP to be fully connected, i.e. every module needs a direct access to all other modules' value. This decreases the performance quite a lot, and is impractical for implementation. To improve this, we introduce the concept of the *set of candidate mates*. For each module $u \in I$, we say that node $v \in G$ belongs to its set of candidate mates if at least one edge incident to u and at least one edge incident to v have the same connection type. Obviously, u must be matched to a node in its set of candidate mates so as to have at least one of its incident edges as a matched edge. For all other nodes in G that are outside of u's *set of candidate mates*, it makes no difference to u which node it will match. In other words, two modules whose sets of candidate mates are disjoint will have no potential conflict to choose the same node in G, and binary constraint between them is not needed.

More precisely, the way to map a configuration matching problem into DCOP is:

- Step 1: Each module X_i finds its set of candidate mates, and sets its domain $D_i = \{\text{set of candidate mates of } X_i, \varnothing\}$. $\varnothing$ is a wild character for all other nodes in G.
- Step 2: For each pair of X_i and X_j with $D_i \cap D_j \neq \{\varnothing\}$, add binary constraint "if $X_i = X_j \neq \varnothing$, $R(X_i, X_j) = \infty$" to X_i and X_j.
- Step 3: For any pair of connected modules X_i and X_j in I, if $\exists$ node matching $\{(X_i, V_i), (X_j, V_j)\}$ so that the edge between X_i and X_j is a matched edge, then add binary constraint "if $X_i = V_i$, $X_j = V_j$, then $R(X_i, X_j) = 0$" to both X_i and X_j. For all other values that X_i and X_j may choose, add "$R(X_i, X_j) = 1$" to X_i and X_j.

Step 2 ensures that no two modules will match the same nodes in G. Step 3 defines the constraint cost whose minimization corresponds to maximizing the number of matched edges. In the DCOP solution, if $X_i = V_i$, we will have (X_i, V_i) in the configuration matching. If $X_i = \varnothing$, it means that all adjacent edges of X_i will not be matched edges no matter which free node X_i chooses, so X_i will choose an arbitrary free node in G.

To illustrate the process, let us go through the example in Fig. 4 to show how to construct a DCOP in detail.

- Step 1: Taking module X_3 as an example. It has four incident edges as $\langle a, a\rangle$, $\langle c, d\rangle$, $\langle b, b\rangle$, and $\langle e, d\rangle$. In the goal configuration G, node $G3$ and $G4$ incident edges of $\langle a, a\rangle$, $\langle c, d\rangle$ and $\langle b, b\rangle$; node $G2$ has incident edges of $\langle c, d\rangle$, $\langle e, d\rangle$, $\langle b, b\rangle$; node $G5$ has incident edges of $\langle b, b\rangle$ and $\langle c, d\rangle$, node $G10$ has incident edge of $\langle e, d\rangle$. Other nodes in G have no incident edge equal to any of $\langle a, a\rangle$, $\langle c, d\rangle$, $\langle b, b\rangle$, and $\langle e, d\rangle$. Therefore the domain of X_3 is $D_3 = \{G2, G3, G4, G5, G10, \varnothing\}$. Similarly, all the modules' domains of the initial configuration are
  $D_1 = D_6 = D_7 = D_8 = \{G1, G6, G7, G8, \varnothing\}$
  $D_2 = D_5 = \{G2, G3, G4, G5, \varnothing\}$
  $D_3 = \{G2, G3, G4, G5, G10, \varnothing\}$
  $D_4 = \{G2, G3, G4, G5, G9, \varnothing\}$
  $D_9 = \{G5, G9, \varnothing\}$
  $D_{10} = \{G2, G10, \varnothing\}$.

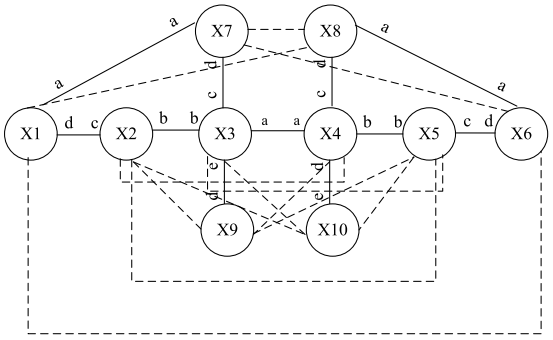


Fig. 7. Constraints graph for reconfiguration from I to G in Fig. 4.

- Step 2: Based on the domain sets defined above, we get that X_1, X_7, X_8, X_6 have the possibility to choose the same nodes in G, so do X_2, X_3, X_4, X_5, X_9, and $X_2, X_3, X_4, X_5, X_{10}$. Dotted lines in Fig. 7 show the added binary constraints between these modules. For all modules X_i and X_j connected by solid or dotted line, we have the binary constraint of "if $X_i = X_j \neq \varnothing$, $R(X_i, X_j) = \infty$".
- Step 3: Take the connection $X_1(d)$–$X_2(c)$ as an example. Since in G, node $G1$ and $G2$, node $G7$ and $G3$, $G8$ and $G4$, $G6$ and $G5$ are also connected by edge $\langle d, c\rangle$, so we get that the binary constraint between module X_1 and X_2 as: "$R(X_1, X_2) = 0$ if $X_1 = G1\, X_2 = G2$, or $X_1 = G7\, X_2 = G3$, or $X_1 = G8\, X_2 = G4$, or $X_1 = G6\, X_2 = G5$; otherwise, $R(X_1, X_2) = 1$".

Ultimately, the *constraint tables* are given in Fig. 8.

Solution to the DCOP is $X_1 = G7, X_2 = G3, X_3 = G2, X_4 = G5, X_5 = G4, X_6 = G8, X_7 = G1, X_8 = G6, X_9 = G9, X_{10} = G10$ or $X_1 = G8, X_2 = G4, X_3 = G5, X_4 = G2, X_5 = G3, X_6 = G7, X_7 = G6, X_8 = G1, X_9 = G10, X_{10} = G9$, with minimized cost $R = 1$. The cost $R = 1$ is due to the constraint between X_3 and X_4, i.e. $R(X3, X4) = 1$ when $X3 = G2$ and $X4 = G5$. Maximum configuration matching for the initial and goal configurations in Fig. 4 is thus achieved.

After converting the configuration matching problem into the DCOP problem, all modules in I can coordinate with each other to find its own matching by existing distributed algorithms for solving DCOP, including NCBB [38], DPOP [39], OptAPO [40], Adopt [41] etc. If $X_i = G_j$ in the DCOP solution, then we have module X_i matched node G_j in G. If $X_i = \varnothing$, we establish a matching between X_i and an arbitrary free node in G. This reaches the solution of maximum configuration matching, and thus the least number of reconfiguration actions.

It is easy to see that the time complexity of mapping to DCOP is polynomial. Assume initial configuration I has N nodes and $E1$ edges, and goal configuration G has N nodes and $E2$ edges. Step 1 takes $O(E1 * E2)$ to establish the domain sets, step 2 takes $O(N^2)$ to establish the constraints between modules with intersecting domain sets, and step 3 takes $O(E1 * E2)$ to establish binary constraints between connected modules. So the total time to convert our problem into DCOP is $O(E1 * E2 + N^2) = O(E1 * E2)$. Compared with the exponential time/communication cost needed to solve DCOP, we can see that solving the NP-complete DCOP problem takes the most resources in the MDCOP algorithm.

5.2. GreedyCM

5.2.1. GreedyCM algorithm

Solving MDCOP needs an exponential number of messages or exponential memory complexity. It is expensive when the robot is large. In most situations, we may settle for a solution close to maximum matched edges, but can be reached in polynomial time. In this section, we proposed a polynomial algorithm called *GreedyCM* (Greedy configuration matching) that runs the configuration matching by iteratively extracting the maximum common edge sub-configuration (*MCESC*) between I and G.

Before describing the GreedyCM algorithm, some terms are defined first.

- *Maximum common edge sub-configuration* (*MCESC*): Similar to the definition of maximum common edge subgraph in graph theory, a maximum common edge sub-configuration (*MCESC*) is a common sub-C-Graph between I and G with the largest number of edges.
- $u \leftrightarrow v$ *rooted MCESC:* Given the condition that node $u \in I$ must be matched to $v \in G$, the maximum common edge sub-configuration that can be achieved is called $u \leftrightarrow v$ *rooted MCESC*. Obviously, for all (u', v') in $u \leftrightarrow v$ rooted MCESC, $u' \leftrightarrow v'$ rooted MCESC is equal to $u \leftrightarrow v$ rooted MCESC.
- *u-rooted MCESC*: For a given $u \in I$, among all the $u \leftrightarrow v$ rooted MCESC ($\forall v \in G$), the one with the maximum number of edges is called *u-rooted MCESC*. Obviously, among all u-rooted MCESC ($\forall u \in I$), the one with maximum edges is the MCESC between I and G.

Since MCESC is the common sub-configuration with maximum number of matched edges, it seems to be the best choice that can make the most progress towards the goal of maximum matched edges. Accordingly, GreedyCM finds the matched edges between I and G by iteratively extracting the MCESC between

i=1, j=2 or i=7 j=3, or i=8, j=4, or i=6 j=5

Xi	Xj	R(xi,xj)
G1	G2	0
G7	G3	0
G8	G4	0
G6	G5	0
Xi=Xj≠Ø		∞
Other		1

i=1, j=7 or i=6, j=8

Xi	Xj	R(xi,xj)
G1	G7	0
G6	G8	0
G7	G1	0
G8	G6	0
Xi=Xj≠Ø		∞
Other		1

i=2, j=3 or i=4, j=5

Xi	Xj	R(xi,xj)
G2	G3	0
G4	G5	0
G3	G2	0
G5	G4	0
Xi=Xj≠Ø		∞
Other		1

i=3, j=4

Xi	Xj	R(xi,xj)
G3	G4	0
G4	G3	0
Xi=Xj≠Ø		∞
Other		1

i=3, j=9 or i=10, j=4

Xi	Xj	R(xi,xj)
G2	G9	0
G10	G5	0
Xi=Xj≠Ø		∞
Other		1

Xi and Xj are connected or have dotted links

Xi	Xj	R(xi,xj)
Xi=Xj≠Ø		∞
Other		1

Fig. 8. Constraint tables for the robot of Fig. 4.

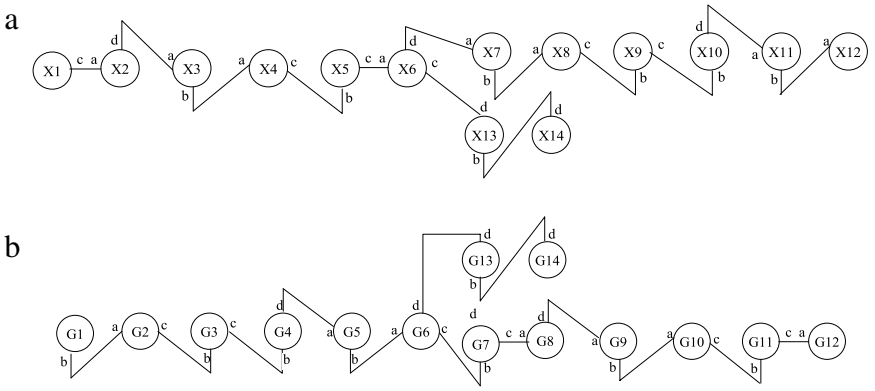


Fig. 9. An instance of (a) Initial C-Graph I and (b) Goal C-Graph G.

```
MCESC_UV (u,v)
// Output u↔v rooted MCESC for u ∈ I , v ∈ G
{
   Mark node u and v as visited, and add node (u, v) into node matching set
   For each edge <Ci, Cj> between u and u' in I
   {
      If v is connected to v' with edge <Ci, Cj> in G
      {
            If  u' and v' have not been visited,
                  MCESC_UV (u',v');
      }
   }
    return the node matching set
}
```

Fig. 10. Polynomial algorithm to find $u \leftrightarrow v$ rooted MCESC.

$I \setminus$ {matched nodes and their incident edges} and $G \setminus$ {matched nodes and their incident edges}. Every time a MCESC is found, corresponding nodes in I and G are matched. The algorithm stops when all the nodes in I and G are matched.

For example, in Fig. 9 the iteratively extracted MCESCs are

(1) {($X2$, $G4$), ($X3$, $G5$), ($X4$, $G6$), ($X5$, $G7$), ($X6$, $G8$), ($X7$, $G9$), ($X8$, $G10$), ($X9$, $G11$)}
(2) {($X13$, $G13$), ($X14$, $G14$)}
(3) {($X11$, $G1$), ($X12$, $G2$)}
(4) {($X1$, $G3$)}
(5) {($X10$, $G12$)}.

All the edges inside the MCESCs are matched edges, and other edges are unmatched. So in the end four unmatched edges are found, which are $X1$–$X2$, $X6$–$X13$, $X9$–$X10$, $X10$–$X11$ in I and $G2$–$G3$, $G3$–$G4$, $G6$–$G13$, $G11$–$G12$ in G.

Finding MCESC is an NP-complete problem in graph theory. However, due to the fact that C-Graphs have bounded degree and distinct connecting ports, the number of ways to represent two isomorphic configurations drops dramatically. Permutation of the neighbors is also not allowed anymore. So $u \leftrightarrow v$ rooted MCESC can be found easily in polynomial time by performing a simultaneous traversal of the C-Graphs of $I \setminus$ {matched nodes} and $G \setminus$ {matched nodes starting from u in I and v in G}. MCESC can thus be easily found by comparing all. So $u \leftrightarrow v$ rooted MCESCs and returning the largest one. The algorithm of finding $u - v$ rooted MCESCs is shown in Fig. 10. MCESC can thus be achieved by comparing all $u - v$ rooted MCESC. Note that if u' and v' are matched in a $u \leftrightarrow v$ rooted MCESC, since $u' \leftrightarrow v'$ rooted MCESC must be equal to $u \leftrightarrow v$ rooted MCESC, it is unnecessary to calculate $u' \leftrightarrow v'$ rooted MCESC anymore.

Fig. 11 shows an example of finding $x1 - 1$ rooted MCESC when I and G have cycles. Initially, $MCESC\_UV(X1, 1)$ are initiated and nodes $(x1, 1)$ are matched. For the common incident

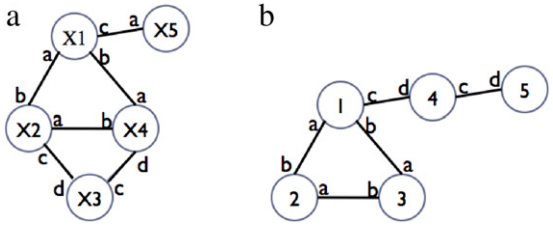


Fig. 11. An instance of (a) Initial C-Graph and (b) Goal C-Graph G with cycle.

edge $\langle a, b\rangle$ and $\langle b, a\rangle$ between $X1$ and 1, it will recursively call $MCESC\_UV(x2, 2)$ first, and then $MCESC\_UV(x4, 3)$ if $X4$ and 3 are not visited yet. $MCESC\_UV(x2, 2)$ will match $(x2, 2)$ and also call $MCESC\_UV(x4, 3)$ due to the common incident edge $\langle a, b\rangle$ for $X2$ and 2. So $(X4, 3)$ will be marked as visited, and not be called again by $MCESC\_UV(X1, 1)$. In the end, $x1 - 1$ rooted MCESC is $\{(X1, 1), (X2, 2), (X4, 3)\}$.

Assume C-Graph I has N nodes and $E1$ edges, and G has N nodes and $E2$ edges. During each iteration, any pair of edges between I and G will only be compared once in finding the MCESC, so the time complexity is $O(E1 * E2)$. After one cycle of MCESC, at least one module will be excluded from I and G, so there are at most N iterations. So the total computational complexity for Greedy GM is $O(N * E1 * E2)$. The time complexity is polynomial to the size of robot, and thus scalable to a robot with many modules.

5.2.2. Distributed implementation

In the greedy algorithm, matching between I and G is inherently ordered, therefore the condition that no two modules are matched to the same node can easily be satisfied. For distributed implementation where all modules simultaneously find their own matched node in G, this requirement brings up several issues

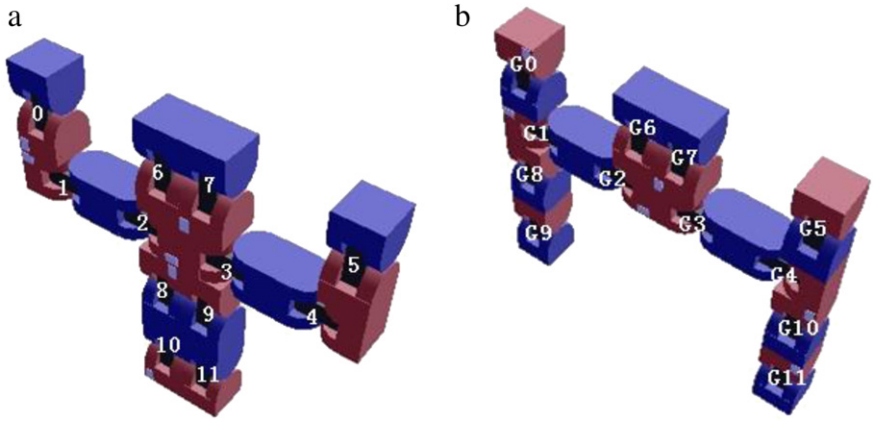


Fig. 12. (a) Biped, (b) StileWalker.

- *Parallelism versus serialization*: On the one hand, we want modules to do the matching as parallel as possible for fast execution. On the other hand, the greedy algorithm has an embedded order in that the matching in $u - v$ rooted MCESC with more matched edges always takes precedence over that with less matched edges. We do not want to waste resources to backtrack and undo the undesirable matching done previously by smaller rooted MCESCs.
- *Data coherency and mutual exclusion*: In the concurrent matching of multi-modules, each module in I must keep a consistent view and achieve most current information of which nodes in G have been taken. If two modules try to match to the same nodes, their attempts have to be executed in a mutually exclusive manner.

To solve these issues, we propose to have the leader module establish a partial order among the u-rooted MCESCs. Information of the matched nodes is passed to the leftover unmatched modules to satisfy data coherency and consistency.

In the beginning, every module $u \in I$ is initiated to find the $u-v$ rooted MCESC for all $v \in G$, and will send the largest one, u-rooted MCESC to the leader. The leader will sort all the rooted MCESCs in a first-come–first-serve basis. Two u-rooted MCESCs with shared modules in I or G are ordered according to their number of edges. Disjoint u-rooted MCESCs have no precedence relationship. This established a partial order among all the u-rooted MCESCs, and the ones ranking foremost in the partial order will be executed. All matched nodes of G are broadcast to the leftover unmatched modules. The leftover modules will reiterate the above cycle again until every node in G is matched.

For illustration, let us go through the example in Fig. 9. In the first cycle, the $u - v$ rooted MCESCs are:

Module $X1$:

$X1 \leftrightarrow G7$ rooted MCESC: $M = \{(X1, G7), \ldots (X6, G12)\}$, with 5 edges
$X1 \leftrightarrow G11$ rooted MCESC: $M = \{(X1, G11), (X2, G12)\}$, with 7 edges
$X1 \leftrightarrow Gi$ rooted MCESC ($Gi \neq G7$ or $G11$): $M = \{(X1, Gi)\}$, no edge

Module $X2$:

$X2 \leftrightarrow G4$ rooted MCESC: $M = \{(X2, G4), \ldots, (X9, G11)\}$, with 3 edges
$X2 \leftrightarrow G8$ rooted MCESC: equal to $X1 \leftrightarrow G7$ rooted MCESC
$X2 \leftrightarrow Gi$ rooted MCESC ($Gi \neq G4$ or $G8$): $M = \{(X2, Gi)\}$, no edge

Module $X3$:

…

The rooted $u - v$ MCESCs sent to leader are:

1. $\{(X1, G7), (X2, G8), (X3, G9), (X4, G10), (X5, G11), (X6, G12)\}$
2. $\{(X2, G4), (X3, G5), (X4, G6), (X5, G7), (X6, G8), (X7, G9), (X8, G10), (X9, G11)\}$
3. $\{(X7, G1), (X8, G2), (X9, G3), (X10, G4), (X11, G5), (X12, G6)\}$
4. $\{(X13, G13), (X14, G14)\}$.

Depending on the number of edges in each rooted MCESCs, the partial order will be "$2 > 1 = 3, 4$". Therefore, in the first cycle, the matching in 2 and 4 is to be done. Similarly, the second cycle will complete the matching of $\{(X11, G1), (X12, G2)\}$, $\{(X1, G3)\}$, $\{(X10, G12)\}$ or $\{(X11, G1), (X12, G2)\}\{(X1, G12)\}$, $\{(X10, G3)\}$ (depending on how to break the tie). All modules find their matching in two cycles. Our matching results lead to four unmatched edges as $X1$–$X2$, $X6$–$X13$, $X9$–$X10$, $X10$–$X11$ in I and $G2$–$G3$, $G3$–$G4$, $G6$–$G13$, $G11$–$G12$ in G.

6. Experimental results

6.1. Implementation on SuperBot

Using SuperBot, we have implemented the reconfiguration example of biped ↔ stiltWalker to show the application of configuration matching on a real robot system. Fig. 12 depicts their shape. (Please note that modules 6 ($G6$) and 7 ($G7$), 8 and 9, 10 and 11, are touching each other but not connected in Fig. 12(a) and (b)).

Under GreedyCM, the first MCESC found is the sub-configuration composed of modules 0, 1, 2, 3, 4, 5, 6, 7 in the initial and modules $G0$, $G1$, $G2$, $G3$, $G4$, $G5$, $G6$, $G7$ in the goal. The second MCESC is composed of modules 8, 10 in initial and $G8$, $G9$ in goal, and the third MCESC is composed of modules 9, 11 in initial and $G10$, $G11$ in goal. Therefore, the unmatched edges are the connection between modules 2 and 8, modules 3 and 9 in initial, modules $G1$ and $G8$, and modules $G4$ and $G10$ in goal. GreedyCM will generate graph-based reconfiguration actions as: attach modules 1 and 10, detach modules 2 and 8, attach modules 11 and 4, and detach modules 3 and 9. There 4 reconfiguration actions are all executable by SuperBot, and Fig. 13 shows the parallel execution of these 4 reconfiguration actions. Video of the reconfiguration process is shown in http://www.isi.edu/robots/reconfigure/BipedStiltwalkerGreedy.swf.

Under MDCOP, the nodes matching between initial and goal are $0 \leftrightarrow G6$, $1 \leftrightarrow G2$, $2 \leftrightarrow G1$, $3 \leftrightarrow G4$, $4 \leftrightarrow G3$, $5 \leftrightarrow G7$, $6 \leftrightarrow G0$, $7 \leftrightarrow G5$, $8 \leftrightarrow G8$, $9 \leftrightarrow G10$, $10 \leftrightarrow G9$, $11 \leftrightarrow G11$. In other words, the sub-configuration composed of modules 0, 1, 2, 6, 8, 10 in initial is isomorphic to the sub-configuration composed of $G6$, $G2$, $G1$, $G0$, $G8$, $G9$ in goal, and the sub-configuration composed of modules 7, 3, 4, 5, 9, 11 is isomorphic to the

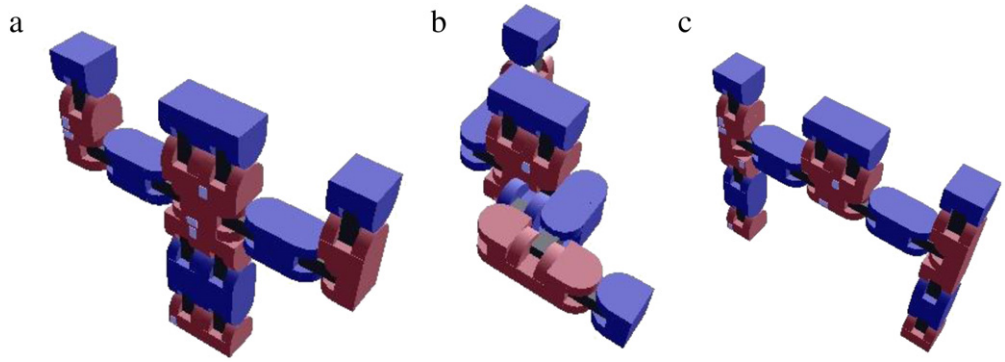


Fig. 13. Biped to stiltWalker under GreedyCM.

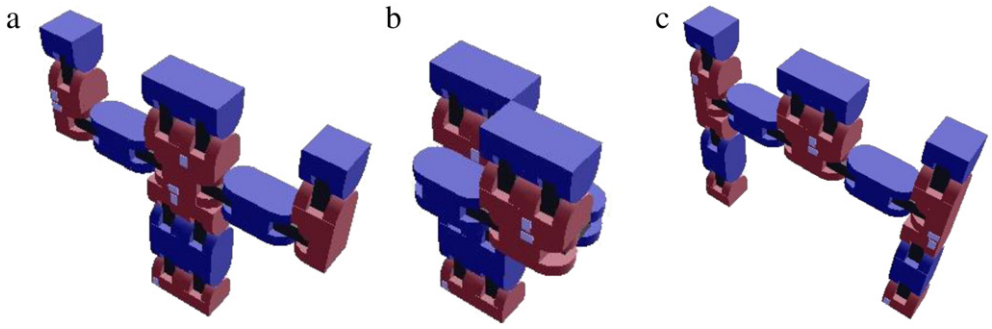


Fig. 14. Biped to stiltWalker under MDCOP.

sub-configuration composed of $G5$, $G4$, $G3$, $G7$, $G10$, $G11$. So, the unmatched edges are the connection between modules 2 and 3 in initial, and modules G2 and G3 in goal. Therefore, MDCOP will generate graph-based reconfiguration actions as: attach modules 1 and 4, and detach modules 2 and 3. This reconfiguration action is only composed of one attach action and one detach action, and they are all executable by SuperBot. Fig. 14 shows the execution process. Video of the reconfiguration is in http://www.isi.edu/robots/reconfigure/BipedStiltwalkerOptimal.swf.

6.2. Statistical experiments

Among the two configuration matching methods we proposed, MDCOP can find the least unmatched edges between initial and goal configurations, and thus generate the shortest graph-based reconfiguration plan. Even though it can find the optimal solution, its exponential computational complexity makes it practical only for a robot with small size. GreedyCM settles for a sub-optimal result but can find the solution very quickly. Its advantage in terms of running time makes it more suitable for a robot with large size in practice. To investigate how far the solution of GreedyCM is from the optimal one, we randomly generated initial and goal configurations with different size and different distance, and studied the results statistically.

Due to the high computational complexity of MDCOP, the least number of unmatched edges between two C-Graphs is hard to compute when the C-Graph has large size. However, we can obtain two C-Graphs $G1$ and $G2$ with minimum number of unmatched edges equal to k by: For $G1$, we first randomly remove k edges which breaks $G1$ into m ($m <= k+1$) disconnected components, and then randomly add $m-1$ edges among the m components to connect them into one C-Graph Gi, and finally reach $G2$ by adding i ($0 \leq i \leq k-m+1$) random edges on Gi. In the above steps, all the added edges are different from the previously removed edges, and i is a random value between $[0, k-m+1]$. When $i = k-m+1$, $G1$ and $G2$ have the same number of edges. After we find the C-Graph pairs $G1$ and $G2$ that differ by k edges, we can evaluate GreedyCM by running it on $G1$ and $G2$ and comparing its result with k.

In our experiments, C-Graphs of size $t = 10, 20, 50, 100$ are examined separately. For each graph size t, and for each minimum number of unmatched edges $k = 1, 2, 5, 10, 20$, we randomly generated 100 C-Graph pairs, and run GreedyCM on them. Table 2 shows the average number of unmatched edges found by GreedyCM, and the percentage of times that GreedyCM finds the least number of unmatched edges in each case.

Based on the results of Table 2, it can be seen that GreedyCM performs quite well. The more similar the C-Graph pairs are, the closer the average number of unmatched edges obtained by GreedyCM is to the optimal value, and the higher chance that GreedyCM will find the least number of unmatched edges. On average, the number of unmatched edges obtained by GreedyCM overestimates the minimum value only by less than 1 edge. So, we can get the conclusion that GreedyCM can find the least unmatched edges and the shortest graph-based plan in most cases.

7. Conclusion and future work discussion

This paper presents significant progress towards the goal of finding the optimal or near-optimal reconfiguration sequence for the reconfiguration planning of modular robots. We have presented the first analysis of the complexity of optimal reconfiguration planning in theory, i.e. finding the least number of reconfiguration actions to transform from an arbitrary initial into an arbitrary goal configuration. We have proved that the problem is NP-complete, even for the seemingly simpler cases where configurations are acyclic. This result provides a theoretical boundary in searching for polynomial algorithms for optimal reconfiguration plans.

We also rephrase the graph-based optimal reconfiguration planning problem into the configuration matching problem, and have proposed two different configuration matching algorithms. The first one is called MDCOP, which can find the maximum

Table 2
Performance of GreedyCM for C-Graphs of various distance and various size.

	Least number of unmatched edges k	Average number of unmatched edges obtained by GreedyCM	Percentage that GreedyCM finds the least number of unmatched edges
C-Graph pairs with size $t = 10$	1	1	100%
	2	2.0201	98%
	5	5.0476	96%
C-Graph with size $t = 20$	1	1	100%
	2	2.0300	99%
	5	5.0573	96%
	10	10.2422	86%
C-Graph with size $t = 50$	1	1	100%
	2	2	100%
	5	5.0452	97%
	10	10.17	90%
	20	20.7099	75%
C-Graph with size $t = 100$	1	1	100%
	2	2	100%
	5	5.0404	99%
	10	10.0408	96%
	20	20.5402	81%

matched edges between the initial and goal configurations, and generates the optimal graph-based reconfiguration steps. The other one is call GreedyCM which runs in polynomial time scalable to the size of robot. Experimental results shows that GreedyCM will find a close-to-optimal reconfiguration plan that has only 0–1 more steps than the optimal reconfiguration steps on average.

The graph-based reconfiguration planner in this paper provides the theoretical results on "what" is the minimum number of connect and disconnect actions needed to reconfigure from one arbitrary shape to another. To apply it on real robots in general, some future works of motion planner on "how" to execute them are needed as:

- Hardware limitation: Each connect action is executed with physical motion for the kinematic chains composed of several modules to align the two to-be-connected connectors. Executability of the alignment depends on many hardware constraints, such as degree of freedom, robot geometries, joint limits, motor torque, stability under gravity, free of collision etc., which vary among different module designs. When applying the graph-based reconfiguration steps on real robots, some extra "repair" steps are needed if the hardware execution insufficiency occurs for some connect actions. How many extra "repair" steps are needed will depend on the capability of different hardware designs. One possible direction is to integrate our graph-based planner with MorphLine [42,43] planner, and implement the motion planning of internal joint angles for a given connect action.
- Order of reconfiguration steps: In what order the reconfiguration steps are executed will make the intermediate configurations different, so it will also affect the executability of some connect and disconnect actions. Finding the correct execution order of the reconfiguration steps will also help bring less "repair" steps for some inexecutable connect actions, and also help making sure the robot is connected all the time.
- Parallel execution: Sometimes, multiple reconfiguration steps can be executed in parallel to reduce the reconfiguration time. In this paper, we are mainly focused on minimizing the number of reconfiguration steps, i.e. the least amount of work to be done for a given reconfiguration task. Minimizing the time to reconfigure can be another interesting topic to explore.

Another possible extension of our research could be the investigation of geometric symmetric configurations. For some module designs, due to the geometric symmetry, some different assembly configurations may lead to robotics structures that are functionally identical. Having interchangeable connectors will help in detecting these symmetry features and could be another interesting topic in our future work.

Acknowledgments

We would thank all members of the Polymorphic Robotics Laboratory at USC/ISI for their intellectual and moral support.

References

[1] S. Murata, H. Hurokawa, E. Yoshida, K. Tomita, S.A. Kokaji, 3D self-reconfigurable structure, in: Proc. of the IEEE International Conference on Robotics and Automation, 1998, pp. 432–439.
[2] K. Kotay, D. Rus, M. Vona, C. McGray, Self-reconfiguring robotic molecule, in: Proc. IEEE International Conference on Robotics and Automation, 1998, pp. 24–431.
[3] J.W. Suh, S.B. Homans, M. Yim, Telecubes: mechanical design of a module for self-reconfigurable robotics, in: Proc. of the IEEE Intl. Conf. on Robotics and Automation, ICRA, Washington, DC, 11–15 May 2002.
[4] Robert Fitch, Zack J. Butler, Daniela Rus, The crystal robot: implementation and demonstration, AAAI Mobile Robot Competition (2002) 65–71.
[5] Daniela Rus, Marsette Vona, Crystalline robots: self-reconfiguration with compressible unit modules, Autonomous Robots 10 (1) (2001) 107–124.
[6] C. Unsal, P.K. Khosla, A multi-layered planner for self-reconfiguration of a uniform group of I-cube modules, in: IEEE International Conference on intelligent Robots and Systems, IROS, October 2001.
[7] M.W. Jorgensen, E.H. Ostergaard, H.H. Lund, Modular ATRON: modules for a self-reconfigurable robot, in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.
[8] B. Kirby, J.D. Campbell, B. Aksak, P. Pillai, J.F. Hoburg, T.C. Mowry, S.C. Goldstein, Catoms: Moving Robots Without Moving Parts, AAAI (Robot Exhibition), Pittsburgh, PA, 2005.
[9] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, T. Nguyen, Self-organizing programmable parts, in: Proc. {IEEE/RSJ} International Conference on Intelligent Robots and Systems, 2005.
[10] P.J. White, K. Kopanski, H. Lipson, Stochastic self-reconfigurable cellular robotics, in: IEEE International Conference on Robotics and Automation, ICRA04, pp. 2888–2893.
[11] K. Gilpin, K. Kotay, D. Rus, I. Vasilescu, Miche: modular shape formation by self-disassembly, The International Journal of Robotics Research 27 (3–4) (2008) 345–372.
[12] R.F.M. Garcia, J.D. Hiller, K. Stoy, H. Lipson, A vacuum-based bonding mechanism for modular robotics, IEEE Transactions on Robotics 27 (5) (2011) 876–890.
[13] W.-M. Shen, S. Behnam, P. Will, Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots, IEEE Transactions on Robotics and Automation 18 (5) (2002) 700–712.
[14] A. Castano, P. Will, Representing and discovering the configuration of conro robots, in: Proceedings of IEEE International Conference on Robotics and Automations.
[15] Mark Yim, David G. Duff, Kimon D. Roufas, PolyBot: a modular reconfigurable robot, in: IEEE International Conference on Robotics & Automation. April 2000.
[16] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, S. Murata, Distributed self-reconfiguration of m-tran III modular robotic system, International Journal of Robotics Research 27 (3–4) (2008) 373–386.
[17] V. Zykov, A. Chan, H. Lipson, Molecubes: an open-source modular robotics kit, in: Proc. IROS-2007 Self-Reconfigurable Robotics Workshop.
[18] Wei-Min Shen, Maks Krivokon, Harris Chiu, Jacob Everist, Michael Rubenstein, Jagadesh Venkatesh, Multimode locomotion for reconfigurable robots, Autonomous Robots 20 (2) (2006) 165–177.
[19] Behnam Salemi, Mark Moll, Wei-Min Shen, SUPERBOT: a deployable, multifunctional, and modular self-reconfigurable robotic system, in: Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Beijing, China, October 2006.
[20] Wei-Min Shen, Self-reconfigurable robots for adaptive and multifunctional tasks, a best paper, in: Proc. 26th Army Science Conference, Florida, USA, December 2008.
[21] M. Park, M. Yim, Distributed control and communication fault tolerance for the ckbot, in: Proc. (ASME/IFToMM) International Conference on Reconfigurable Mechanisms and Robots, London, UK, pp. 682–688.
[22] A. Lyder, R.F.M. Garcia, K. Stoy, Mechanical design of odin, an extendable heterogeneous deformable modular robot, in: Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems, IROS2008, Nice, France, 2008, pp. 883–888.
[23] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, et al., Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface, Industrial Robot 33 (4) (2006) 285–290.
[24] A. Spröwitz, S. Pouya, S. Bonardi, J. van den Kieboom, R. Möckel, et al., Roombots: reconfigurable robots for adaptive furniture, IEEE Computational Intelligence Magazine 5 (3) (2010) 20–32. Special issue on Evolutionary and developmental approaches to robotics.

[25] Kenneth C. Cheung, Erik D. Demaine, Jonathan Bachrach, Saul Griffith, Programmable assembly with universally foldable strings (moteins), IEEE Transactions on Robotics 27 (4) (2011) 718–729.
[26] A. Casal, M. Yim, Self-reconfiguration planning for a class of modular robots, in: Proceedings of SPIE Sensor Fusion and ntralized Control in Robotic Systems II, September 1999.
[27] M. Yim, D. Goldberg, A. Casal, Connectivity planning for closed-chain reconfiguration, in: Proceedings of SPIE Sensor Fusion and Decentralized control in Robotic Systems III, vol. 4196, November 2000.
[28] C.A. Nelson, A framework for self-reconfiguration planning for unit-modular robots, Ph.D. Thesis, Purdue University, Department of Mechanical Engineering.
[29] K. Payne, B. Salemi, P. Will, W.M. Shen, Sensor-based distributed control for chain-typed self-reconfiguration, in: Proceedings of the 2004 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Sendai, Japan, September–October 2004, pp. 2074–2080.
[30] S. Gay, Roombots: toward emancipation of furniture. a kinematics-dependent reconfiguration algorithm for chain-type modular robots, Master Thesis, Ecole Polytechnique, Department of Computer Science.
[31] M. Asadpour, M. Ashtiani, A. Sproewitz, A. Ijspeert, Graph signature for self-reconfiguration planning of modules with symmetry, in: Proceedings of IEEE IROS2009, St. Louis, USA, 2009.
[32] I-Ming Chen, Theory and applications of modular reconfigurable robotic systems, Ph.D. Dissertation, California Institute of Technology, 1994.
[33] M. Park, S. Chitta, A. Teichman, M. Yim, Automatic configuration recognition methods in modular robots, The International Journal of Robotics Research 27 (3–4) (2008) 403–421.
[34] G.S. Chirikjian, A. Pamecha, I. Ebert-Uphoff, Evaluating efficiency of self-reconfiguration in a class of modular robots, Journal of Robotic Systems 13 (5) (1996) 317–338.
[35] Feili Hou, Wei-Min Shen, On the complexity of optimal reconfiguration planning for modular reconfigurable robots, in: Proc. 2010 IEEE Intl. Conf. on Robotics and Automation, Anchorage, Alaska, USA, 2010.
[36] M. Neuhaus, H. Bunke, Bridging the Gap Between Graph Edit Distance and Kernel Machines, World Scientific Publishing Company, ISBN: 9812708170, 2007.
[37] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York, 1979.
[38] Anton Chechetka, Katia Sycara, No-commitment branch and bound search for distributed constraint optimization, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1427–1429.
[39] Adrian Petcu, Boi Faltings, DPOP: a scalable method for multiagent constraint optimization, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI 2005, Edinburgh, Scotland, pp. 266–271.
[40] Roger Mailler, Victor Lesser, Solving distributed constraint optimization problems using cooperative mediation, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, IEEE Computer Society, pp.438–445.
[41] P.J. Modi, W.-M. Shen, M. Tambe, M. Yokoo, ADOPT: asynchronous distributed constraint optimization with quality guarantees, Artificial Intelligence Journal 161 (1–2) (2005) 149–180.
[42] Feili Hou, Wei-Min Shen, Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots, in: Proc. 2008 IEEE International Conference on Robotics and Automation, ICRA 2008, Pasadena, CA, May 2008.
[43] Feili Hou, Wei-Min Shen, Collective reconfigurable systems: fundamentals of self-reconfiguration planning, in: Serge Kernbach (Ed.), Handbook of Collective Robotics: Fundamentals and Challenges, Pan Stanford Publishing, ISBN: 9789814316422, 2012.

Feili Hou received her Ph.D. degree from the University of Southern California in 2011, under Prof. Wei-Min Shen. During her Ph.D. study, she conducted research at the Polymorphic Robotics Laboratory, Information Science Institute. Her interests include modular self-reconfigurable systems, autonomous robots, and artificial intelligence.

Wei-Min Shen received the Ph.D. degree from Carnegie-Mellon University, Pittsburgh, PA, in 1989, under Prof. Herbert Simon.

Currently, he is the Director of the Polymorphic Robotics Laboratory, Information Science Institute (ISI), the Associate Director of the Center for Robotics and Embedded Systems, and a Research Associate Professor of Computer Science all at the University of Southern California (USC), Los Angeles. His research is sponsored by the National Science Foundation (NSF), Air Force Office of Scientific Research (AFOSR), Defense Advanced Research Projects Agency (DARPA), and the National Aeronautics and Space Administration (NASA). He is the author of the book, "*Autonomous Learning from the Environment*" (New York: W.H. Freeman, 1994). His achievements have been reported by the news media, including CNN, PBS, the Discovery channel, The LA Times, BYTE, the Chinese World Journal, and SCIENCES. He has chaired several international conferences and workshops in robotics, machine learning, and data mining and served on the Editorial Boards for two scientific books and one international journal. His current research interests include self-reconfigurable robots, artificial intelligence, and machine learning.

Dr. Shen is the recipient of a 1996 AAAI Robotics Competition Silver-Medal Award, a 1997 RoboCup World Championship Award, and a 1997 Meritorious Service Award at USC/ISI.