Graph signature for self-reconfiguration planning of modules with symmetry

Masoud Asadpour, Mohammad Hassan Zokaei Ashtiani, Alexander Sproewitz, Auke Ijspeert, *Members, IEEE*

***Abstract*— In our previous works we had developed a framework for self-reconfiguration planning based on graph signature and graph edit-distance. The graph signature is a fast isomorphism test between different configurations and the graph edit-distance is a similarity metric. But the algorithm is not suitable for modules with symmetry. In this paper we improve the algorithm in order to deal with symmetric modules. Also, we present a new heuristic function to guide the search strategy by penalizing the solutions with more number of actions. The simulation results show the new algorithm not only deals with symmetric modules successfully but also finds better solutions in a shorter time.**

I. INTRODUCTION

Modular robotics is an approach to build robots with complex structures by connecting individual simple robots together. Beside ease of mass-producing the simple modules, self-reconfigurable modular robots have the ability to reconfigure from a configuration of modules to another one. These characteristics make modular robots more adaptive and robust than conventional robots in different circumstances. For example, such a robot can switch between snake and spider shapes to move in narrow or uneven paths. At the other hand, the only thing should be done to repair a modular robot is to replace an out-of-work module with a working one.

Modular robots are generally classified as *lattice-type* or *chain-type*. Lattice-type modules use cluster-flow locomotion and reconfiguration. In order to move, the robot continuously reconfigures (modules attach and detach over a lattice of other modules), thereby giving the impression that the cluster “flows” on the ground and around obstacles. Crystalline [3], Telecube [4], ATRON [5] and Molecule [6] modules use this type of reconfiguration.

Chain-type reconfiguration is similar to substrate reconfiguration except that modules of this type have power joints which enable them to locomote without the need of reconfiguration. Reconfiguration is usually used to adapt to a new environment or task. M-TRAN [8], YaMoR [9], CONRO [10], Polybot [11] and Molecube [13] are some implementations of this type. We work on this type of modular robots.

In this work, we use a framework based on graph signature and graph edit distance introduced by Asadpour et al. [2] to tackle the problem of Self-Reconfiguration Planning (SRP). We make a big improvement to the algorithm by introducing a way to deal with symmetric modules in an efficient way. We also introduce a new heuristic function for the search strategy that enhances the algorithm by finding better solutions with less computational cost.

The next section describes the related works. The third section explains our new method. The paper is finalized by the simulation results and conclusions.

M.Asadpour and M.H Zokaei Ashtiani are with Control and Intelligent Processing Center of Excellence, ECE Dept., University of Tehran, Iran. Corresponding author: asadpour@ut.ac.ir, +98-21-82084951

A.Sproewitz and A.Ijspeert are with Bioinspired Robotics Group (BIRG) at Echole Polytechnic Federal de Lausanne (EPFL), Switzerland.

II. RELATED WORKS

A *configuration* is a particular arrangement of connectivity between independent modules [1]. Self-reconfigurable modular robots must have the ability to plan a series of atomic actions to reach some configuration in *configuration space*. SRP addresses the design of an efficient algorithm to find an optimal (or suboptimal) sequence of predefined actions to reach a final configuration, starting from an initial one.

Mechanical limitations put some difficulties upon SRP for chain-type robots. Individual modules must be strong enough to perform motions while lifting the weight of chains of other modules, taking care of collisions, and maintaining the stability of the whole structure [2]. As a consequence, finding a good solution needs more effort.

Casal and Yim [1][13] present a divide-and-conquer strategy to plan reconfiguration for closed-chain robots. The configuration is first decomposed into a hierarchy of small sub-structures belonging to a finite set. Sets of substrates must be topologically non-homeomorphic, and reconfiguration between them must be simple. Reconfiguration between the sub-structures in the set are pre-computed and stored in a lookup table. The entire reconfiguration then consists of an ordered series of pre-computed actions happening locally among the sub-structures.

The authors present two algorithms for closed-chain reconfiguration: The first algorithm reconfigures the structure to an intermediate form (e.g. a single chain) and builds the final configuration from that intermediate structure. The second algorithm tries to match the initial and final configurations in a hierarchical manner, i.e. first matching the number of levels, then matching the number of sub-structures per level, then size of sub-structures, etc.

Yoshida et al. [14] presents a centralized planning-based approach to reconfigure a group of M-TRAN modules. The planner uses macro-actions with a block of modules instead of one. As a result the planning problem is simplified due to dealing with smaller number of sub-structures. The planner consists of an upper layer that plans the overall cluster motion called *flow* to realize locomotion along a given desired trajectory and the lower layer determines locally cooperative module motions, called *motion schemes*, based on a rule database.

Guided search is a natural choice to solve SRP problems. A heuristic function is used to guide the search toward the final configuration. This function usually approximates the number of actions needed to reach the final configuration.

Pamecha et al. [15] introduces some metrics to reflect the distance between two configurations. The most useful metric is the o*ptimal assignment metric*, which tries to optimally assign the modules of the initial configuration to the modules of the final configuration so that, the assignment cost function is minimized. Optimal assignment problem can be solved in $O(n^3 \times d)$ time using *Hungarian* method [16] where n is the number of modules and d is the cost of assigning a module to another.

Asadpour et al. [2] propose a graph theoretical approach to SRP. They use a similarity metric between configurations as a heuristic function, so that the configuration with more similarity to the final configuration is visited first. This enables them to find sub-optimal solutions in shorter time. They also use a graph isomorphism test to find out whether two configurations are isomorphic or not. This helps to cut some repetitive branches in search space and avoid solving a sub-problem multiple times.

III. PROPOSED METHOD

Our method is mainly based on the framework introduced by Asadpour et al. [2]. Each configuration in the *configuration space* is represented in a graph format, called *configuration graph*. In the configuration graph, modules are represented by vertices and connections between them are represented by edges (directed edge for male-female connections and undirected edge for genderless connections).

Starting from an initial configuration graph, a set of feasible *edit actions* (attach, detach) can be performed. These actions create a set of new configuration graphs. Among the unexplored configuration graphs, the configuration that is more likely to guide to a good solution (according to a heuristic function, discussed later) is selected to expand. In case of tie, a configuration will be selected randomly.

The concept of *graph signature,* which is an isomorphism invariant property of the configuration graph, is used to find repetitive configurations and avoid expanding them again. These repetitions are more frequent particularly when we deal with symmetric modules.

The *transition* from a configuration graph to another configuration graph via an edit action is saved in a *transition graph*, where configuration graphs are represented as nodes and edit actions are represented as edges.

A. Isomorphism Test

Graph isomorphism is one of the key problems in graph theory. It has not yet been proved whether it is NP-Complete or not [17], however it can be solved in polynomial time for special cases e.g. Graphs with bounded degrees [18] and ordered graphs [19]. In order to find isomorphic graphs, we try to extract an isomorphism invariant property of the graphs, called *graph signature*. The process of isomorphism test then consists of comparing the graph signatures.

Asadpour et al. [2] compute graph signature of configuration graphs using the properties of ordered graphs. The algorithm takes quadratic time in worst-case in term of the number of the vertices. However, dealing with symmetric modules and genderless connections makes it difficult to find isomorphic graphs. The reason is that the configuration graph cannot be trivially transformed to an ordered graph anymore. As a consequence, their method needs exponential time to compute graph signature. Here we try to tackle this problem by putting some order on the connections of the symmetric modules.

A *labeled module* is a module with unique labels on each of its connectors. Fig. 1 shows a labeled Molecube [6] with 6 connectors. Two *module labels* are *compatible* if it is possible to completely match the two labeled modules only by some servo movements or rotation of the whole module in 3D space. For example, there are 23 other ways to label the Molecube modules with compatible module labels. Hence we say that the *symmetry factor* of the module is 24.

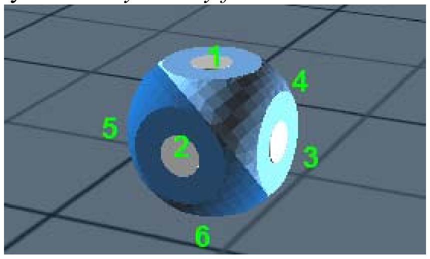

Fig. 1. A labeled Molecube

At the other hand, we can assign a label to each *connection* (*edge* in the configuration graph). A *connection label* consists of 3 values put together: the label of the connector of the source module, the label of the connector of the destination module, and the rotation code. The *rotation code* indicates the orientation of the destination module related to the source module e.g. two connected Molecubes have 4 possible relative orientations (0^o, 90^o, 180^o, 270^o).

Now assume we have a connected configuration graph G. Starting from a vertex v labeled as 1, we begin a Depth First Search (DFS) to visit the vertices. When we are in a vertex with multiple unvisited neighbors, we choose the next vertex according to the connector's labels. The neighbor that is connected to a connector with the lowest label is visited first. Every time we visit a new vertex (say k_{th} new vertex) we assign a new vertex label (k) to it. Then we label its connectors so that the label of its connection with its parent vertex is minimized. DFS continues till the entire vertices are visited. We will have the signature of v if the label of the vertices and the connections we visit in each step are written down. Fig. 2 and Fig. 3 show an example of this procedure.

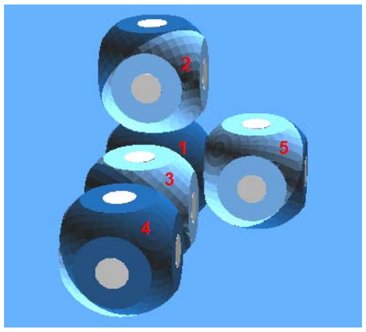

Fig. 2. A configuration of Molecube modules. The numbers are labels of the vertices.

If we start from the module marked 1 in Fig. 2, and use the module label shown in Fig. 1 for the initial module, the other modules will be visited in a sequence shown on Fig. 3. In result, the signature of vertex 1 with will be a sequence of vertex and edge labels as shown in Fig. 3.

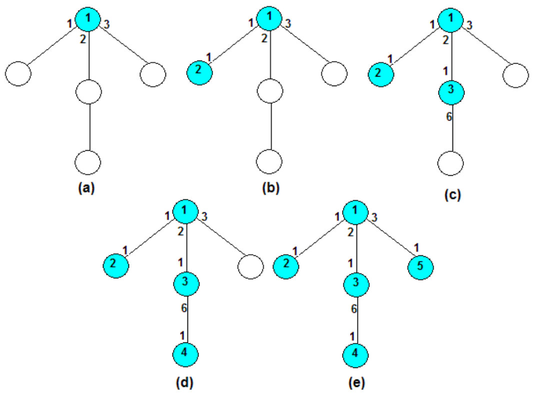

Fig. 3. Steps to compute signature starting from a vertex. The numbers written on the edges are connector labels. The signature is: 1[1 0° 1]2[2 0° 1]3[6 0° 1]4[3 0° 1]

Now if we compute all vertex signatures with all compatible labeling, and then choose the signature with minimum lexicographical order, we will have a unique signature of the configuration graph.

The time complexity of a DFS is $O(v + e)$ where v and e are the number of vertices and edges respectively. As far as the number of connectors of each module is finite, $v \sim e$, the cost is $O(v)$. Since we perform DFS for each initial vertex and each compatible module labeling, the total cost is $O(v^2 \times s)$ where s is the symmetry factor of the module. This method has lower worst-case complexity than the old method [2] that was $O(v^2 + v \times s^v)$ for symmetric modules.

B. Search Method

In our previous work [2] we used greedy heuristic search to find the solutions. Starting from the initial configuration, the next configuration is selected from the list of unvisited neighbors of the visited configuration. Among them, the configuration with maximum similarity to the final configuration according to a similarity metric is selected first. If the selected configuration is isomorphic to a previously visited configuration (i.e. they have the same signatures) it's not expanded anymore.

This approach tends to find solutions with no preference for the number of edit actions. However, we are interested in solutions with minimum number of actions (i.e. depth in the configuration graph). Therefore we try to provide a new heuristic function which puts some priority over the configurations with less depth in the configuration graph. A* search [21] sounds good for this purpose and guarantees optimality. But the extremely large search space of configurations does not allow finding a solution in appropriate time with this approach.

To tackle the problem, we introduce a new heuristic function. While this heuristic function dramatically improves the performance of the search, there is no guarantee that the first encountered solution is optimal. However, we believe this function is a good balance between time complexity and quality of solutions; it finds good solutions in appropriate time.

The new heuristic function consists of two parts. The first part is the old heuristic function [2] that was a metric based on the *graph edit distance*. Graph edit distance, δ, is the minimum number of graph edit actions (deletion or insertion of vertices or edges) to transform an initial graph to a final graph. It has been proved that it has the following relation with Maximum Common Sub-graph (MCS) of the graphs G and F [19]:

$$\delta(G,F) = 1 - \frac{|MCS(G,F)|}{\max(|G|,|F|)} \qquad (1)$$

But finding MCS is proved to be NP-Complete [17]. Raymond et al. [20] calculate an upper-bound for MCS. We have used a simplified version of their method to calculate an upper-bound for the edit distance of labeled graphs. We know two vertices can be matched if their edge labels match. So, if E^l_G and E^l_F are the number of edges of the input graphs that have label l, the upper-bound for the distance is:

$$\delta(G,F) = 1 - \frac{\sum_{l=0}^{l_{\max}} \min(E^l_G, E^l_F)}{\max(|G|,|F|)} \qquad (2)$$

It can be computed in linear time $O(max(|G|,|F|))$ using a hash table [2].

The second part of the new heuristic function is the number of edit actions required to reach the current configuration starting from the initial configuration, i.e. the length of the shortest path from the initial configuration to the current configuration(or a configuration isomorphic to the current configuration) in the transitions graph. This part can be calculated in constant time by adding a *depth* variable to the configuration graphs and saving the transition history. The depth variable is updated whenever a shorter path is found.

In summary, the heuristic function of a configuration G in terms of its distance d to the initial graph I (i.e. its depth) and its distance δ from the final configuration F is:

$$g(\, d(G,I),\ \delta(G,F)\,) \qquad (3)$$

where g is a suitable monotonically decreasing heuristic function of d and δ. The function that was empirically derived for our application is:

$$\left[\delta(G,F)\sqrt{d(G,I)}\right]^{-1} \qquad (4)$$

IV. RESULTS AND ANALYSIS

We have tested our method on a group of simulated M-TRAN modules. Each module has 6 genderless connectors and 2 rotational servos. The similarity factor of M-TRAN module is 4 since it has two symmetry lines. Therefore it is a good benchmark to test the proposed method.

Three reconfiguration tasks were considered: quadruped (Fig. 4) to snake (Fig. 5) studied by [2], quadruped to line (Fig. 6) and 8-module quadruped (Fig. 7) to 8-module snake (Fig. 8) studied by Kurokawa et al.[22]. Experiments in the first and the second tasks were repeated 30 times and the last one was repeated 15 times with different random seeds.

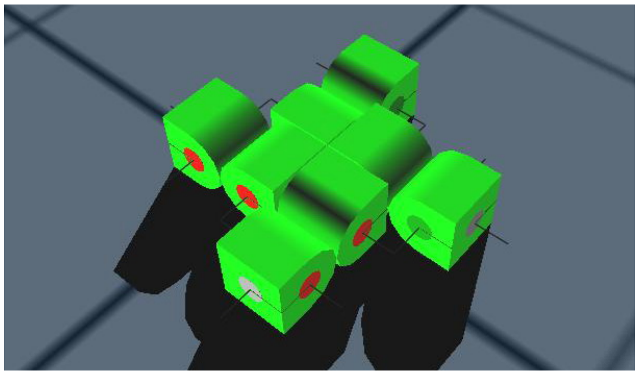
Fig. 4. Quadruped configuration

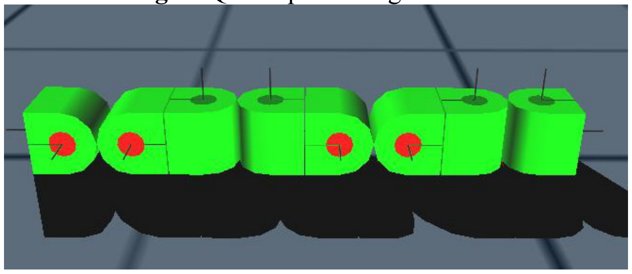
Fig. 5. Snake configuration

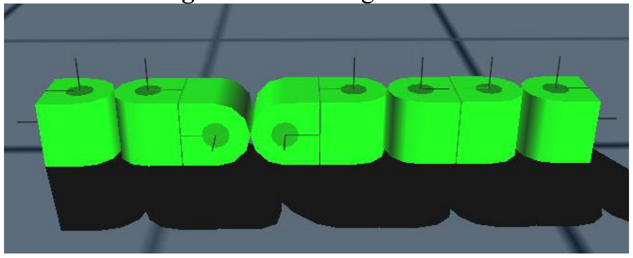
Fig. 6. Line configuration

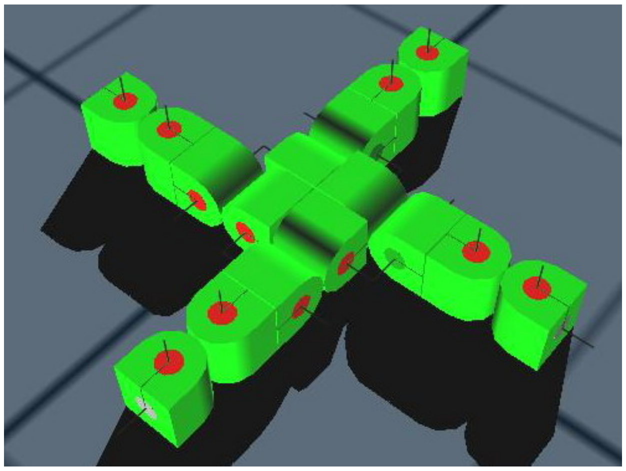
Fig. 7. 8-module quadruped

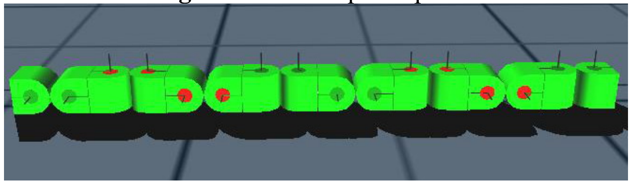
Fig. 8. 8-module snake

The results of the method are compared to the previous work in terms of quality of the found solutions (i.e. number of attach/detach actions) and computation cost (i.e. number of encountered graphs).

A. Quadruped to Snake:

The best solution we have found for this task consists of 9 attach/detach actions. This solution was always among the first 20 solutions encountered. This is a good result compared to [2] in which the best solution was only found in some cases. At the other hand, efficient isomorphism test allowed us to do a full BFS (exploring about 130,000 configurations of which about 44,000 are distinct) and find out that there is no better solution.

The first found solution had always less than 20 actions. This is also a much better result than [2] where the first found solution had always more than 20 actions.

The number of graphs visited before finding the first solution is depicted in Fig. 9. Compared to results of [2] (reshown in Fig. 10), we observe that the new heuristic is quite successful to guide the method toward the final configuration. Therefore the first found solution is always among the first 4,000 visited graphs, while the old method sometimes needed encountering more than 50,000 graphs.

The number of graphs visited to find the best solution among first 20 solutions (which was always the global optimum for our experiment) is presented in Fig. 11. Compared to results of [2], (reshown in Fig. 12) performance of the new heuristic function is one order of magnitude better than the old method.

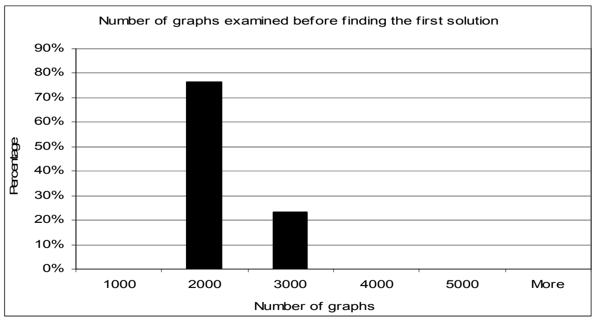

Fig. 9. Percentage of simulations vs. Number of graphs examined before finding the first solution for quadruped-to-snake task

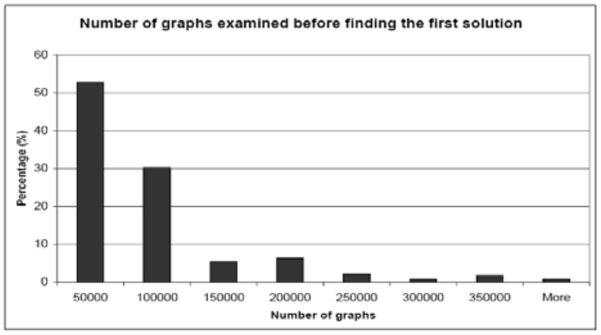

Fig. 10. Percentage of simulations vs. Number of graphs examined before finding the first solution among first 20 in [2] for quadruped-snake task

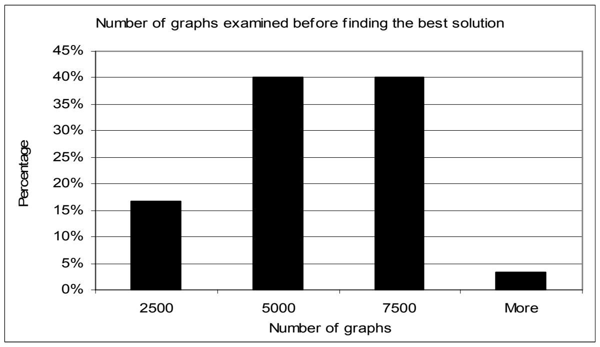

Fig. 11. Percentage of simulations vs. Number of graphs examined before finding the optimal solution for quadruped-to-snake task

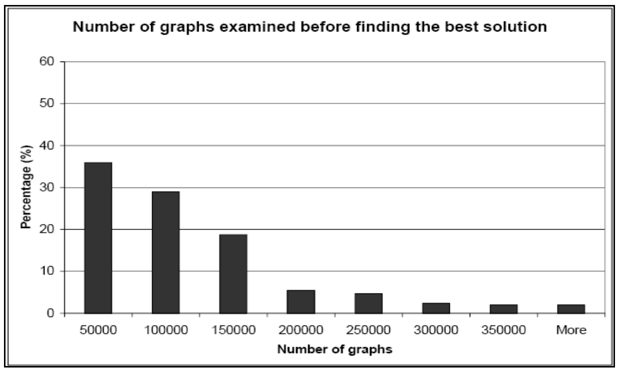

Fig. 12. Percentage of simulations vs. Number of graphs examined before finding the best solution among first 20 in [2] for quadruped-to-snake task

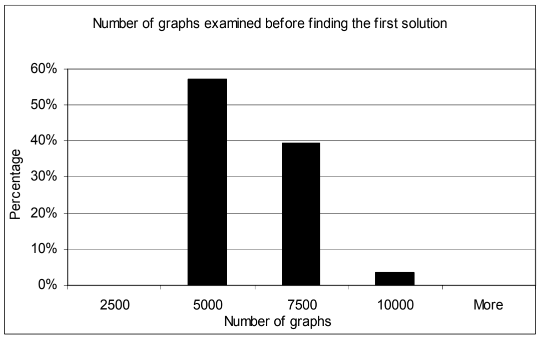

Fig. 13. Percentage of simulations vs. Number of graphs examined before finding the first solution for quadruped-to-line task

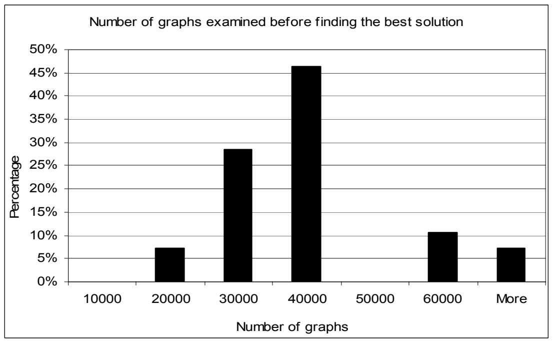

Fig. 14. Percentage of simulations vs. Number of graphs examined before finding the best solution for quadruped-to-line task

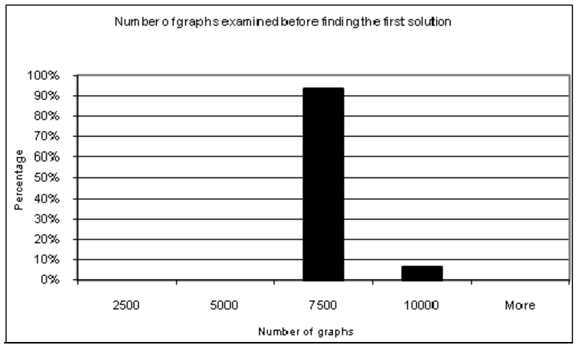

Fig. 15. Percentage of simulations vs. Number of graphs examined before finding the first solution for 8-module quadruped-to-snake task

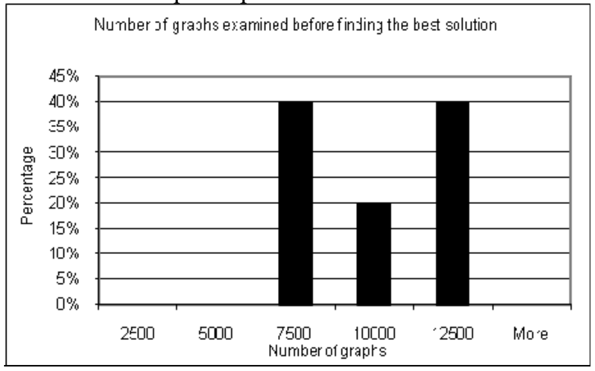

Fig. 16. Percentage of simulations vs. Number of graphs examined before finding the best solution for 8-module quadruped-to-snake task

B. Quadruped to line:

Our method found a solution with 9 attach/detach actions. The solution provided by Kurokawa et al. [22] consists of 14 attach/detach actions. The first found solution had always 11 actions. The number of graphs visited to find the first and the best solutions are depicted in Fig. 13 and Fig. 14. We see that this problem is somewhat harder than the previous experiment and more graphs should be visited to find good solutions.

C. 8-module quadruped to line:

Kurokawa et al. [22] provides a solution to this problem that includes 12 attach/detach actions (forgetting servo movements which are not counted here). However, this problem has been solved manually with the help of a planner. Our method found a solution with 7 attach/detach actions. The results of the experiments are depicted in Fig. 15 and Fig. 16. The first solution is usually found within visiting 7500 graphs and the best solution is encountered before visiting12500 graphs.

V.CONCLUSION

We tackled the problem of self-reconfiguration planning for modular robots by enhancing our previous method [2]. We presented, a graph isomorphism test based on the signature of labeled graphs and showed how it can be used to generate an isomorphism invariant code for modules with or without symmetry. This test was used to cut redundant paths from the initial configuration to the final one. We showed the graph isomorphism test runs in polynomial time (i.e. quadratic worst case time) even in case of symmetric genderless modules.

A heuristic function was used in a guided search to find feasible solutions based on the graph edit-distance between the current configuration and the final one and the length of the shortest path from the initial configuration to the current configuration. The simulation results showed this heuristic leads the search algorithm to find better solutions by examining fewer graphs.

For future we would like to investigate the optimal way of combining the two parameters we used in the new heuristic function. Also, optimizing the planning based on other criteria e.g. the time required to reconfigure instead of the number of attach/detach actions would be another interesting issue.

REFERENCES

[1] A. Casal and M. H. Yim, “Self-reconfiguration planning for a class of modular robots,” in Proc. SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II, G. T. McKee and P. S. Schenker, Eds., vol. 3839, Aug. 1999, pp. 246–257.

[2] M. Asadpour, A. Sproewitz, A. Billard, P. Dillenbourg, A. Ijspeert. Graph Signature for Self-Reconfiguration Planning, Accepted for publication, IROS 2008.

[3] D. Rus and M. Vona, “A physical implementation of the self-reconfiguring crystalline robot.” in Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2000, pp. 1726–1733.

[4] S. Vassilvitskii, J. Kubica, E. Rieffel, J. Suh, and M. Yim, “On the general reconfiguration problem for expanding cube style modular robots,” in Proceedings of the 2002 IEEE Int. Conference on Robotics and Automation, 11-15 May 2002, pp. 801–808.

[5] E. H. Ostergaard and H. H. Lund, “Evolving control for modular robotic units,” in Proceedings of CIRA’03, IEEE International Symposium on Computational Intelligence in Robotics and Automation, Kobe, Japan, 16-20 July 2003, pp. 886–892.

[6] K. Kotay, D. Rus, M. Vona, C. McGray, The self-reconfiguring Molecule: design and control algorithms. Proc. of the Algorithmic Foundations of Robotics, Houston, USA.

[7] T. Fukuda and S. Nakagawa, Dynamically Reconfigurable Robotic Systems. Proc. of IEEE Intl. Conf. on Robotics and Automation, 1988.

[8] H. Kurokawa, K. Tomita, A. Kamimura, S. Murata, Y. Terada, and S. Kokaji, “Distributed metamorphosis control of a modular robotic system M-TRAN,” in Distributed Autonomous Robotic Systems (DARS) 7, Springer, pp. 115–124, 2006.

[9] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. Ijspeert, “Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface,” Industrial Robot, vol. 33, no. 4, pp. 285–290, 2006.

[10] W.-M. Shen, P. Will, A. Galstyan, and C.-M. Chuong, “Hormone-inspired self-organization and distributed control of robotic swarms,” Autonomous Robots, vol. 17, no. 1, pp. 93–105, 2004.

[11] D. Duff, M. Yim, and K. Roufas, “Evolution of polybot: A modular reconfigurable robot,” in Proc. of the Harmonic Drive Intl. Symposium and Proc. of COE/Super-Mechano-Systems Workshop, Japan, Nov 2001.

[12] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson, “Self-reproducing machines”, Nature, 435, No. 7038, pp. 163-164, 2005.

[13] M. H. Yim, D. Goldberg, and A. Casal, “Connectivity planning for closed-chain reconfiguration,” in Proc SPIE, Sensor Fusion and Decentralized Control in Robotic Systems III, G. T. McKee and P. S. Schenker, Eds., vol. 4196, Oct. 2000, pp. 402–412.

[14] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, “A self-reconfigurable modular robot : Reconfiguration planning and experiments,” The International Journal of Robotics Research, vol. 21, no. 10, pp. 903–916, 2002.

[15] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian, “Useful metrics for modular robot motion planning,” IEEE Trans. on Robotics and Automation, vol. 13, no. 4, pp. 531–545, 1997.

[16] H. W. Kuhn, “The hungarian methods for the assignment problem,” Naval Research Logistic Quarterly, vol. 2, pp. 83–97, 1955.

[17] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: W. H. Freeman, 1979.

[18] E. M. Luks, “Isomorphism of graphs of bounded valence can be tested in polynomial time,” Journal of Computer and System Sciences, vol. 25, p. 4265, 1982.

[19] X. Jiang and H. Bunke, “Optimal quadratic-time isomorphism of ordered graphs.” Pattern Recognition, vol. 32, no. 7, pp. 1273–1283, 1999.

[20] J. W. Raymond, E. J. Gardiner, and P. Willett, “RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs,” The Computer Journal, vol. 45, no. 6, pp. 631–644, 2002.

[21] S. Russel and P. Norvig, “Artificial Intelligence: A Modern Approach”, Prentice Hall, Englewood Cliffs, NJ, p. 96, 1995.

[22] http://unit.aist.go.jp/is/dsysd/mtran3/FlashMovie/mtran3/movie.htm