Connectivity Planning for Closed-Chain Reconfiguration

Mark Yim[a] David Goldberg[a] and Arancha Casal[ab]

[a]Xerox Palo Alto Research Center, Palo Alto, CA 94304
[b]Department of Computer Science, Stanford University, Stanford, CA 94305

ABSTRACT

Modular reconfigurable robots can change their connectivity from one arrangement to another. Performing this change involves a difficult planning problem. We study this problem by representing robot configurations as graphs, and giving an algorithm that can transform any configuration of a robot into any other in $O(\log n)$ steps. Here n is the number of modules which can attach to more than two other modules. We also show that $O(\log n)$ is best possible.

Keywords: modular, self-reconfigurable, robot, reconfiguration planning

1. INTRODUCTION

Unit-modular and bi-unit-modular self-reconfigurable robots are made up of one or two types of repeated modules, each with limited ability to connect, disconnect, and move relative to its neighbors. Most notably the system can rearrange its modules to form new connectivity arrangements. For example, a robot with modules arranged in the form of a two legged robot can rearrange its own modules to form a six-legged robot. Some of the advantages of these systems include versatility from reconfigurability, robustness from redundancy and low cost from batch fabrication.[1,2]

There are many instances of these types of robots.[1–12] These systems can be categorized into three classes based on the mode of reconfiguration as introduced in Ref. 13. One is called *mobile* reconfiguration in which modules may separate and move on the environment to reconfigure, for example, many mobile robots which independently attach together to form trains.[14] Another involves modules that move on each other and locally disconnect and connect to a discrete set of neighboring modules all of which sit on a lattice. This class is referred to as *lattice* or *substrate* reconfiguration. The final class is *closed-chain* reconfiguration in which modules attach and detach to non-local modules. These are the focus of this paper.

The motions for closed-chain reconfiguration are characterized by chains of modules emanating from the robot that attach to other parts forming closed-chains. These closed-chains then detach from another location, effectively moving the chain to a different location on the robot. One of the constraints of these systems is that all the modules in the system need to remain one connected component. This constraint is often required for power distribution and inter-module communication.[15,16]

Many useful tasks can be achieved from a handful of configurations that are identified manually. The sequence of actions (connects, disconnects, and motions of degrees of freedom) to reconfigure a system from one connectivity arrangement to another can sometimes be determined manually for systems with small numbers of modules.[1] As the number of modules increase, the probability of a module failing increases. If these systems are to be robust, reconfiguring themselves to compensate for failing modules, it is difficult to predict every possible connectivity arrangement a system may take. In fact, the number of connectivity arrangements of these systems grows exponentially with the number of modules and is difficult to enumerate exactly.[17] So, to truly exploit the versatility and robustness afforded by the reconfiguration ability, an automatic process for determining the sequence of actions to reconfigure is needed.

On most modular systems, there are computational elements on every module. We will assume that each module has a processor and that the processors can communicate with neighboring modules. Computation can be achieved in a distributed manner, with message passing utilizing the connectivity structure of the system, or this distributed nature can easily be simulated for centralized computational architectures.

Further author information: (Send correspondence to M.Y.)
M.Y: E-mail: yim@parc.xerox.com

1.1. Robots that use Closed-Chain Reconfiguration

Currently existing robot systems that use the closed-chain reconfiguration include the CEBOT, CONRO, Polypod and PolyBot. Fukuda introduced CEBOT[3] which is a heterogenous modular reconfigurable system that combined mobile modules as well as manipulators.[18] Will et al. introduced CONRO,[12] a self-sufficient unit-modular reconfigurable system. Yim introduced Polypod[19] showing a variety of locomotion modes. The PolyBot system, Polypod's successor, currently has up to 24 modules in simple configurations, however in 2001 it is expected to have hundreds of modules with a large number of possible configurations.

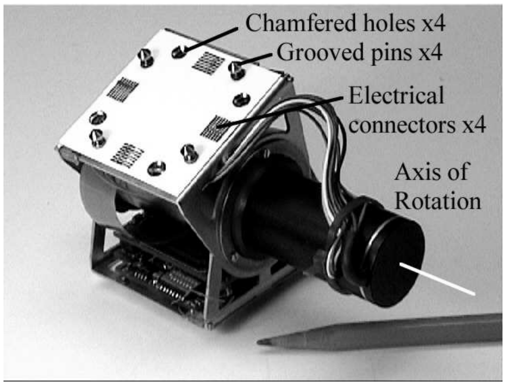


Figure 1. A PolyBot segment showing a connection port. A node is made of 6 ports in a cube shape.

The PolyBot system is composed of two types of modules. These systems will be used as examples throughout the paper. PolyBot has a one degree of freedom module called a *segment* which has two connection ports and is shown in Figure 1, and a zero degree of freedom cube-shaped module called a *node* which has six connection ports. Two modules mate by joining at their connection ports. The ports on PolyBot are hermaphroditic however, in general ports may have a gender. Since segments have only two connection ports, a robot consisting of segments alone could only be connected to form chains. By adding nodes, PolyBot can approximate a much larger variety of shapes. Figure 2 shows photos of the system reconfiguring between a loop, a snake and a 4 legged spider.

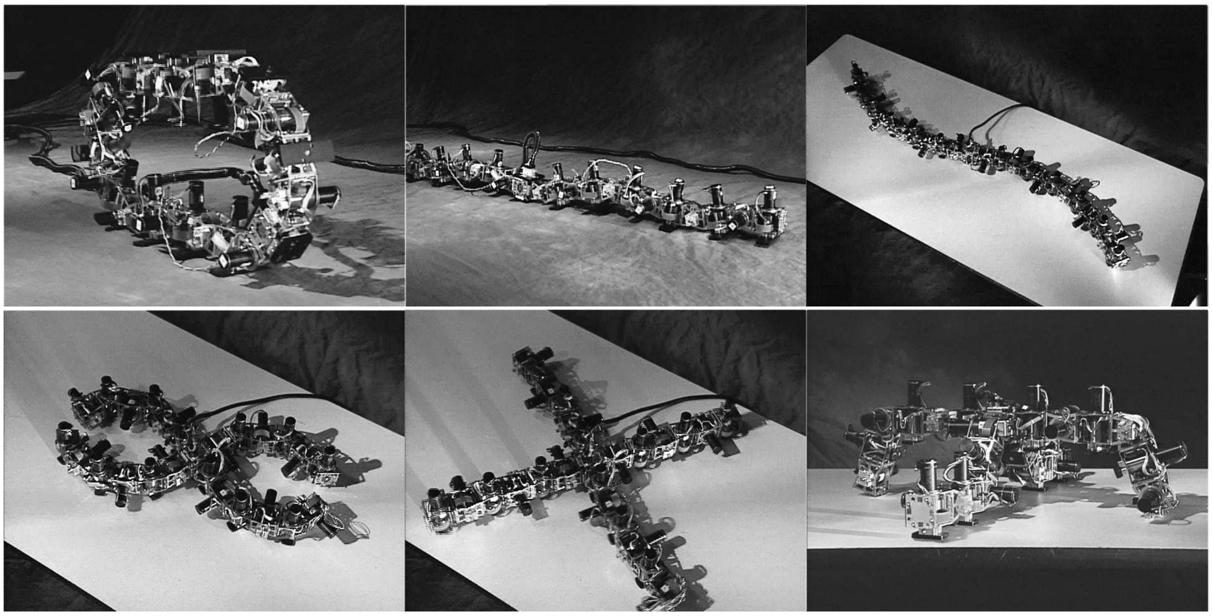


Figure 2. PolyBot in a loop, then opening to a snake shape then an 'S' shape. After reconfiguring from the 'S' to a figure–8, then opening to cross, it stands up in a four-legged spider stance.

1.2. The Reconfiguration Problem

The shape of a bi-unit modular robot that uses closed-chain reconfiguration can be defined in two independent parts. The first part is the connectivity arrangement of the modules, defined by which modules are physically attached to which. In order to change the connectivity, a set of modules must attach and detach from others. The second part is the position of the internal degrees of freedom (joint angles) for a given connectivity arrangement.

A reconfiguration plan includes the sequence of connectivity changes as well as the motion plan for the internal degrees of freedom as needed. Taken together, this is a complex hybrid problem that is very difficult. One way to approach the problem is to have a connectivity planner at a high level generate the sequence of connectivity changes and make calls to a lower level motion planner that generates the physical motion plan, or fails and causes the higher level to replan. This paper addresses the connectivity planning problem.

The motion planning problem is a difficult one considering the numbers of degrees of freedom and the possibility of non-serial chain configurations. There are examples of progress in this area, most notably in randomized planning.[20,21] In addition a motion planner should account for self-collisions, stability under gravity and inverse kinematics among others. We are currently working on integration of these techniques with the connectivity planner.

In order to study the reconfiguration process, we need a mathematical model of our robot. We want a model general enough that it doesn't depend on implementation details that vary from one model of robot to another, but specific enough to be realistic. In this paper, we consider a very general model. It is perhaps too simple to be fully realistic, but we feel it is complex enough to shed light on the closed-chain reconfiguration problem.

1.3. Related Work

There is a growing literature on planning for the lattice class of reconfigurable robots.[2,7,9–11,22,23] However, the closed-chain class has received less attention. Casal et al. addressed closed-chain reconfiguration[13] by decomposing a graph representation of the robot into a hierarchy of well known "substructures", and restricting reconfiguration to occur among these substructures. However, no analysis was made of the number steps required for reconfiguration nor of the running time. Several other concepts for reconfiguration algorithms were also introduced in that paper, including a canonical form algorithm. This algorithm is more formally presented here along with analysis of the number of steps it requires.

There are many algorithms for the manipulation of trees. For example, tree contraction[24] is an algorithm for solving a wide variety of problems on a parallel random access machines (PRAM). A typical application is for expression evaluation. The attach and detach operations of different modules in parallel during reconfiguration are analagous to parallel computations on different processors in a PRAM. While the graph manipulations used here may seem similar to these other algorithms, the fact that the graph represents a physical system makes it very different. For example, the tree contraction algorithm utilizes a "SHUNT" operation in which the leaves together with one parent vertex are removed by manipulating the contents of the grandparent and edges. Since in our problem, some components of the graph correspond to physical objects, they cannot simply disappear.

2. GRAPH REPRESENTATION

The modules of a bi-unit modular system and its connectivity can be represented by a graph. As in PolyBot we will use the node and the segment as different elements in the graph. We will assume that each node has the same number of ports p, where $p \geq 3$. The segment has some internal degrees of freedom with a port at both ends. The robot can be reconfigured by various attaching and detaching operations. An attaching operation consists of attaching two segments at a port, attaching up to p segments into the ports of a node, or by attaching two nodes at a port. The detaching operations are the reverse of these, and break at a port.

2.1. The Model

Our model will represent a robot connectivity arrangement as a graph with labeled edges and vertices.

We represent a segment by a white edge with a white vertex at each end (think of the vertices as representing the ports). A node is represented by a black vertex. To represent two directly attached nodes, we connect them with a black edge. So there are black and white vertices, and black and white edges. As usual, the degree of a vertex is the number of edges (black or white) incident on it. A given physical robot with n nodes and k segments would be represented by n black vertices and k white edges.

The set of graphs representing a robot that uses closed-chain reconfiguration is described in two stages. First we give a recursive definition of a *generalized robot connectivity graph* (GRCG). These are graphs that may have more than one connected component and could therefore represent a set of robots that use mobile reconfiguration. A robot that uses closed-chain reconfiguration needs to remain as one connected component, so we define a robot connectivity graph (RCG) as a connected GRCG.

It is easiest to describe the set of GRCGs using recursive production rules, just as is often done to describe the syntax of a programming language. Here are the five rules for a GRCG:

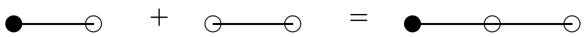
1. One or more white edges, each with a white vertex at both ends form a GRCG. White edges and vertices are drawn like this (where a white edge is drawn as a thin black line):

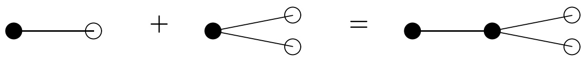
2. One or more black vertices form a GRCG. They are drawn like this:

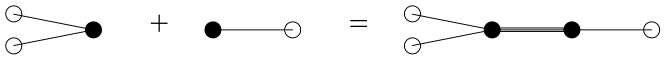
3. Combining two white vertices of a GRCG forms a GRCG, provided both white vertices have degree one and they are not already connected by a white edge. Here is an example:

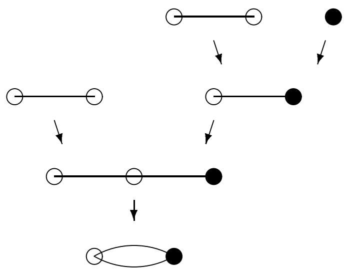
4. Combining a white and black vertex of a GRCG, and replacing it with a black vertex forms a GRCG, provided the white vertex has degree one, the black vertex has degree $p-1$ or less, and they are not already connected by an edge. Here is an example:

5. Linking two black vertices of a GRCG with a black edge forms a GRCG, provided each of the vertices have degree $p-1$ or less, and they are not already linked by a black edge. Here is an example (a black edge is drawn as at thick black line):

From these rules it follows that in a GRCG, each white vertex has degree two or less, and each black vertex has degree p or less. The tree below is an example of building up a GRCG step by step using the production rules

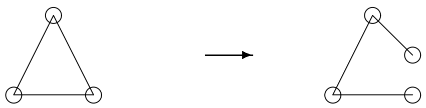


2.2. Reconfiguration

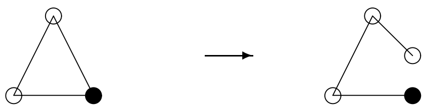
Now that we have a precise definition of a robot's connectivity (namely an RCG), we can begin to study the reconfiguration process. We would like to know when one RCG can be tranformed into another, the minimum number of steps it will take, and a practical algorithm to achieve the reconfiguration.

To answer these questions, we need to define what we mean by a *step*. A step is built out of attaching and detaching operations. There are three attaching operations, which correspond exactly to production rules 3, 4 and 5 given above. Similarly, there are three detaching operations, namely

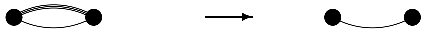
1. Detach one of the edges from a white vertex of degree two, forming two white vertices of degree one.



2. Detach a white edge from a black vertex, and add a new white vertex:



3. Detach two black vertices by removing a black edge.



A step is a set of attach or detach operations done in parallel that don't interfere with each other, and don't split the graph into two pieces. The easist way of stating this precisely is to consider two types of steps. A detaching step is a set of detaching operations done in parallel, and an attaching step is a set of attaching operations done in parallel. We require that the graph be connected at the end of a step (both at the end of a attaching step and a detaching step). Reconfiguration is performed in a series of steps, each of which can be either an attaching step or a detaching step.

Note that a graph G_1 can be reconfigured to a different graph G_2 only if they have the same number of black vertices and white edges, since these are conserved by all the attaching and detaching operations. This is also a requirement for a physical system with a fixed number of modules as black vertices correspond to nodes and white edges to segments.

We can now precisely state how many steps are required to reconfigure G_1 to G_2.

THEOREM 2.1. *Given two RCGs with k white edges and n black vertices, one can be reconfigured into the other in $O(\log n)$ steps. This is best possible in that there are two graphs with n black vertices that require at least $O(\log n)$ steps.*

First, we will show that there are two graphs G_1 and G_2 that require $O(\log n)$ steps to reconfigure one to the other. The graph G_1 is a row of n black vertices (connected by black edges) followed by a white chain of $n-1$ white vertices with $n-1$ white edges. We refer to a white *chain* or a chain of white edges as a set of white edges connected only by white vertices.

Graph G_2, consists of n black vertices, each connected by a white edge.

If we can prove the claim,

> *If a graph has a linear chain of k white edges, after a detaching step it must have a chain of at least $\lceil k/2 \rceil$ white edges.*

then it will follow that it takes $O(\log n)$ steps to go from G_1 to G_2 since G_1 has a chain of $n-1$ white edges, while the longest chain of white edges in G_2 has length one.

The only way a chain of white edges can be broken up in a detaching step is by detaching a white vertex of degree two. If a chain were to detach at more than one place in one step, the graph would become disconnected. So at most one vertex can be detached, and the k-long chain can separate into at most two pieces, one of which must have length at least $\lceil k/2 \rceil$.

Can an attaching step spoil the white chain by attaching to the middle of it? That is not possible, because each white vertex in the interior of the chain already has degree two, and so nothing can be added to it.

To complete the proof of the theorem, we give an algorithm that converts any RCG into any other in $O(\log n)$ steps where n is the number of black vertices.

3. THE CANONICAL FORM ALGORITHM

This algorithm works by converting any RCG, G, into an intermediate simple form, the *canonical* form. Then G_1 can be reconfigured to G_2 by converting G_1 to this canonical form, followed by reversing the steps required to convert G_2 to canonical form.

The canonical form is a linear chain consisting of all the black vertices (connected by black edges) followed by all the white edges.

The algorithm consists of the following steps:

1. **Convert to Tree** Perform a detaching step that converts the graph to a tree. For example, using a breadth-first search.
2. **Number leaf vertices** Number the leaf vertices (whether black or white) 0 to $f-1$, for f leaf vertices
3. **Form linear graph**

   In the loop below, the operations within each step are all performed in parallel (e.g. in the first step, all the even-numbered leaves send messages in parallel).

   while more than one leaf vertex **do**

   - Have each leaf numbered $2k$ send a message up the tree until it reaches a black vertex of degree three or more. Mark that vertex, and the edge leading up to it. (If the highest-numbered leaf is even, it does none of this except the final renumbering step).
   - Attach each leaf vertex $2k$ to vertex $2k+1$
   - Detach the previously marked edge from the marked vertex.
   - Renumber, changing leaf vertex $2k$ to k.

   end loop
4. **Number black vertices** Number the black vertices in linear chain order from 0 to $n-1$
5. **Detach white side chains**

   - In parallel, attach black vertex k to $k+1$ by adding a black edge between each.
   - In parallel, **if** a chain of white edges connects adjacent black vertices k and $k+1$ **then** detach the junction of the white edge with black vertex $k+1$
   - If vertex 0 was not the beginning of the linear graph, then it may have two white chains attached to it. If so, then merge them by joining their ends and then doing one detach operation.
6. **Coalesce white side chains** Number the white chains from 0 to $m-1$.

   while more than one side chain remains **do**

- Send a message from the end of the chain numbered $2k$ until it reaches a black vertex, which will have degree three or more. Mark the black vertex and the last edge leading up to it. (if the final chain has an even number, it does none of this except the final renumbering step).
- Attach the end of the chain labeled $2k$ to $2k+1$
- Detach the previously marked edge and vertex
- Renumber, changing chain $2k$ to k.

end loop

The graph is now in canonical form. An example of a 24 step reconfiguration connectivity sequence is shown in Figure 3 for a system with 4 nodes and 11 segments.

3.1. Analysis

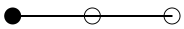
To find the order of the number of steps in this algorithm, we only need to analyze the two **while** loops. The second one clearly has $O(\log n)$ steps since it is a binary merge of n white chains. The first loop is slightly more complicated.

A path from the root to a leaf of the tree will contain s *branching* vertices, that is, vertices of degree three or more. Call the depth d of the tree the maximum value of s computed over all leaves. Since every vertex has maximum degree p, and every iteration of the first while loop merges half the chains, after $\log_2 p$ iterations the depth of the tree will be reduced by at least one. Each iteration has one attach step and one detach step, so it will take at most $2\log_2 p \times d$ steps to reduce the tree to depth 0, which will be a linear graph.

Since white nodes have degree two or less, only a black node can be a branching vertex. Thus the depth of the tree is $O(\log n)$, and the number of steps it takes to linearize is also $O(\log n)$.

4. EXTENSIONS

While the model presented uses examples of Polypod and PolyBot, it is easily extended to a variety of other bi-unit-modular and unit-modular robots. Through grouping elements as metamodules[25] we can form the modules of many other systems. For example, the unit-modular CONRO[12] modules could be represented by a 4 connection port node (black vertex) with a chain of two segments (white edges) attached to one port with the other end free.



In one hardware design of CONRO,[26] the connection ports are not hermaphroditic, the white vertex is female and the black vertex has three male ports. As such, every connected component of n modules has at most one free female port; every added module to a group uses one female port, and only adds one female port). So, the most attach and detach operations that can happen in parallel in one step is one. This means the closed-chain reconfiguration for a worst case takes $O(n)$ steps. However, this system is not constrained to maintain one connected component.

4.1. Tree Inversion

Clearly, the sequence of steps generated by the canonical form algorithm for two arbitrary RCG's is unlikely to be the minimum number since it is unlikely that the canonical form will be an intermediate RCG in an optimal sequence of RCGs. Here we briefly introduce the tree-inversion concept as a way of reducing the number of steps or in the worst case, the algorithm reduces to the canonical form algorithm.

This method is based on the observation that white chains must first have one end free before they can attach to a new location. We call chains with one end free, external chains, while chains terminated with black vertices on both sides are called internal chains.

Under ideal conditions the following tree inversion algorithm generates a sequence of four alternating detach/attach steps. The five procedures are shown below for reconfiguring between two RCGs G_1 and G_2.

1. Convert the initial RCG, G_1 to a tree. For example a breadth first traversal generates the first step, a detach step.
2. Similarly the reverse of the fourth step (an attach step) converts the final RCG G_2 to a tree.

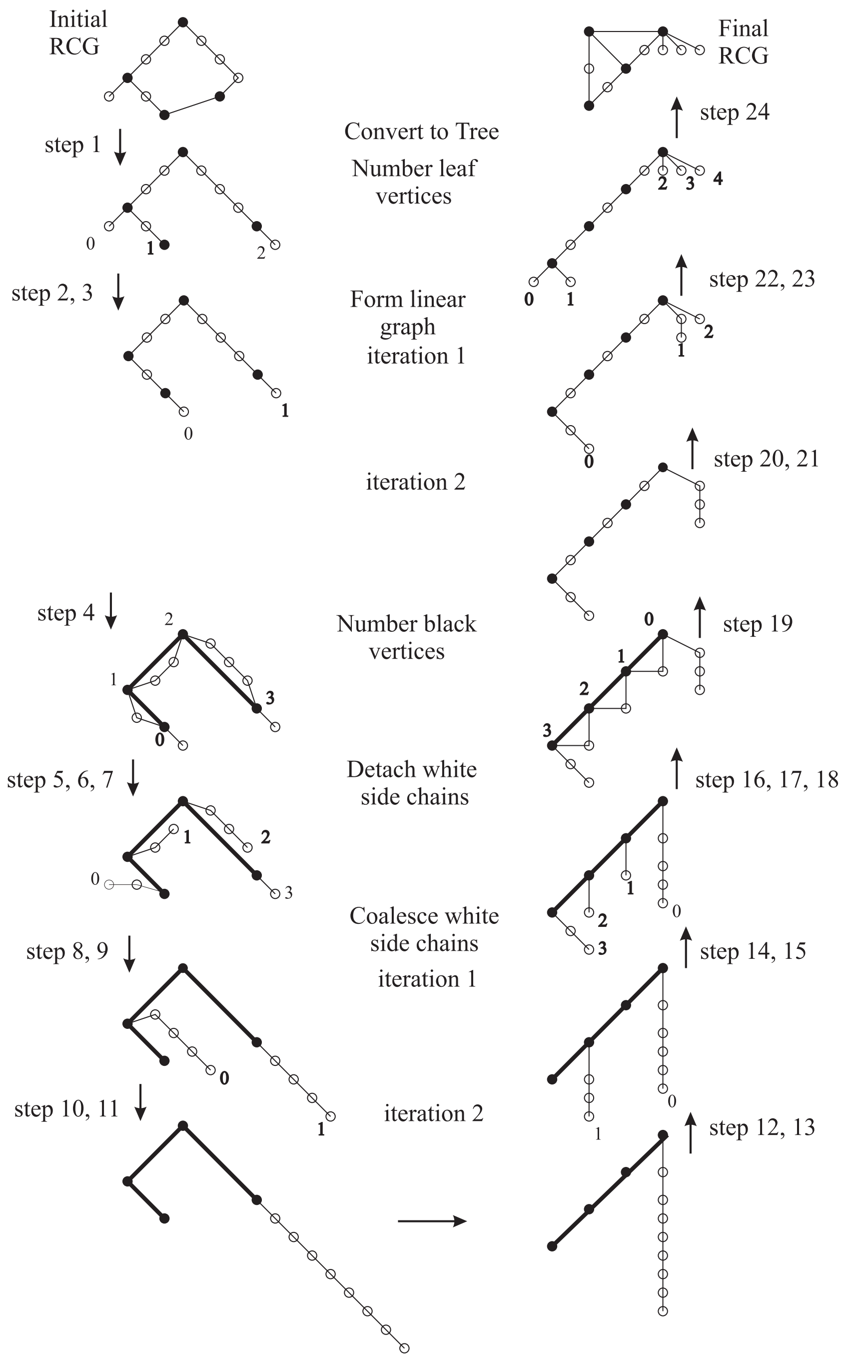


Figure 3. Example stages of canonical form algorithm.

3. The algorithm then matches external chains on G_1 with internal chains on G_2.
4. The second step in the sequence is an attach step attaching the external chains of G_1 to the corresponding black vertices to form the internal chains in G_2.
5. A detach step releasing all the internal chains of G_1 to form external chains on G_2 is the third step and the plan is complete.

Procedures 3-5 are called the tree inversion since internal and external chains swap places. In most cases, procedure 3 and 5 fail; there is not a one-to-one correspondence between external and internal chains. There may not be the same number of chains, and even if there were, the lengths of the chains may not match.

To make procedure 3 work, the external chains on G_1 must first be rearranged to map to the internal chains on G_2. This rearrangment can be done in $O(\log(m-n))$ steps where m and n are the number of external chains of G_1 and the internal chains of G_2 using straight forward modifications to the coalescing white sides procedures in the canonical form algorithm.

Also for procedure 5, the newly formed external chains (formerly internal G_1 chains) need to be arranged to match the external chains of G_2. Again the same techniques used for the external chains of G_1 can be used generating $O(\log(l-k))$ steps, where l and k are the number of external chains of G_2 and the internal chains of G_1.

A procedure must also be inserted before the tree inversion procedures to ensure that the degree of the initial and final trees generated after procedure 2 are both under $p/2$ where p is the maximum number of ports on the node. In at most four steps, the number of external chains can be reduced by at least half, by coalescing. In parallel with this, the $n-1$ internal chains can be redistributed to reduce the highest degree vertices.

Figure 4 shows the stages for the same input as Figure 3 using tree inversion. The details of the algorithm is difficult to explain and analyze given the nature of the algorithm and the space constraints of this paper. However, the algorithm has been implemented and in most cases, the number of steps generated by this algorithm will be fewer than the canonical form algorithm, although the running time will usually be longer.

5. CONCLUSION

We have introduced a mathematical model for the connectivity of modular reconfigurable robots and for closed-chain reconfiguration of these robots. This model is general and useful enough for studying the reconfiguration process of many unit-modular or bi-unit modular systems.

Using this model, we presented a connectivity planning algorithm using canonical forms which outputs a sequence of attach and detach steps that result in the desired goal connectivity arrangement. Our algorithm generates $O(\log n)$ steps which is best possible. We also presented the idea of tree inversion, which may be superior to the canonical form algorithm for many graphs.

To be able to implement the generated sequence in a real robot, each connectivity step must include a physically feasible tragectory. We are in the process of integrating a motion planner based on the probablistic roadmap method[20] to generate trajectories.

Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract #MDA972-98-C-0009. The authors would also like to acknowledge the help of Marshall Bern.

REFERENCES

1. Yim, M., Duff, D.G., and Roufas, K.D., "PolyBot: a modular reconfigurable robot," in *Proc. of the 1999 IEEE International Conf. on Robotics & Automation, May 2000.*
2. Murata, S., Kurokawa, H. and Kokaji, S., "Self-assembling machine", in *Proc. of the 1994 IEEE International Conf. on Robotics & Automation,* pp. 441–448, 1994.
3. Fukuda, T. and Nakagawa, S., "Dynamically reconfigurable robotic system," in *Proc. IEEE International. Conf. Robotics and Automation*, pp 1581-1586, 1988.
4. Chirikjian G., "Metamorphic hyper-redundant manipulators," in *Proc. JSME International Conf. on Advanced Mechatronics*, pp. 467–472, 1993.
5. Yim, M., *Locomotion with a unit-modular reconfigurable robot*, Stanford University PhD Thesis, 1994.
6. Pamecha, A., Chiang, C-J., Stein, D., and Chirikjian, G., "Design and implementation of metamorphic robots," in *Proceedings of the 1996 ASME Design Engineering Technical Conf. and Computers in Engineering Conf., Aug 1996.*
7. Kotay, K., Rus, D., Vona, M., and McGray, C. "The Self-Reconfiguring Robotic Module: Design and Control Algorithms," *Robotics: The Algorithmic Perspective, 1998 WAFR*, Agarwal, P.K., Kavraki, L.E., and Mason, M.T. (eds), A K Peters, Natick, MA, pp. 375-386. 1998.

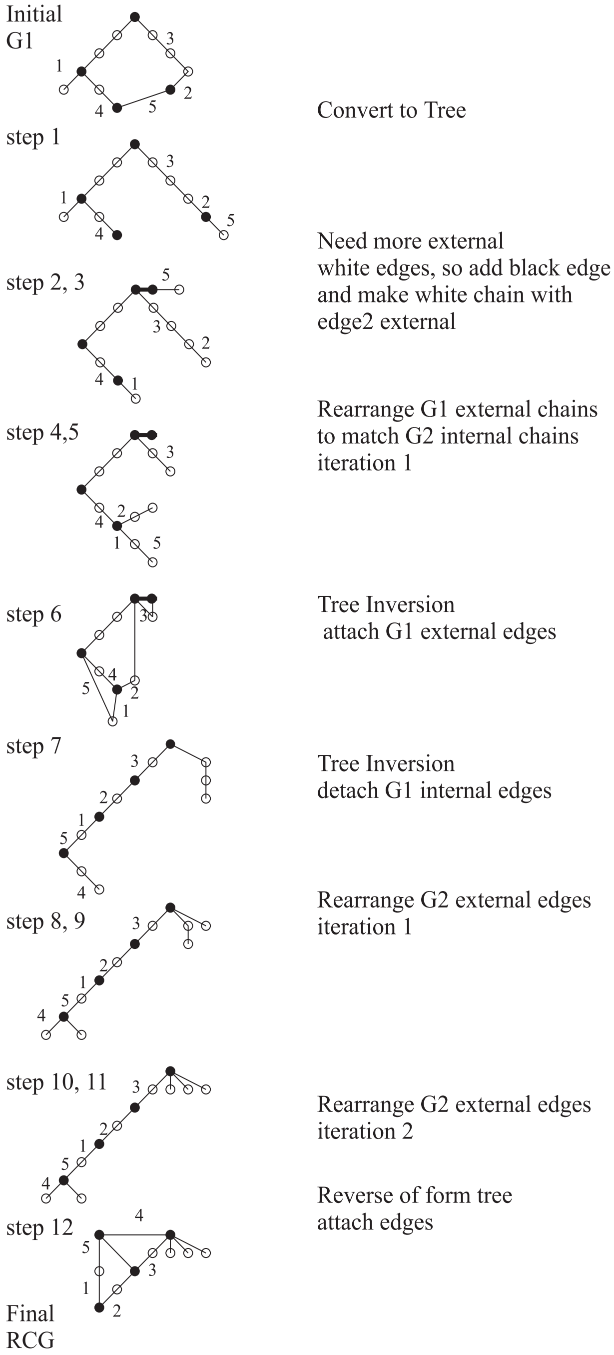


Figure 4. Example stages of the tree inversion algorithm.

8. Hosokawa, K., Tsujimori, T., Fuji, T., Kaetsu, H., Asama, H., Kuroda, Y., and Endo, I., "Self-organizing collective robots with morphogenesis in a vertical plane." in *Proc. of the 1998 IEEE International Conf. on Robotics & Automation,* pp. 2858–2863, 1998.
9. Murata, S., Kurokawa, H., Yoshida, E., Tomita, K., and Kokaji, S., "A 3-D self-reconfigurable structure," in *Proc. IEEE International Conf. Robotics & Automation*, pp. 432–439, 1998.
10. Rus, D. and Vona, M., "Self-reconfiguration planning with compressible unit modules," in *Proc. of the 1999 IEEE International Conf. on Robotics & Automation,* May 1999.

11. Yim, M., Zhang, Y., Lamping J.,O., Mao, E., "Distributed control for 3D shape metamorphosis," *Autonomous Robots Journal, special issue on self-reconfigurable robots*, to appear.
12. Will, P., Castano, A. and Shen W-M., "Robot modularity for self-reconfiguration," in *Proc. of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II,* Vol. **3839** pp. 236–245 1999.
13. Casal, A. and Yim, M., "Self-reconfiguration planning for a class of modular robots," *Proc. of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II,* vol. **3839** Sept. 1999.
14. Fukuda, T., Nakagawa, S., Kawauchi, Y., and Buss, M., "Structure decision method for self organising robots based on cell structures-CEBOT", *Proc. of the 1989 IEEE International Conf. on Robotics & Automation,* 1989.
15. Yim, M., Lamping, J., Mao, E. and Chase, J., "Rhombic dodecahedron shape for self-assembling robots," Xerox PARC, SPL TechReport P9710777, 1997.
16. Chirikjian, G., "Kinematics of a metamorphic robotic system," in *Proc. of the 1994 IEEE International Conf. on Robotics & Automation,* 1994.
17. Chen, I. M., *Theory and Applications of Modular Reconfigurable Robotics Systems,* PhD Dissertation, Division of Engineering and Applied Science, California Institute of Technology, 1994.
18. Fukuda., T, and Kawauchi, Y., "Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator," in *Proc. of the 1990 IEEE Conf. on Robotics & Automation*, pp. 662–667. 1990
19. Yim, M., "A reconfigurable modular robot with multiple modes of locomotion," in *Proc. of the 1993 JSME Conf. on Advanced Mechatronics*, pp. 283–288, 1993.
20. Kavraki, L. E., Svestka, P., Latombe, J. C. and Overmars, M., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE Transactions on Robotics and Automation*, **12**(4), pp. 566–580, 1996.
21. Han, L. and Amato, N. M.,"A kinematics-based probabilistic roadmap method for closed chain systems," in *Proc. of the Workshop on Algorithmic Foundations of Robotics (WAFR'00)*, March 2000.
22. Pamecha, A. and Chirikjian, G., "Useful metrics for modular robot motion planning," in *IEEE Transactions on Robotics and Automation,* Vol. **13**, No 4, August 1997.
23. Chirikjian, G., Pamecha, A. and Ebert-Uphoff. I., "Evaluating efficiency of self-reconfiguration in a class of modular robots," in *Journal of Robotic Systems,* pp. 317–338, 1996.
24. Miller, G. L., and Reif, J. H., " Parallel tree contraction and its application," in *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science* pp. 487–489, 1985.
25. Nguyen, A., Guibas L.J., Yim, M., "Controlled module density helps reconfiguration planning," in *Proc. of the Workshop on Algorithmic Foundations of Robotics (WAFR'00)*, March 2000.
26. Castano, A. and Will, P. "Mechanical design of a module for autonomous reconfigurable robots," *Proc. IEEE/RSJ International Conf. Intelligent Robots Systems, (IROS)* 2000.