On the Complexity of Optimal Reconfiguration Planning for Modular Reconfigurable Robots

Feili Hou, Wei-Min Shen

Abstract— **This paper presents a thorough analysis of the computational complexity of optimal reconfiguration planning problem for chain-type modular robots, i.e. finding the least number of reconfiguration steps to transform from the initial configuration into the goal configuration. It establishes a formal proof that this problem is NP-complete, even if the configurations are acyclic. This result gives a compelling reason that a polynomial algorithm for optimal reconfiguration plan is unlikely to exist. To facilitate future evaluation of reconfiguration algorithms, the paper also provides the lower and the upper bounds for the minimum number of reconfiguration steps for any given reconfiguration problem.**

I. Introduction

Composed of multiple modular robotic units, self-reconfigurable modular robots are metamorphic systems that can autonomously rearrange the modules and form different configurations for dynamic environments and tasks. For example, in the search and rescue scenario, the self-reconfigurable robot can become a wheel to run quickly on the flat terrain to reach the place, change to a spider to climb over the rubble pile, and then morph into a snake shape to penetrate the cracks to reach the victim.

Self-reconfiguration is to solve how to change connectivity among modules to transform the robot from the current configuration into the goal configuration within the restrictions of physical implementation. Depending on the hardware design, reconfiguration algorithms fall into two groups: reconfiguration for lattice-type modular robot and reconfiguration for chain-type modular robot. In lattice-type robot, modules lie in 2D or 3D grids, and the reconfiguration is achieved through discrete movements of modules detaching from the current lattice location, moving along and surface of the robot and docking at the adjacent cells. Example reconfiguration work includes Pamecha[1], Yim[2], Kurokawa and Murata [3], Hosokawa[4], Rus[5], Walter[6], Khosla[7], Slee[8], Aloupis[9] etc. In chain-type robots, modules can form moving chains and loops of any graph topology, and the reconfiguration is achieved through "connect" and "disconnect" operations between modules along with the joint motion of chains composed of several modules. Due to its difficulty, the chain-type reconfiguration has received less attention. Existing algorithms include Yim[10], Nelson[11], Gay[12], Shen[13,14].

The different geometric arrangement of modules between lattice-type and chain-type modular robots makes their reconfiguration planning mechanisms fundamentally different. The work in this paper is more focused on chain-type reconfiguration. For simplicity, we will use the term "modular robots" or simply "robots" to denote "the chain-type modular robots.", and use "reconfiguration" to denote "chain-type reconfiguration" in the following.

The existing reconfiguration algorithms used different methods, such as divide-and-conquer [10], graph matching [11] etc, to reduce the reconfiguration cost. However, the optimal solution with least reconfiguration steps has never been reached. Can we achieve the optimal solution efficiently? How hard is it to find out the least number of reconfiguration steps? It is commonly agreed that this problem is computationally intractable, but a concrete support for this belief is still lacking. One widely used explanation is that the configuration space is exponential [1, 10, 12], but that alone is not enough to show that the problem cannot be solved efficiently in polynomial time. Actually, many optimization problems can be solved efficiently even if the search space is exponential. For example, the shortest- path problem for graphs without negative cycles can be solved efficiently, but the number of paths between two nodes can grow exponentially with the number of nodes in the graph.

The aim of this paper is to provide computational complexity analysis of optimal reconfiguration for modular robots. To our knowledge, this paper is the first one that provides a theoretical proof to the problem. We have proved 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. This result demonstrates that suboptimal solutions in current literature are satisfactory, and provides a compelling reason to stop searching for a polynomial algorithm for optimal reconfiguration plan in the future. Furthermore, we have established the lower bounds and upper bounds of the minimum number of reconfiguration steps, which can be used to facilitate future evaluation of reconfiguration algorithms.

The rest of paper is organized as follows. Section II defines the problem of optimal reconfiguration planning,. Section III gives an analysis of the computational complexity, and proves that it is NP-complete. The lower and upper bounds analysis is given in section IV, and conclusions are drawn in section V.

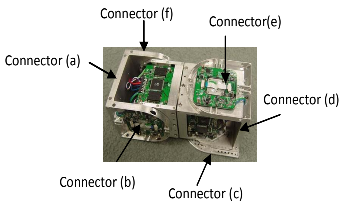
II. Optimal Reconfiguration Planning Problem

A. Configuration Representation

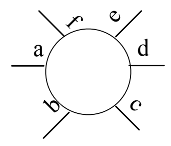
Before defining the optimal reconfiguration planning

Feili Hou, Wei-Min Shen are with Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292,USA. (e-mail: fhou@usc.edu, , shen@isi.edu).

problem, we would describe our representation of robot's configuration first. Two robots with the same graph topology can function differently if the modules are connected via different connectors. For example, Fig.1-a is a SuperBot module with six connectors. We name the connectors as *a*, *b*, *c*, *d*, *e*, and *f* (Figure 1-d). Three SuperBot modules can form a T-shaped robot (Fig.1-b) or a snake robot (Fig.1-c), although topologically both of them are in a "line" shape if we view modules as nodes and connections as edges (Figure 1-e and 1-f). To fully represent a robot's configuration, a special graph called C-Graph (Connector-Graph) is proposed here. C-Graph is the extension of normal graph with differentiated connecting points. 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 *i* and module *v*'s connector *j* corresponds to an edge '$i \leftrightarrow j$' between *u* and *v*, or '$j \leftrightarrow i$' between *v* and *u*. Fig. 1-e and 1-f are the C-Graphs representation for our T-shaped robot and snake robot in Fig. 1-b and 1-c. In the following, we will call "node" or "module", "edge" or "connection" interchangeably. Two configurations are regarded as equivalent if and only if their C-Graphs are isomorphic.

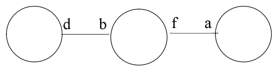


(a) A SuperBot module

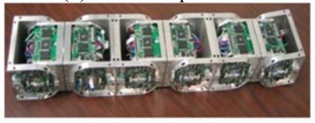


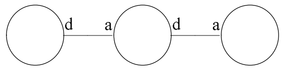
(d) C-Graph of SuperBot module

(b) The T-shape robot



(e) C-Graph of T-shape robot


(c) The snake robot



(f) C-Graph of snake robot

Fig. 1 Modular robots and their C-Graph representation

B. *Reconfiguration Actions*

The two elementary reconfiguration actions are: (1) making new connections or (2) disconnecting current connections between modules for connectivity rearrangement. The robot can bend its body through module joints, so any two modules with free connectors can potentially be aligned and dock with each other. Also, any existing connections can potently be disconnected. Fig. 2 shows an instance of reconfiguring process.

Execution of reconfiguration actions depends on many factors such as degree of freedom, motor forces, robot geometries etc, which vary among different module designs. However, one reasonable assumption can be made as: an open chain with three or more modules can always dock the two ends and form a loop. Moreover, it is preferred that the robot keeps connected throughout the reconfiguration process.

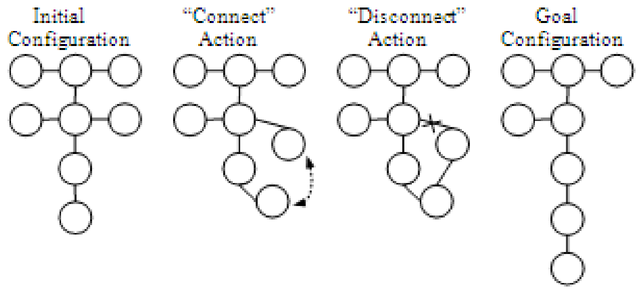


Fig. 2 Example of reconfiguration process

C. *Optimal Reconfiguration Planning Problem*

The reconfiguration planning problem is defined as how modules in one configuration rearrange into another using several sets of reconfiguration actions. Basically, what connections to make and what connections to disconnect so as to reconfigure from arbitrary one shape to another? Without loss of generality, we will always assume that the number of modules in the initial configuration is the same as that in the goal configuration.

During the reconfiguration process, the reconfiguration actions are most time- and energy-consuming, so it is a common practice to aim at minimizing the number of reconfiguration steps, i.e. the number of "connect" actions plus the number of "disconnect" actions. 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.

Since the number of physical connections is predefined in the initial and goal configurations, the number of "connect" actions is fixed once the number of "disconnect" action is known, and vice versa. So we get

Lemma 1: The optimal reconfiguration planning problem is to find the either one of the following metrics:
(1) Least number of "connect" actions,
(2) Least number of "disconnect" actions,
(3) Least number of reconfiguration steps (i.e., the number of "connect" actions plus the number of "disconnect" actions).

III. COMPUTATIONAL COMPLEXITY OF OPTIMAL RECONFIGURATION PLANNING

A. *A Short Review of NP-Completeness*

A problem X is defined as NP-complete if:

1) 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.

2)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 $Y<_pX$. $Y<_pX$ 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 problem 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. From the view of polynomial-time solvability, there is no significant difference between them. An optimization problem is NP-complete if its corresponding decision problem is NP-complete [15].

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.

B. NP-Completeness of Optimal Reconfiguration Planning Problem

Intuitively, reconfiguration cost is low when initial and goal configurations are similar, and high when they are quite different. This drives us to relate the reconfiguration planning problem to graph similarity and matching problem. As we know, many graph similarity problems, such as largest common subgraph etc, are NP-complete, but they become polynomial-time solvable when the underlying graphs are acyclic. This leads to an expectation that the optimal reconfiguration planning may also be solved efficiently when the initial and goal configurations are acyclic. To be short, we refer this problem as ACYCLIC OPTIMAL RECONFIGURATION problem. Unfortunately, based on our study, we find that even for acyclic configurations, the optimal reconfiguration problem 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, so the decision version of 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 a NP-complete problem is as follow:

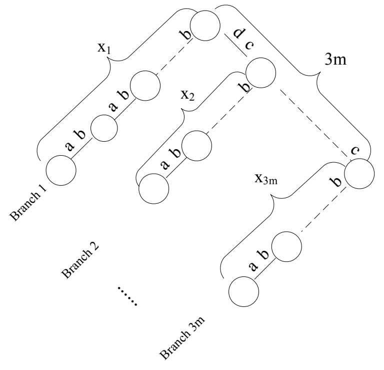
Step 1: Show that it is in NP. Given any reconfiguration plan with at most *2n* steps, it is obvious that we could check in polynomial time that whether I can be transformed into G.

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

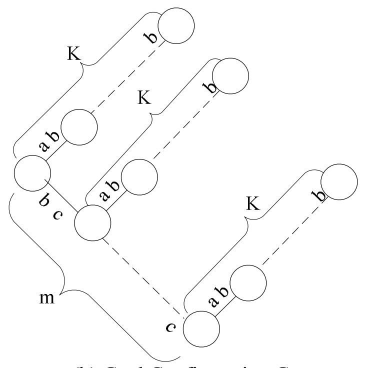
The 3-PARTITION problem is to decide whether a given multiset of integers can be partitioned into subsets that all have the same sum [15]. More precisely, it is defined as :

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,..., 3m). Can S be partitioned into m disjoint subsets S_1, ..., S_m such that the sum of the numbers in each subset is equal, i.e.=K (j=1…m) ?

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.3. The connectors are labeled alphabetically as *a b c d*. In the initial configuration I, we start with *3m* branches with each branch *i* has X_i number of nodes connected in a line by edge 'a⟷b'. Then, we connect these line branches consecutively by their rightmost nodes via edge 'd⟷c'. 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 'a⟷b'. These *m* branches are connected consecutively by their leftmost nodes via edge 'b⟷c'.



(a) Initial Configuration I



(b) Goal Configuration G

Fig. 3 Constructed instance of initial and goal configurations

To be precise, it actually takes pseudo-polynomial instead of polynomial time to construct I and G, 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 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 [15].

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

Lemma 2 (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.

Proof: If a 3-PARTITION problem with instance S has a solution, then S can partitioned into *m* disjoint subsets $S_1, \ldots S_m$. It has been proved that if 3-PARTITION has a solution, each subsets S_j must be a triple with exactly three elements [15]. Using a given solution to the 3-PARTITION, we can reconfigure I into G as follows. We first disconnect all the connection 'd⟷c' in I, which requires *3m-1* "disconnect" actions. This results in *3m* line branches where branch *i* has X_i nodes. Then, for each triple $S_i=\{X_{i_1}, X_{i_2}, X_{i_3}\}$, we connect the corresponding branches i_1 i_2 and i_3 consecutively into a single line branch by connecting the rightmost node of branch i_1 with the leftmost node of branch i_2 by edge 'a⟷b', and also the rightmost node of branch i_2 with the leftmost node of branch i_3 by edge 'a⟷b'. Bolded lines in Fig. 4 show this process, which takes *2* "connect" actions. Since $X_i^1 + X_i^2 + X_i^3 = K$, the generated line branch has K nodes. By this way, we end up with *m* length-K line branches after *2m* "connect" actions. After this, we will connect these *m* lines consecutively by their leftmost nodes with edge 'b⟷c', which needs *m-1* "connect" actions. The goal configuration G is thus reached in the end. We can see that the whole reconfiguration process takes *3m-1* "disconnect" actions of 'd⟷c', *2m* "connect" actions of 'a⟷b', and *m-1* "connect" actions of 'b⟷c', i.e. *6m-2* reconfiguration steps in total. Therefore, I can be transformed to G in at most *6m-2* steps.

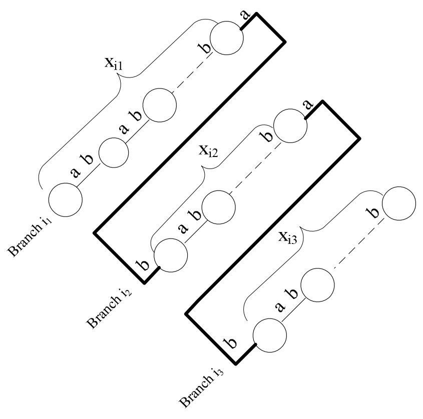


Fig. 4 Example of connecting three line branches into one

Lemma 3 (Completeness): Let S be an arbitrary instance in the 3-PARTITION problem, and 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 the edges inside those *3m* line branches, then for each "connecting" action between a branch *i* and a branch *j*, we put integer X_i and X_j into same subset. This defines a partition of set S into *m* subsets with equal sum K.

Let's first compare the number of edges in each type between I and G. Take the edge of type 'a⟷b' as example. In I, each branch *i* has X_i*-1* edges of 'a⟷b', and thus $\sum_1^{3m} X_i$*-3m* edges of 'a⟷b' 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 I.

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

	# of edge 'a⟷b'	# of edge 'b⟷c'	# of edge 'd⟷c'
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)

Based on the values in the third row of table I, it can be seen that to reconfigure I into G, it is a must-have to make *2m* new connections of 'a⟷b', *m-1* new connections of 'b⟷c', and disconnect *3m-1* existing connections of 'd⟷c'. 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 'd⟷c' will disconnect I into *3m* line branches without breaking the 'a⟷b' edges within any one 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 'a⟷b' must be that of connecting the two branches into one without interfering the inner modules. Since the goal configuration has *m* branches, where each branch has K nodes connected in a line by 'a⟷b', 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 'b⟷c'.

For each connect action 'a⟷b' between branch i and branch j, we put integer X_i and X_j into 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 is equal to K. Namely, the 3-PARTITION problem with instance S has a solution. This completes proof of lemma 4.

Lemma 4: When the initial and goal configurations are acyclic, optimal reconfiguration planning is still NP-complete

Proof: In the above discussion, step 1) shows that ACYCLIC OPTIMAL RECONFIGURATION is in NP. Lemma 2 and lemma3 demonstrate that ACYCLIC OPTIMAL RECONFIGURATION is NP-hard. So, the problem is NP-complete.

One thing to clarify before completing the proof of lemma 4 is: we describe the "disconnect" actions before "connect" actions during our proof of lemma 2 and lemma 3 just for easy understanding, and it doesn't mean the robot has to execute all the "disconnect" actions first and be separated apart into several lines. Since the order of reconfiguration actions will not affect the configuration to be reached, we can have the robot always execute one 'connect' action followed by one 'disconnect' action in the reconfiguration process. The 'disconnect' action will always break the loop resulting from the previous "connect" action, so that the robot can keep connected all the time. All the "connect" actions are also compatible with the hardware design since it always involves three or module modules in an open chain.

Since ACYCLIC OPTIMAL RECONFIGURATION problem is a special case of the optimal reconfiguration planning problem for all configurations, it is obvious that:

Lemma 5: Optimal reconfiguration planning of finding the least number of reconfiguration steps for chain-type modular robots is NP-complete.

IV. Bounds of Least Reconfiguration Steps

Though it is hard to find the least number of reconfiguration steps, knowing the bounds where the optimal solution lies will be invaluable for evaluating future algorithms. This can help to estimate the "distance" between the initial and the goal configurations, and provide a criterion to evaluate a non-optimal reconfiguration solution. We desire that these bounds can be computed quickly.

A. The lower bound of the least reconfiguration steps

It is known that two isomorphic configurations are exactly the same in terms of the number of connections of each type. So the lower bounds can be quickly derived by comparing and counting edges in the initial and goal configurations. For each connection type '$c_i\longleftrightarrow c_j$', if the number of $c_i\longleftrightarrow c_j$ in I, #I(ci⟷cj), is less(or more) than that in G, #G(ci⟷cj), it needs at least $\#G(c_i\longleftrightarrow c_j) - \#I(c_i\longleftrightarrow c_j)$ "connect" (or $\#I(c_i\longleftrightarrow c_j) - \#G(c_i\longleftrightarrow c_j)$ "disconnect") actions to reach the goal configurations. . Our proof process in lemma 3 is an example of computing the lower bound of reconfiguration steps. More precisely, it can be expressed as:

Lemma 6 For a given module design with *q* connectors as $c_1,\ldots,c_q,$ assume that for each connection '$c_i\longleftrightarrow c_j$' initial configuration I has $\#I(c_i\longleftrightarrow c_j)$ edges, and goal configuration G has $\#G(c_i\longleftrightarrow c_j)$ edges, then the lower bound of least number of "connect" actions is

$$\sum_{\substack{i=1\ldots q,j=i\ldots q\\ \#G(ci\leftrightarrow cj)-\#I(ci\leftrightarrow cj)>0}} \#G(ci \leftrightarrow cj) - \#I(ci \leftrightarrow cj) \qquad (2)$$

, the lower bound of the least number of "disconnect" actions are

$$\sum_{\substack{i=1\ldots q,j=i\ldots q\\ \#G(ci\leftrightarrow cj)-\#I(ci\leftrightarrow cj)<0}} \#I(ci \leftrightarrow cj) - \#G(ci \leftrightarrow cj) \qquad (3)$$

, and the lower bound of total number of the least reconfiguration steps is

$$\sum_{i=1\ldots q,j=i\ldots q} |\#G(ci \leftrightarrow cj) - \#I(ci \leftrightarrow cj)| \qquad (4)$$

B. The upper bound of the least reconfiguration steps

The upper bound of the reconfiguration steps is inspired from MorphLine algorithm [14]. Since MorphLine can always find a solution for arbitrary initial and goal configurations, the number of reconfiguration steps it takes can serve as an upper bound of the least reconfiguration steps.

The main idea of MorphLine is to first transform the initial configuration I into an acyclic (tree) configuration Ia by spanning tree algorithm, and then transform Ia into a line Il. The goal configuration G can grow from an acyclic (tree) configuration Ga embedded in G by closing the corresponding loops, and Ga can be transformed from another line configuration Gl by reversing the steps from Ga to Gl. The two line configurations of Il and Gl may be different in terms of the connectors used by modules. So, the whole process to transform I to G is: I->Ia->Il->Gl->Ga->G.

The procedure of transforming an acyclic (tree) configuration Ia to a line Il is: in a bottom up traversal, whenever a node has more than one children branches, it will keep merge one branch into another until it only has one child branch. Each merging step consists of a "connect" action and a "disconnect" action. Fig. 5-a shows an example of reconfiguring an acyclic configuration into a line.

The process of growing to an arbitrary acyclic (tree) configuration Ga from a line Gl is the reversion of the process of Ga->Gl. Fig. 5-b shows an example of this.

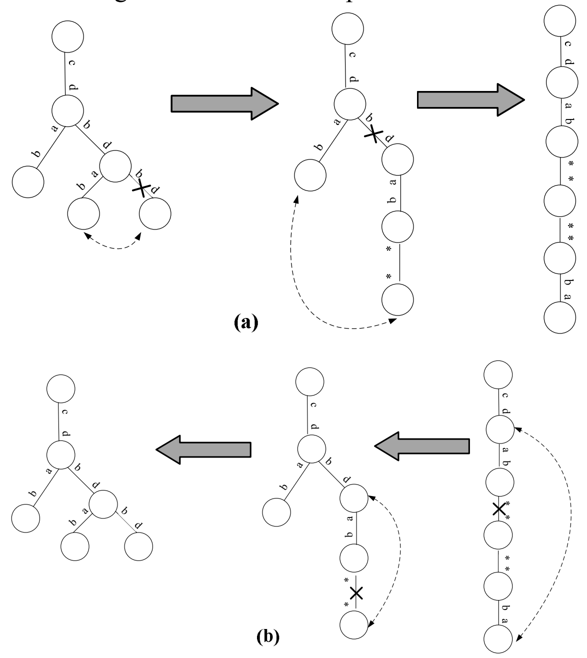


Fig. 5 (a) Example of reconfiguring an acyclic configuration into a line
(b) Example of reconfiguring a line into an acyclic configuration

In the process of Il->Gl, a temporary loop may be formed to ensure that the robot keeps connected. Fig. 6 shows an example of transforming between two lines.

Actually, we can efficiently compute this upper bound without executing the MorphLine algorithm. Suppose there are N modules in I and G, and the number of edges in I and G is E(I) and E(G), respectively. Since Ia and Ga are acyclic (tree) configurations with N-1 edges, we know that it takes

$$E(I)-(N-1) \qquad (5)$$

"disconnect" actions in the process of I->Ia, and

$$E(G)-(N-1) \qquad (6)$$

"connect" actions in the process of Ga -G.

During the process of Ia->Il, since every merging process reduce the degree of a bifurcation module by one (Here, degree of a module is the number of edges incident to it. A module with degree greater than 2 is called bifurcation module), and our goal is to have every module's degree to be no larger than 2, it can be derived that the number of "connection" actions is the same as that of 'disconnection' actions, and is equal to

$$\sum_{V\in Ia} \overline{\mathrm{Deg}}(V) \qquad (7)$$

Where $\overline{\mathrm{Deg}}(V)$= Deg(V)-2>0? Deg(V)-2:0 .

Since the process of Gl ->Ga is the reverse of Ga ->Gl, the number of "connect" and 'disconnect' pairs in Gl->Ga is

$$\sum_{V\in Ga} \overline{\mathrm{Deg}}(V) \qquad (8)$$

Based on (7) and (8), we know that out of the N-1 edges in Il, $\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)$ edges are newly formed and not exist in Ia. In other words, these $\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)$ edges can be controlled in the process of Ia->Il so as to make them consistent with the edges in Gl. Similarly, out of the N-1 edges in Gl, $\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)$ edges is to be disconnected during Gl->Ga. Namely, these $\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)$ edges do not affect the configuration Ga that can be reached.

So the worst case is that there are

$$Min[\ N\text{-}1\text{-}\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)\ ,\ N\text{-}1\text{-}\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)\ ] \qquad (9)$$

edges that are inconsistent between Il and Gl and needs to be changed. Counting the pair of "connect" and "disconnect" actions to form the temporary loop in Il-Gl, the total number of "connect" and "disconnect" action pairs in Ia->Il is no greater than

$$Min[\ N\text{-}\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)\ ,\ N\text{-}\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)\ ] \qquad (10)$$

Also, since

$$\sum_{V\in Ia} \overline{\mathrm{Deg}}(V) < \sum_{V\in I} \overline{\mathrm{Deg}}(V), \quad \sum_{V\in Ga} \overline{\mathrm{Deg}}(V) < \sum_{V\in G} \overline{\mathrm{Deg}}(V) \qquad (11)$$

We get the upper bounds in summary as follow:

Lemma 7 For a given initial configuration I and a goal configuration G, the upper bound of the least number of "connect" actions is

$$\begin{aligned} & E(G)\text{-}(N\text{-}1)+\sum_{V\in Ia} \overline{\mathrm{Deg}}(V) + \sum_{V\in Ga} \overline{\mathrm{Deg}}(V) \\ & + Min[\ N\text{-}\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)\ ,\ N\text{-}\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)\ ] \\ & = E(G)+1+Min[\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)\ ,\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)] \\ & \le E(G)+1+Min[\sum_{V\in I} \overline{\mathrm{Deg}}(V),\ \sum_{V\in G} \overline{\mathrm{Deg}}(V)\ ] \qquad (12) \end{aligned}$$

, the upper bound of the least number of "disconnect" actions is

$$\begin{aligned} & E(I)\text{-}(N\text{-}1)+\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)+\sum_{V\in Ga} \overline{\mathrm{Deg}}(V) \\ & +Min[\ N\text{-}\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)\ ,\ N\text{-}\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)\ ] \\ & = E(I)+1+Min[\sum_{V\in Ia} \overline{\mathrm{Deg}}(V)\ ,\sum_{V\in Ga} \overline{\mathrm{Deg}}(V)] \\ & \le E(I)+1+Min[\sum_{V\in I} \overline{\mathrm{Deg}}(V),\ \sum_{V\in G} \overline{\mathrm{Deg}}(V)\ ] \qquad (13) \end{aligned}$$

, the upper bound of the number of reconfiguration actions is

$$\begin{aligned} & E(G)+1+Min[\sum_{V\in I} \overline{\mathrm{Deg}}(V),\ \sum_{V\in G} \overline{\mathrm{Deg}}(V)] \\ & + E(I)+1+Min[\sum_{V\in I} \overline{\mathrm{Deg}}(V),\ \sum_{V\in G} \overline{\mathrm{Deg}}(V)] \\ & = E(I)+E(G)+2+2Min[\sum_{V\in I} \overline{\mathrm{Deg}}(V),\ \sum_{V\in G} \overline{\mathrm{Deg}}(V)] \qquad (14) \end{aligned}$$

, where $\overline{\mathrm{Deg}}(V) = \mathrm{Deg}(V)\text{-}2>0?\ \mathrm{Deg}(V)\text{-}2:0$.

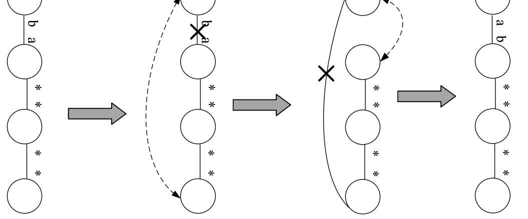


Fig. 6 Example of reconfiguring between two line configurations

V. CONCLUSION

This paper provides a thorough complexity analysis of the optimal reconfiguration planning for chain-type modular robots. We have proved that the optimal reconfiguration planning problem of finding the least number of reconfiguration steps is NP-complete, and presented an efficient procedure to estimate the lower and upper bound for the optimal solution. The findings in this paper provide a theoretical foundation for guiding the search for future reconfiguration algorithms and propose an objective criterion to evaluate the performance of reconfiguration algorithms for modular and reconfigurable robots.

REFERENCES

[1] Pamecha A, Ebert-Uphoff I, Chirikjian G: Useful metrics for modular robot motion planning. IEEE Trans. on Robotics and Automation 13(4): 531-545, 1997

[2] Sergei Vassilvitskii, Jeremy Kubica, Eleanor G. Rieffel, John W. Suh, Mark Yim: On the General Reconfiguration Problem for Expanding Cube Style Modular Robots. ICRA 2002: 801-808

[3] Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Eiichi Yoshida, Shigeru Kokaji and Satoshi Murata. Distributed Self-reconfiguration Control of Modular Robot M-TRAN, Proceedings of 2005 IEEE International Conference on Mechatronics and Automation (ICMA2005), pp. 254-259, 2005

[4] K. Hosokawa, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, I. Endo, "Self-organizing collective robots with morphogenesis in a vertical plane," JSME Intl. Journal Series C Mechanical Systems Machine Elements and Manufacturing 42(March 1999):195-202

[5] Zack Butler, Satoshi Murata, Daniela Rus, Distributed Replication Algorithms for Self-Reconfiguring Modular Robots, Proceedings of DIstributed Autonomous Robotics Systems 5, 2002

[6] J. Walter, E. Tsai, and N. Amato, Algorithms for Fast Concurrent Reconfiguration of Hexagonal Metamorphic Robots, IEEE Transactions on Robotics, Vol. 21, No. 4, pages 621-631, 2005

[7] Ünsal, C., and Khosla P. K. A Multi-layered Planner for Self-Reconfiguration of a Uniform Group of I-Cube Modules, IEEE International Conference on intelligent Robots and Systems (IROS), October 2001.

[8] Reif J. H., Slee S. Optimal Kinodynamic Motion Planning for 2D Reconfiguration of Self-Reconfigurable Robots, Robotics: Science and Systems Conference, Georgia Institute of Technology, Atlanta, GA, June 27-30, 2007.

[9] Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer, "Reconfiguration of Cube-Style Modular Robots Using O(log n) Parallel Moves", in Proceedings of the 19th Annual International Symposium on Algorithms and Computation (ISAAC 2008), Gold Coast, Australia, December 15–17, 2008, pages 342–353.

[10] Casal, A. and Yim, M. (1999). Self-Reconfiguration Planning For a Class of Modular Robots, Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II, Sept. 1999.

[11] Nelson C.A, "A framework for self-reconfiguration planning for unit-modular robots", Phd Thesis, Purdue University, Department of Mechanical Engineering, 2005

[12] Gay S., Roombots: Toward Emancipation of Furniture. A Kinematics-Dependent Reconfiguration Algorithm for Chain-Type Modular Robots, Master Thesis, Ecole Polytechnique, Department of Computer Science, 2007

[13] Wei-Min Shen, Behnam Salemi, and Peter Will. Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots. IEEE Trans. on Robotics and Automation, 18(5):700–712, October 2002.

[14] Feili Hou, Wei-Min Shen, "Distributed, Dynamic, and Autonomous Reconfiguration Planning for Chain-Type Self-Reconfigurable Robots", Proc. 2008 IEEE International Conference on Robotics and Automation (ICRA 2008),Pasadena, CA, May 2008.

[15] Garey, Michael R. and David S. Johnson. Computers and Intractability: A Guide tothe Theory of NP-Completeness. New York: W. H. Freeman and Company, 1979