Distributed, Dynamic, and Autonomous Reconfiguration Planning for Chain-Type Self-Reconfigurable Robots

Feili Hou, Wei-Min Shen

Abstract— **This paper presents a dynamic and distributed reconfiguration planning algorithm for chain-type self-reconfigurable robots, by which a robot can autonomously self-reconfigure from one arbitrary acyclic configuration to another in a distributed way. The novel features of this algorithm include: (1) an efficient representation for unlabeled complex configurations; (2) a distributed comparison to detect common/different substructures in two configurations; (3) reconfiguration are limited to those modules that indicate the differences in topology; and (4) reconfiguration actions are performed in parallel and distributed fashion, where every module decides its own actions locally and coordinate asynchronously to rearrange into the goal configuration. The algorithm is applicable to any chain-type self-reconfigurable robots in general.**

I. Introduction

SELF-RECONFIGURABLE modular robots are metamorphic systems that can autonomously change their configurations and locomotion, based on the mission and the environment. Due to their modularity, versatility and self-healing ability, they have a great potential in achieving complex tasks in unstructured and dynamic environments.

The automatic self-reconfiguration ability allows the modular robot to radically adapt to changes in the environment. For example, in the search and rescue scenario, a wheeled configuration is used to run quickly on the flat terrain to reach the rubble pile. Then a spider configuration is needed to walk over the rubble pile. Once the victim is found, a snake shape is required to penetrate the cracks to reach the victim. The ability to self-reconfigure with a large number of independent modules is one of the vital issues to realize the versatility of self-reconfigurable robots.

Depending on the hardware design, the self-reconfigurable robots fall into two groups: lattice-type and chain-type robots. Currently, most reconfiguration planning algorithms are focused on the lattice type robots, such as the work done by Pamecha[1], Yim[2], Kurokawa and Murata [3], Hosokawa[4], Rus[5], Walter[6] etc. All the above algorithms were proposed for the reconfigurable robot whose modules lie in discrete positions in a 2D or 3D lattice.

In contrast, the modules in a chain-type reconfigurable robot are not restricted to lattice cells positions, but instead can form movable chains or loops. This offers more versatility in its locomotion, but opens up more challenges for the reconfiguration planning algorithm. Configuration transformation involves the motion of chains instead of only one module, which increases the dimensionality of potential reconfiguration actions. Besides, the number of possible configurations also grows exponentially with the number of modules included in the robot. How the independent modules collaborate their local actions to accomplish the goal configuration is a big challenge.

Hardware implementations of the chain-type robots include PolyBot[7], Conro[8], M-TRAN[9] and SuperBot[10]. Currently, only a few works were published on the reconfiguration of chain-type robots. Casal, A. and Yim [11] presented a divide-and-conquer strategy to the chain-type reconfiguration problem. Nelson [12] used the graph theory for solving the reconfiguration problem. However, they are all centralized off-line planner that the goal configuration is predetermined and the action sequences are stored in the modules beforehand. During the task execution, features of the environment may be not known beforehand, or unexpected change may happen. It is preferable that the robot can figure out the appropriate configurations and do the reconfiguration actions by itself at run time. In 2004, Shen et. al. [13] used the hormone- inspired distributed control for automatic planning and execution of self-reconfiguration, but it is only limited from "I" shape to "T" shape.

This paper proposes a reconfiguration planning algorithm for chain-type robots, with which the robot can autonomously transform from an arbitrary acyclic configuration to another in a distributed way. Every module is homogeneous without ID, and the unlabeled configuration of the robot is represented by a novel way called configuration string. The common/different substructure between the robot's current configuration and the goal configuration is detected, so that reconfiguration actions can be limited only to the modules that indicate the difference. The reconfiguration behavior is not monitored by a central controller, or any predefined action sequences, but emerges from the local behaviors of individual modules and communication between them. Once the environment is changed and a new goal configuration is required, every module can form its own local interpretations and reconfiguration actions, and coordinates asynchronously to achieve it. Several chains in the robot can move in parallel to speedup the overall process.

The rest of this paper is organized as follows. Section II

This work was supported by NASA (NNA05CS38A).

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)

describes the reconfiguration planning problem. The distributed configuration comparison algorithm is proposed in Section III, while the distributed reconfiguration actions are described in Section IV. Section V discusses the features of our algorithms. Finally, conclusion and future work are made in Section VI.

II. Reconfiguration Planning Problem

Before defining the reconfiguration planning problem, we explain our representation of a robot's configuration first. A robot's configuration is represented by a modular graph, where each node is a module and each edge is the physical connection between modules. Nodes do not have any unique global IDs, and they can have half-duplex communication only with their immediate neighbors through existing links. Fig.1 shows a SuperBot configuration and its modular graph. We define two configurations as equivalent if and only if their modular graphs are the same. The configuration space is the set of all configurations that a given set of modules may form. Two configurations are adjacent if one can be transformed into the other by one set of reconfigure actions, which is an attach action followed by a detach action, together with some motion of chains. Fig. 2 shows an instance of two adjacent configurations and the set of reconfigure actions between them. For the purpose of reliability, we require that the robot remains connected throughout the reconfiguration process.

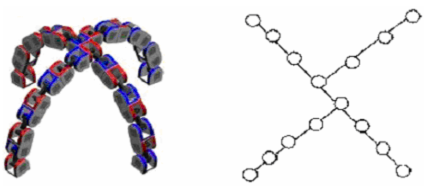
Fig. 1. A SuperBot configuration and its modular graph

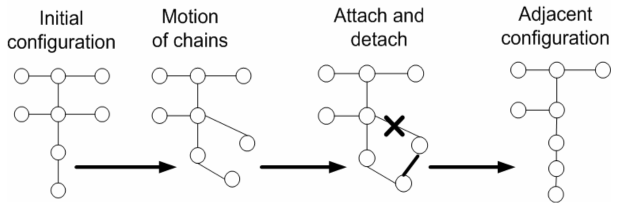

Fig. 2. A set of reconfiguration actions to transform from one configuration to its adjacent configuration

The reconfiguration planning problem is defined as how modules in one configuration rearrange into another using several sets of reconfiguration actions within the restrictions of the physical module implementation. In this paper, we only consider the planning of connectivity rearrangement in modular graphs. The real word concerns like kinematic constraint, gravity, collision avoidance etc are abstracted away and will be the subject of future work. Reconfiguration between cyclic graphs will also be considered in the future.

III. Distributed Configuration Comparison

The modular graphs of the current configuration and the goal configuration usually have some subgraphs sharing the common topology, such as the white nodes in Fig. 3(a) and Fig.3(b). We call these as Not-To-be-Reconfigured (*NTBR*) subgraphs, while the others as To-be-Reconfigured (*TBR*) subgraphs. To be effort efficient, we have the reconfiguration changes limited only within the modules in the *TBR* subgraphs. Hereby, before describing the reconfiguration actions, we propose a distributed comparison algorithm in this section, by which each module can decide whether it belongs to the *TBR* subgraph by communicating with their immediate neighbors.

In the following context, we will use *NC* to represent the number of connectors in a module. As an illustration to our algorithm, we will go through an example from Fig 3(a) to Fig 3(b), and assume *NC* = 4 in the example. Please note that the node IDs in the figures and our assumption of *NC* value are just for explanation. The reconfiguration planning algorithm is applicable to different module design with arbitrary *NC* value.

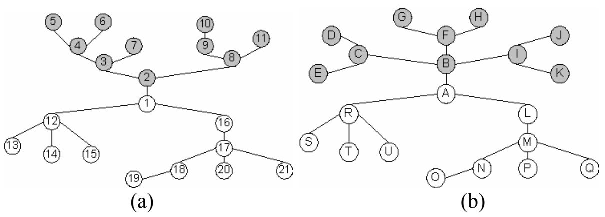

Fig. 3. Modular graph of the current configuration and the goal configuration

A. Goal Configuration Representation

Once the reconfiguration task is initiated, the modular graph of the goal configuration is informed to every module in the robot. In the graph theory, there are many ways to represent a graph, like adjacent matrix, incidence matrix etc. However, they all need label each nodes, and here in our problem all the modules are homogeneous and thus do have IDs in the modular graph. So a new way to represent the unlabeled graph called configuration string is proposed here. First, we define an array called connection number (*CN*), with size *NC*. Each module has a *CN* variable, where *CN*[*i*] denotes the number of modules connected to its ith connector. For example, in Fig. 3(b), node A has 4 connectors, where the first three are connected to 10(module B, C, …, K), 6(module L, M, …, Q) , and 4 (module R, …, U) modules respectively, and the fourth one has no connection. So, its *CN* is [10, 6, 4, 0].

For each module, the sum of all the elements in its *CN* equals to the number of modules it connects. Since all the modules in the robot keep connected, the total number of modules included in the robot is equal to

$$\text{Sum\_Modules} = \sum_{i=1}^{NC} CN[i] + 1 \qquad (1)$$

, namely the number of all its connected modules plus the module itself. It can be seen that the sum of *CN* is the same for all the modules, and is equals to Sum_Modules-1.

Theorem 1[14]: The center of an acyclic graph always exists. It is a unique vertex or a unique pair of adjacent vertices such that removing that vertex (or pair of vertices) from the graph leaves a collection of components each having less than half of the vertices

According to theorem 1, only one node (or a pair of nodes) will be the center of any acyclic graph. A node is the center if all of its *CN* elements are less than half of Sum_Modules. Starting from the center node (or an arbitrary one from the center pairs), we traverse the modular graph of the goal configuration by Depth-First-Search (DFS). Table I shows the traversed order and the *CN* values of all the nodes of Fig. 3(b).

TABLE I DFS SEQUENCE AND CN VALUES OF THE NODES IN FIG 3(B)

DFS Order	Node ID	CN Value	DFS Order	Node ID	CN Value
1	A	[10 6 4 0]	12	L	[15 5 0 0]
2	B	[11 3 3 3]	13	M	[16 2 1 1]
3	C	[18 1 1 0]	14	N	[19 1 0 0]
4	D	[20 0 0 0]	15	O	[20 0 0 0]
5	E	[20 0 0 0]	16	P	[20 0 0 0]
6	F	[18 1 1 0]	17	Q	[20 0 0 0]
7	G	[20 0 0 0]	18	R	[17 1 1 1]
8	H	[20 0 0 0]	19	S	[20 0 0 0]
9	I	[18 1 1 0]	20	T	[20 0 0 0]
10	J	[20 0 0 0]	21	U	[20 0 0 0]
11	K	[20 0 0 0]			

Now, we can represent an unlabeled modular graph by a configuration string, which is a sequence of all the nodes' *CN* values in the DFS order. The goal configuration is informed to every module in the form of the configuration string, and we call it the Configuration String of Goal (*CSG*), where *CSG*[*i*] is the *CN* value of the i^{th} node in the DFS order. From Table I, we know that *CSG* of Fig. 3(b) is

$$\begin{aligned} CSG = {} & [10\ 6\ 4\ 0][11\ 3\ 3\ 3][18\ 1\ 1\ 0][20\ 0\ 0\ 0][20\ 0\ 0\ 0] \\ & [18\ 1\ 1\ 0][20\ 0\ 0\ 0][20\ 0\ 0\ 0][18\ 1\ 1\ 0][20\ 0\ 0\ 0] \\ & [20\ 0\ 0\ 0][15\ 5\ 0\ 0][16\ 2\ 1\ 1][19\ 1\ 0\ 0][20\ 0\ 0\ 0] \\ & [20\ 0\ 0\ 0][20\ 0\ 0\ 0][17\ 1\ 1\ 1][20\ 0\ 0\ 0][20\ 0\ 0\ 0] \\ & [20\ 0\ 0\ 0] \end{aligned} \qquad (2)$$

B. Current Configuration Recognition

Besides being informed the *CSG*, the robot needs to recognize its current configuration so as to compare with the goal. It may involve redundant communication and heavy computation burden if having a single module sense the whole robot's configuration and do the comparison. Here we have the current configuration information distributed among all the modules in the way that every module explore their own *CN* value, and then collectively do the configuration comparison. *CN* exploration and comparison can be achieved by asynchronous communication with nearest neighbors.

Due to the constraint that the robot keeps connected throughout the reconfiguration process, the total number of modules included in the current configuration is the same as that in the goal configuration. So, every module can calculates the total number of modules in the current configuration from the given *CSG* according to equation (1) In our example, every module can use the first *CN*, [10 6 4 0], in the given *CSG* and get the Sum_Modules as

$$\text{Sum\_Modules} = 10+6+4+0+1= 21 \qquad (3)$$

After that, each module runs the same code to explore its *CN* value. A *probe* message is used, and the *probe* value received from a connector describes the number of modules connected through it. Whenever a module has received "*probe* = j" from its i^{th} connector, it will set its *CN*[i]=j. If its i^{th} connector has no connection, then its *CN*[i]=0. When a module has received p*robe* messages from all of its connected connectors except one (we call it connector k), it will set

$$CN[k]= \text{Sum\_Modules-1-} \sum_{i \neq k} CN[i] \qquad (4)$$

, and send out a *probe* message through connector k with value

$$probe= \sum_{i \neq k} CN[i] +1 \qquad (5)$$

Equation (4) (5) can be derived from equation (1).

Initially, only leaf nodes have only one connected connector that does not get any message, so they will initiate the *CN* exploration process by sending out "*probe*=1". Because all the links are half duplex, in the end there must one module that will receive messages from all its linked connectors. The *CN* exploration process ends then.

In our example, each leaf node in of Fig. 3(a) sends out the "*probe*=1" to the neighbor, and set *CN*= [21-1, 0, 0, 0]=[20, 0, 0, 0]. When module 4 has received two "*probe*=1" from module 5 and 6, it will send out "*probe*=1+1+1=3"to module 3, and set its *CN* value to be *CN*=[21-3, 1, 1,0]=[18,1,1,0]. All other modules will act in a similar way, and their explored *CN* values are shown in Table II.

TABLE II CN VALUES OF THE MODULES IN FIG 3(A)

Node ID	CN Value	Node ID	CN Value	Node ID	CN Value
1	[10 6 4 0]	8	[17 2 1 0]	15	[20 0 0 0]
2	[11 5 4 0]	9	[19 1 0 0]	16	[15 5 0 0]
3	[16 3 1 0]	10	[20 0 0 0]	17	[16 2 1 1]
4	[18 1 1 0]	11	[20 0 0 0]	18	[19 1 0 0]
5	[20 0 0 0]	12	[17 1 1 1]	19	[20 0 0 0]
6	[20 0 0 0]	13	[20 0 0 0]	20	[20 0 0 0]
7	[20 0 0 0]	14	[20 0 0 0]	21	[20 0 0 0]

C. Configuration Comparison

After knowing the *CN* value, the distributed configuration comparison is performed to find out the modules that indicate the different topology between current configuration and goal configuration. Each node will do the configuration comparison locally by comparing its *CN* value with the corresponding goal *CN* in the *CSG*, and decide whether it belongs to the *TBR* subgraph.

According to theorem 1, a node can decide whether it is the center node by checking its *CN*. If there is a pair of adjacent center nodes, they will negotiate to choose one as center. Duplex communication links avoid the case where two center nodes communicate simultaneously, which in some situation would result in both nodes designating themselves as center.

The local configuration comparison starts at the center node, and an *index* message is initiated by the center node and propagated to trigger the local configuration comparison of other modules. The *index* message sent to a node describes the location of its goal *CN* in the *CSG* string. The center node is matched with the corresponding center node in *CSG*, and herby its *index* equals to 1. Since the nodes in the *CSG* are in DFS order, the *index* message sent to a node also depends on its DFS traversal order rooted at the center node. So, if a node has its *index* equal to k, then the *index* message sent out from its ith connector, *index*[*i*], is

$$index[i]=k+\sum_{j=1}^{i-1} CN[j]+1 \tag{6}$$

If the goal *CN* , *CSG*[*index*], is the same as the module's current *CN* value, it means that the module shares the common topology with the goal configuration, so it will mark its *TBR* state as false, and send out the related *index* messages to its neighbors respectively. Otherwise, the module will be the root of the *TBR* subtree. Its *TBR* is set to be true, and this information is propagated to all its descendent modules to set their *TBR* as true. Please note that, since in our modular graph, module connectivity is abstracted as links without differentiating the connectors, the *CN* values are compared ignoring the order of their elements.

As an illustration, we will go through the configuration comparison process of Fig. 3(a). First, node 1 will find that it is the center because all the elements in its *CN*, [10 6 4 0], is less than half of the modules in the robot, i.e. 21/2. Then, it will compare its *CN* with the 1st *CN* in the *CSG*, [10 6 4 0]. Since they are the same, module 1 will set *TBR=false*, and send out *index* messages. According to equation (6), it will send out "*index*=1+0+1=2" through its 1st connector to module 2, "*index*=1+10+1=12" to module 16, and "*index*=1+10+6+1=18" to module 12. After receiving the *index* message, module 12 and module 16 will do the same comparison process as module 1. With regards to module 2, its *CN* value [11 5 4 0] is different from the 2nd *CN* in *CSG*, [11 3 3 3], so node 2 will be the root of the *TBR* subtree. It then set its *TBR=true*, and propagate this information to all the descendants of nodes 3~11. In the end, the *TBR* value is false for module 1, 12 ~ 21, and true for module 2 ~ 11. So the reconfiguration will be performed within the gray subgraph composed of module 2~11 in Fig. 3(a). Also, module 1 will ignore its connection with module 2 to exclude the *NTBR* part from the *TBR* subgraph in the reconfiguration process.

IV. Reconfiguration Action

This section describes the reconfiguration algorithm that rearranges the *TBR* subgraphs to convert the robot into the goal configuration. In our example, Fig 3(a) only has one *TBR* subgraph. If there are more than one *TBR* subgraphs, the modules in different *TBR* subgraphs can work simultaneously in parallel without interfering each other.

It is hard to have a general solution to convert an arbitrary graph to another directly. One possible solution is to use an intermediate structure [11]. Here, we first transform the *TBR* subgraph into a simple configuration, a line, and then convert this line to the corresponding parts in the goal configuration.

A. Reconfiguration from an acyclic graph to a line

In a line configuration, every node is connected to at most two nodes. So, the main idea of the algorithm is that every node with more than two branches will keep attaching a shorter branch to a longer one until the above requirement is satisfied. In the ends, each node will have at most two branches, and thus a line configuration is formed.

First, all the leaf nodes send out a *request* message to request the action of forming a line. When a node receives a *request* message, it will create a "parent-of" relationship towards the sender. A node with two neighbors will relay the received message, while a node with more than two connections will wait until it receives two *request* messages from two different connectors. It will then do a set of reconfiguration actions, which includes finding its two children braches according to the established "parent-of" relationship, merging one branch into the other by attaching the ends of the two branches and detaching the connection with the shorter branch, and clearing the *request* messages and "parent-children" relationship through all the nodes on newly generated branch. We do detaching after attaching to keep the robot connected all the time. During executing the set of above reconfiguration actions, all the modules on the two braches are locked temporary to exclude the interaction with other modules. In the end, stopping time is reached when a module with only one or two linked connectors has received messages from all of its linked connectors. It will then propagate the stopping message to all other modules.

The reconfiguration steps of converting the *TBR* subgraph in Fig. 3(a) into a line is shown in Fig. 4. One thing to note is that, as stated before, module 2 will ignore its connection with module 1 to exclude the modules in the *NTBR* subgraphs from reconfiguration. So, as shown in Fig. 4(e), when module 8 send r*equest* message to module 2, module 2 will ignore its link with module 1, and just relay the message to module 3.

We do not require the communication speed to be the same among all the modules. Every module can act asynchronously. The final topology is always a line, but the position of each module in the line is not deterministic. For example, in Fig. 4(d), if the message from module 2 reaches module 3 before module 4, then module 7 will be attached to module 11 instead of module 5, but the final configuration is still a line.

Our algorithm is also robust to the loss of message. For example, suppose that the message sent from module 4 to module 3 is lost in Fig.4(c), it can still complete the transformation into a line, as shown in Fig. 5.

B. Reconfiguration from a line to an acyclic graph

After converting the *TBR* subgraph into a line, the final

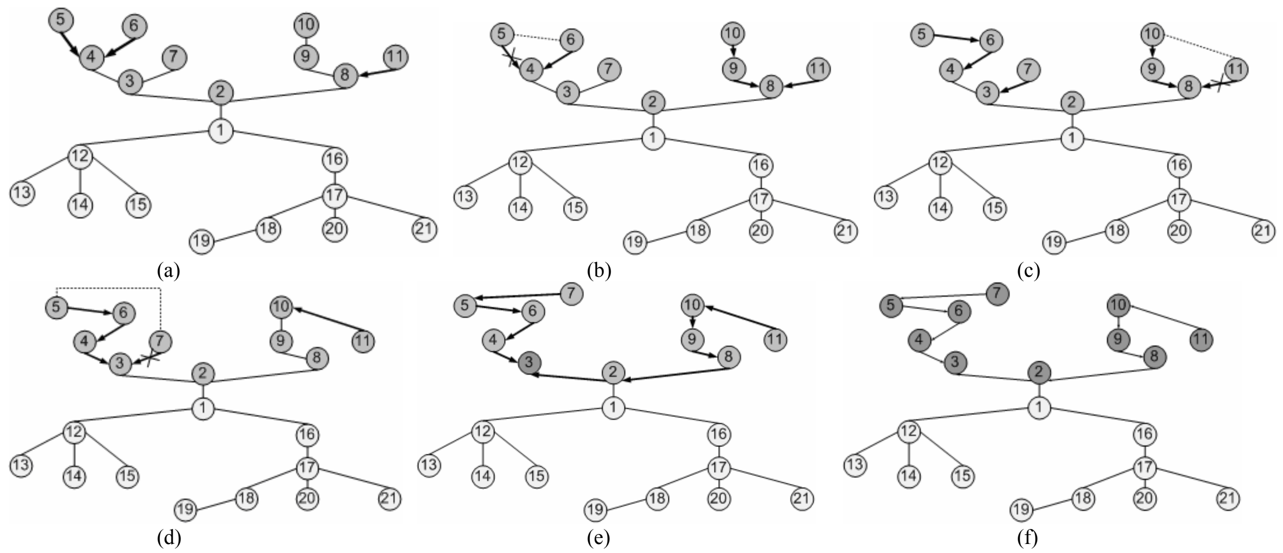

Fig. 4 reconfiguration steps of the TBR sub-graph in Fig. 3(a) to a line

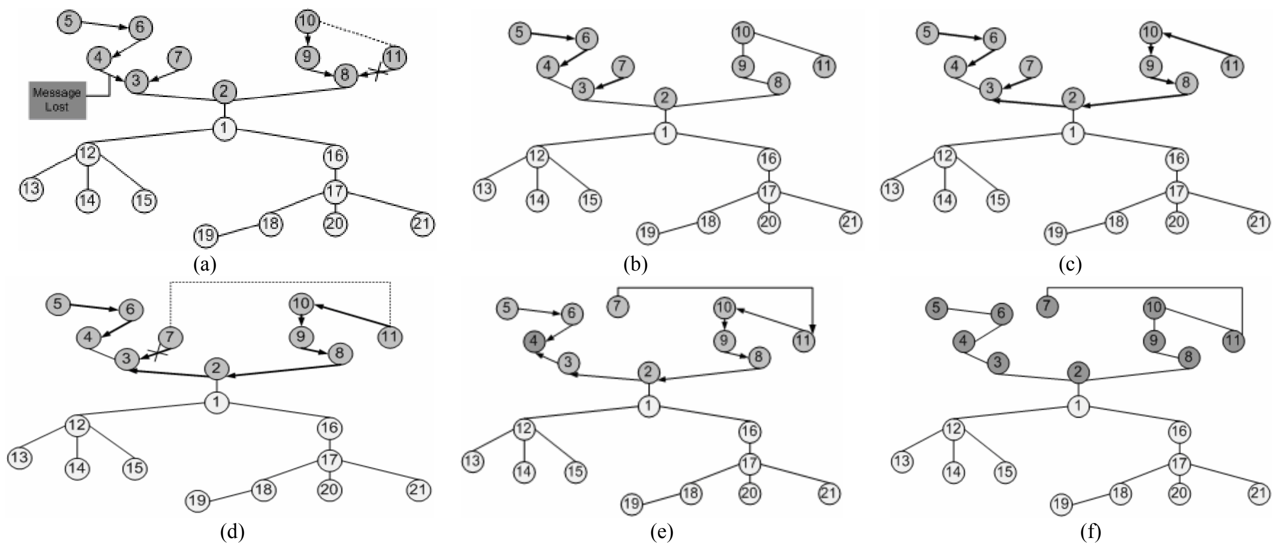

Fig. 5 Rreconfiguration steps from Fig. 4 (c) when a message is lost

step is to convert this line to the corresponding topology in the goal configuration.

The process is initiated from the root of the *TBR* subtree, which is determined in Section III-C. In our example, it is node 2. Upon receiving the *index* message, each module will rearrange its children branches to be consistent with the goal *CN* described by *CSG*[*index*], and send out *index* messages to its children respectively according to equation (6). For each module, the rearrangement of its descendents is achieved by having a line of *CN*[*i*] nodes connected to connector i for all the connectors. All the modules receiving *index* message can do the reconfiguration in parallel. The process ends when the *index* messages reach the leaf nodes. Fig. 6 shows the reconfiguration steps for from Fig. 4(f) to Fig. 3(b), where the numbers on the links are the value of *index* messages.

V. Discussion of The Algorithm

The reconfiguration algorithm proposed in this paper has the following properties that offer great advantages:

It is dynamic and on-line planning. Whenever unexpected change happens, or confronted with to a new environment, the robot can self-reconfigure without any prior knowledge of preplanned reconfiguration steps.

It is effort efficient. Due to the configuration comparison, substructures that have common topology with the goal configuration are extracted out, and reconfiguration changes are made only on the necessary parts. The configuration comparison is distributed without redundant communication and bottleneck. In the recognition of current configuration, *probe* message passes each link only once, while in the configuration comparison, the *index* message is also transmitted through every link only once. So, the communication complexity of the comparison is O(N).

It is scalable and time efficient. All the modules communicate and act locally under the same rule, independent of the size of the robot. The modules work asynchronously using only local information and local communication with neighbors. Many chains can reconfigure

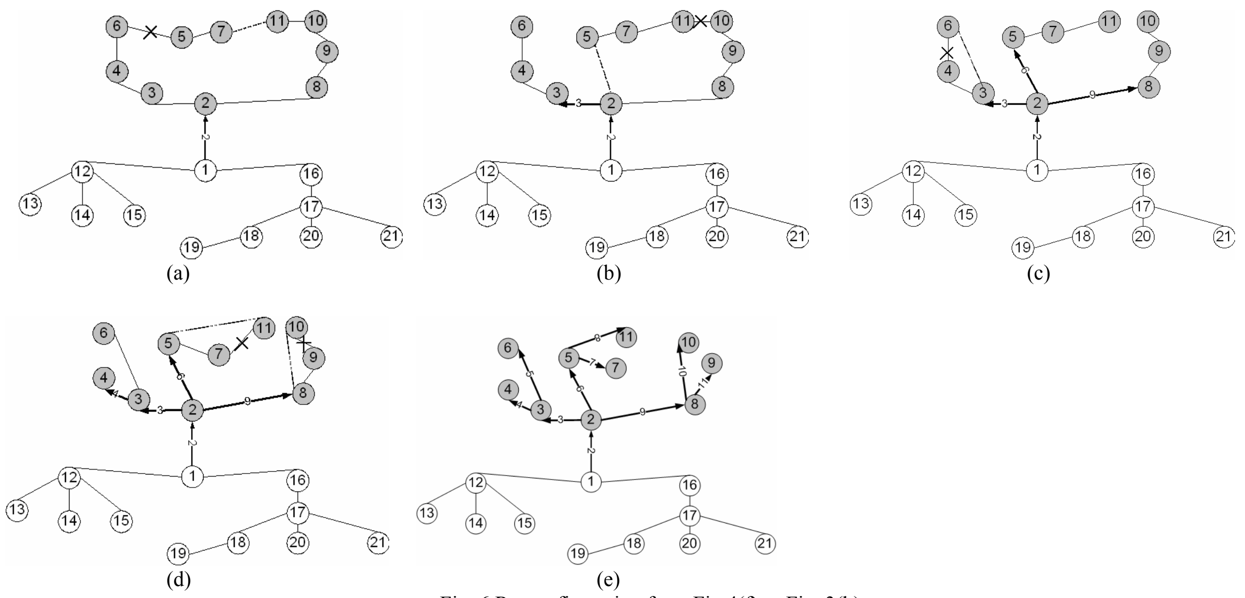
Fig. 6 Rreconfiguration from Fig 4(f) to Fig. 3(b)

simultaneously as long as they share no modules. The parallelism attribute allows for faster performance.

It is fault tolerant. Fig. 5 has demonstrated that it is robust to message loss to some extent. Moreover, modules are homogeneous that do not have IDs, and run the same program. There is not any predetermined special module. Whenever a module dies, we can just replace it by another one.

VI. Conclusion and Future Work

This paper has proposed a distributed and dynamic reconfiguration algorithm for chain-type self-reconfigurable robots to transform from one arbitrary acyclic configuration to another arbitrary one. The goal configuration is not generated by any pre-planned action steps, but emerges from the collaboration of all the modules in the robot. This allows the robot to reconfigure dynamically in unknown environments. The algorithm is not limited to the hardware design of any specific chain-type module, and is general for all chain-type reconfigurable robots.

One of our future works is to improve our configuration comparison algorithm for the situation where the center module has shifted in the goal configuration. In this paper, the intermediate structure is used and it is redundant in some cases, so a possible shortcut and more intelligent way for reconfiguration action will be explored. Another future work is to extend our modular graph to include the connector information. The robot with same configuration can function differently if the modules are connected in a different way. One more thing to do is to extend our algorithm to make it applicable to cyclic configuration, and implement it on some real and physical chain-type 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, Mark Yim, John W. Suh: A Complete, Local and Parallel Reconfiguration Algorithm for Cube Style Modular Robots. Proceedings of the IEEE International Conference on Robotics and Automation May 11-15; Washington; DC. NY: 2002; 117-122

[3] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa and S. Kokaji, "A Self-Reconfigurable Modular Robot: Reconfiguration Planning and Experiments", International Journal of Robotics Research, Vol. 21, No. 10, pp.903-916, 2003

[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] M. Yim, D. Duff, K. Roufas, "PolyBot: a Modular Reconfigurable Robot", Proc. of the IEEE Int. Conf. on Robotics and Automation, April 24-28; San Francisco, CA.

[8] 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.

[9] S. Murata, et al., "M-TRAN: Self-Reconfigurable Modular Robotic System," IEEE/ASME Trans. Mech. Vol.7, No.4, pp.431-441, 2002

[10] Behnam Salemi, Mark Moll, and Wei-Min Shen. SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System. In Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Beijing, China, October 2006

[11] Casal, A. Reconfiguration planning for modular self-reconfigurable robots. PhD dissertation; Stanford University. 2002

[12] Nelson, C. A., and Cipra, R. J., "An Algorithm for Efficient Self-Reconfiguration of Chain-Type Unit-Modular Robots," *ASME DETC'04*, Salt Lake City, Utah, September 28 - October 2, 2004, ASME Paper No. DETC2004-57488.

[13] Kenneth Payne, Behnam Salemi, Peter Will, and Wei-Min Shen. Sensor-Based Distributed Control for Chain-Typed Self-Reconfiguration. In Proc. 2004 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Sendai, Japan, Sept./Oct. 2004.

[14] Gregory L. McColm, On the structure of random unlabelled acyclic graphs, Discrete Mathematics, 2004