A New Graph Signature Calculation Method Based on Power Centrality for Modular Robots

Keyvan Golestan, Masoud Asadpour, and Hadi Moradi

Abstract. Graph signature is a fast isomorphism test that is used in self-reconfiguration planning of modular robots. In case of dealing with homomorphic modules, the required time to calculate the signature grows exponentially with the number of symmetry lines. We tackle this problem by introducing an isomorphism-invariant signature calculation method, which is based on the power centrality of nodes. We also introduce a new sample-based search method. Simulation results show the new method finds better solutions in a significantly shorter time.

1 Introduction

Modular robots are composed of some relatively simplified and usually small-sized robotic parts called *modules*. The modularity comes with properties such as versatility, robustness and low cost. The modules can be connected in different ways (either manually or automatically) thus creating different configurations.

Based on the connection structure and the movement of modules, they are classified into two main categories. *Lattice-type* modules use cluster-flow to move and reconfigure. Crystalline [1], ATRON [2], Telecube [3] and Molecule [4] are examples of this kind. *Chain-type* modules form chain structures and have joints that help them move without necessarily doing reconfiguration. M-TRAN [5], CONRO [6], Roombot [7], PolyBot [8], YaMoR [9] and SuperBot [10] are examples of this type. Our work is based on chain-type modular robots.

Self-Reconfiguration Planning (SRP) is a task in which an optimal or sub-optimal solution is found for reshaping a modular robot from an initial configuration to a final one. Solution to this task is hard to achieve as the time complexity of planning problem grows exponentially when the number of modules or their degrees of freedom (DOF) increases. In this paper we propose a general framework for SRP to reduce the time complexity specially when dealing with modules

Keyvan Golestan ; Masoud Asadpour ; Hadi Moradi
Faculty of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
e-mail: {kgolestan,asadpour,moradih}@ut.ac.ir

A. Martinoli et al. (Eds.): Distributed Autonomous Robotic Systems, STAR 83, pp. 505–516.
springerlink.com

with high DOF. This method uses the idea of graph signature introduced by Asadpour et al. [11][12] and improves it by using the concept of power centrality in social networks [13] and establishing a hierarchical graph signature calculation algorithm. A sample-based method for investigating feasible "attach" actions, has also been incorporated with our previous general search to make it faster.

The rest of this paper is organized as follows: In the next section, some previous works on SRP are explained. The third section explains our proposed method. The paper is then finalized by simulation results and conclusions.

2 Background

SRP is one of the most challenging tasks in modular robots. The task is even more difficult in chain-type robots where mechanical limitations come into play. SRP becomes practically intractable when dealing with large number of modules, or modules with high DOF.

Reconfiguration planners are usually based on a guided search strategy and a distance function. Casal and Yim [14], [15] introduced a divide-and-conquer strategy to solve SRP for chain-type robots. The configuration was decomposed into a hierarchy of small sub-structures belonging to a finite set. These sub-structures were non-homomorphic and reconfiguration between them was simple. So the reconfiguration steps could be specified and stored in a look-up table in advance. The reconfiguration was then consisted of an ordered set of pre-defined actions among sub-structures which happen locally.

Hou & Shen [16] presented a distributed reconfiguration method on the unlabeled graph representation of configurations. They did the reconfiguration by first utilizing a distributed comparison to detect substructures in two configurations. Then the reconfiguration was limited only to the modules that indicated difference in topology. Reconfiguration took place by first converting the initial configuration to an intermediate structure and then transforming it to final configuration.

Asadpour et al. [11] proposed a method based on graph theory. They encoded the 3D structure of a configuration by an isomorphism-invariant code, called *signature,* and used *edit distance* based similarity metric as an upper-bound for isomorphism test. The time complexity of their algorithm grows exponentially as the number of modules increases. They improved the method in [12] in order to deal with modules with symmetry. Again the time complexity grows exponentially as the number of modules increased. Another disadvantage of their method is where they are looking for feasible attach actions; they had to search all possible permutations of joint angles of each module in a configuration. Here, an increase in the DOF of the modules would cause the time complexity of finding feasible attach actions grow exponentially.

3 Our Proposed Method

We represent a configuration by a graph with modules as *nodes* and connection between modules as *edges* (directed for male/female connectors and undirected for genderless ones). Reconfiguration takes place by performing feasible edit actions,

i.e. attach or detach, on initial configuration hoping to make it closer to the final configuration. Among the new unexplored configurations, the *closest* configuration (based on a distance function) to the final one is chosen for further exploration. This process continues until a path of *transitions* from the initial graph to the final graph is found. The network of transitions between configurations in which, nodes are configurations and edges are edit actions, is called a *transition graph*.

The isomorphism checking of graphs is not yet proved to be NP-Complete or not [17] except some special cases like graphs with bounded degrees [18] or ordered graphs [19]. Asadpour et al. [11] compute a unique identifier, called *graph signature*, for a configuration using properties of ordered graphs. An ordered graph is a graph whose edges have a specific order. So they adopted an edge labeling method to assign a unique identifier to each edge. The labeled graph is trivially transformable to an ordered graph by sorting the out-edges of the vertices in lexicographic order. It is shown in [19] that Isomorphism test on such a graph takes quadratic time in worst-case in terms of the number of nodes.

3.1 Labeling the Graph

Edge labels include information about how modules are connected to each other. A labeling strategy may be as follows: First, the connectors of a module are indexed. The indexing order is arbitrary, but should be the same for all modules. Fig.1.A and B show an indexing order for SuperBot modules.

Relative rotation of two modules around their connection point should also be encoded. This is done by assigning an index to each relative rotation. For instance, if only multiples of 90 is allowed, the 90° angle between the modules in Fig.1-C would have index 1 (index 0 for 0°, index 1 for 90°, … , index 3 for 270°).

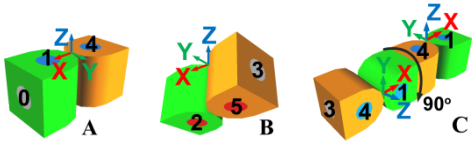

Fig. 1 Connector indexing of a SuberBot module. (A) Top view (B) Bottom view (C) Relative rotation at the connection point is 90°, so rotation index is 1.

Putting connector and relative rotation indices together, connection of two modules can be labeled as [11]:

$$l_{ij} = |C| \, |R| \, c_{ij} + |R| \, c_{ji} + r_{ij} \tag{1}$$

where c_{ij} is the index of the connector of module i which is connected to module j, $|C|$ is the total number of connectors, $|R|$ is the number of possible relative rotations, and r_{ij} is the relative rotation of module i with respect to module j. For SuperBot, $|C|$ is 6 and $|R|$ is 4. This way, each connection gets a unique label and an ordering can be imposed on the edges. Fig. 2 shows two different configurations and their graph representations.

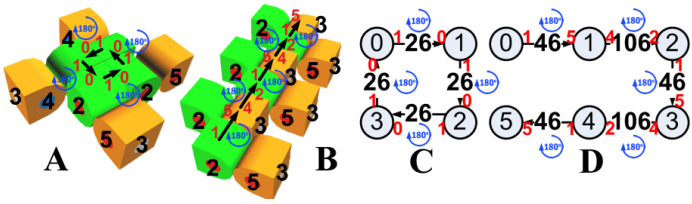

Fig. 2 SuperBots configured as (A) 4-Module quadruped (B) 6-Module climber. (C) and (D) Their corresponding labeled graphs. Red indices belong to the used connectors.

3.2 Graph Signature

The term *Graph Signature* proposed by Asadpour et al. [11] is an isomorphism invariant property of the configuration graph that encodes the 3D structure of a modular robot. It is inspired by the way we help somebody who is searching for an address (e.g. go straight, turn left at the junction). A configuration is like a city whose entire map should be encoded in a string. The code specifies for a tourist (here a planner), what would he/she see when walking along the streets? A big problem is to specify (in an isomorphism invariant way) the place he should start walking. That is why Asadpour et al. [11] had to try all possible start places (i.e. all nodes) and select in some way a unique code among them (e.g. by sorting).

The graph signature is created by performing a modified DFS on an ordered graph and recording what is seen meanwhile, in this way: 1) start from a node (i.e. a module) and record its index; start indexing from one, and increment it upon visiting new unvisited node; 2) If possible traverse the out-edges in the lexicographic order of their labels and record their labels, after that 3) traverse the in-edges (i.e. opposed to their direction) and record the negation of their labels; and finally 4) if no move is possible, back track to the previous node(s). In case of dealing with undirected (i.e. hermaphrodite) edges, the traversal direction is decided once it is first encountered, from the current node towards an unvisited node.

The procedure is repeated for all nodes as start position. Each time a signature is created whom we call a *node signature*. Among all node signatures, the one with maximum lexicographical order is selected as the graph signature. The worst

Table 1 Signature of nodes of the graphs in Fig.2. Graph signature is shown in bold face.

4-Module Quadruped Configuration		**6-Module Climber Configuration**	
Node	Node Signature	Node	Node Signature
0	**(0 26 1) (1 26 2) (2 26 3) (3 26 0)**	0	(0 46 1)(1 106 2)(2 46 3)(3 106 4)(4 46 5)
1	**(0 26 1) (1 26 2) (2 26 3) (3 26 0)**	**1**	**(0 106 1)(1 46 2)(2 106 3)(3 46 4)(0 -46 5)**
2	**(0 26 1) (1 26 2) (2 26 3) (3 26 0)**	2	(0 46 1)(1 106 2)(2 46 3)(0 -106 4)(4 -46 5)
3	**(0 26 1) (1 26 2) (2 26 3) (3 26 0)**	3	(0 106 1)(1 46 2)(0 -46 3)(3 -106 4)(4 -46 5)
		4	(0 46 1)(1 -106 2)(2 -46 3)(3 -106 4)(4 -46 5)
		5	(0 -46 1)(1 -106 2)(2 -46 3)(3 -106 4)(4 -46 5)

time complexity of signature calculation is $O(|V||E|)$ for $|V|$ and $|E|$ as the number of nodes and edges, respectively [11]. Table 1 shows all possible graph signatures for the configurations of Fig.2.

3.2.1 Symmetric Modules

A module is called *homomorphic* or *symmetric* if there exist at least one symmetry line that rotating the module around it produces the same (matchable) 3D shape. Fig. 3 shows some SuperBots that have the same 3D shapes. The number of matchable shapes, whom we call, *symmetry factor*, can be calculated once for each module type through exhaustive search. For instance, M-TRAN modules, if all connectors are genderless, have 3 symmetry lines, and their symmetry factor is 4. The symmetry factor is 8 for SuperBots [10] and 36 for Roombots [7]. This is like *re-indexing* the connectors and acquiring the same shape. Thus, a mapping between connector indices of symmetric shapes could be achieved and saved in a lookup table. Since re-indexing the connectors changes the edge labels, a new graph signature might be achieved.

Fig. 3 (A)-(F) Examples of SuperBots with identical 3D shapes. In B, E and, F, the middle servo rotates 180° (G) Two isomorphic configurations with a module rotated around a symmetry line.

The case of symmetric modules is tackled in [12] by putting some order on the connections. They compute the node signatures by testing all permutations of symmetric positions of each module and choose the one with maximum lexicographical order. The time complexity of signature calculation is $O(|V|^2 + |V| \times S^{|V|})$ for S being the symmetry factor.

3.2.2 The Improved Signature Calculation Algorithm

The inefficiency of signature calculation in symmetric modules returns back to the calculation of multiple node signatures. If we find an isomorphism invariant way to fix the starting node, we could calculate the signature in one run. Here we use a centrality measure from social networks domain called, *power* centrality [13] based on which the most powerful node is selected as the starting node for signature calculation. This measure can also specify the priority of visit for nodes in case of tallies. Following, the steps of the proposed method are explained:

Step 1: Isomorphism-invariant node prioritization based on power centrality

Based on the power centrality [13], (social) power of a node recursively depends on the sum of the power of its friends with attenuation factor $0 \leq \beta \leq 1$:

$$C_p(n_i) = \sum_j a_{ij}(\beta . C_p(n_j)) \tag{2}$$

where $C_p(n_k)$ is the power of node k and a_{xy} is 1 if x and y are friends (or neighbors) and 0 otherwise. If we start from a positive initial value for $C_p(n_k)$, $k=1…n$ and recursively calculate (2) for all nodes, power centralities finally converge if $|\beta|<1/\lambda_{max}$ where λ_{max} is the highest eigenvector of the adjacency matrix of the graph[13]. The number of required iterations can go up to the diameter of the graph that is at most $|V|-1$.

It is evident that for isomorphic configurations the same power centralities are gained for corresponding nodes. In case of generating different power centralities for some nodes, we can surely say the configurations are different. However, the reverse is not always true i.e. if power centralities are the same we cannot surely say the configurations are isomorphic.

We hope the most powerful node is unique such that it could be selected as the starting node, otherwise we have to run the signature calculation once for each of the most powerful nodes. Table 2 shows the converged power centralities of graphs presented in Fig. 2.

Table 2 Normalized initial and converged node power centralities of the graphs in Fig. 2C-D

4-Module quadruped configuration			**6-Module climber configuration**		
Node	Initial power	Converged power	Node	Initial power	Converged power
0	**0.25**	**0.25**	0	0.1073	0.1038
1	**0.25**	**0.25**	1	0.1964	0.1804
2	**0.25**	**0.25**	**2**	**0.1964**	**0.2157**
3	**0.25**	**0.25**	**3**	**0.1964**	**0.2157**
			4	0.1964	0.1804
			5	0.1073	0.1038

Choosing the initial values for power centralities is very important. A popular initial value is the nodal degree [13], which is not appropriate here because the degree keeps only the information about the number of neighbors, but not how they are connected. Instead, we assign a value to each node that is calculated by summing the labels of the edges connected to a specific node. We call it the *Vicinity Value*. The *maximum* possible vicinity values of nodes are used as their initial powers. If we assume the modules are genderless, the vicinity value of node i is:

$$v(i) = \sum_{j \in V} a_{ij} l_{ij} \tag{3}$$

where l_{ij} is the label of the edge between nodes i and j. Based on (1) and (3) the maximum vicinity value is:

$$v^*(i) = \sum_{j \in V} a_{ij} l_{ij} = \sum_{j \in V} a_{ij} (|C| \, |R| \, c_{ij}^* + |R| \, c_{ji}^* + r_{ij}^*) \tag{1}$$

where c_{ij}^* and c_{ji}^* are the connector indices that maximizes the vicinity value, and are obtained by re-indexing the modules i and then j; and r_{ij}^* is the maximum rotation index between i and j. Equation (4) can be written as:

$$v^*(i) = \sum_{j \in V} a_{ij} (|C| \, |R| \, c_{ij}^* + r_{ij}^*) + |R| \sum_{j \in V} a_{ij} (c_{ji}^*) \tag{2}$$

Maximizing both terms would maximize the whole expression. The first term, which we call *Node Value,* only depends on the module i and can be maximized using the equation below:

$$n^*(i) = \max_{s \in S} \{ \sum_{j \in V} a_{ij} (|C| \, |R| \, s(c_{ij}) + s(r_{ij})) \} \tag{3}$$

where S is the set of possible connector mappings (|S| being the symmetry factor), and $s(c_{ij})$ and $s(r_{ij})$ are the connector and rotation indices, respectively, provided by the mapping s. We call this process, *Node Value Maximization Procedure (NVMP)*. The neighboring nodes can independently run NVMP and maximize the second term of (5) and finally the sum of two terms. The whole process is called *Vicinity Value Maximization Procedure (VVMP)*. Fig.4 shows an example of how NVMP and VVMP are performed using equations 3, 5 and 6. If two nodes are connected in more than one way, VVMP selects the mapping that leads to maximum vicinity value.

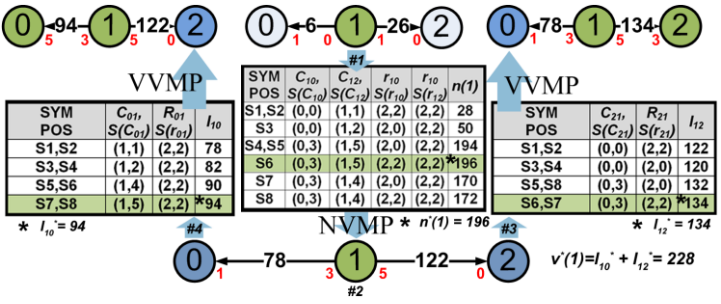

Fig. 4 Finding the maximum vicinity value of node 1 for 4-Module quadruped graph shown in Fig. 2C. (#1 & #2) NVMP is done by node 1 using the Eq. 6 for each symmetric position S1-S8. (#2 & #3) VVMP is done independently by each neighbor using Eq. 6 for each symmetric position S1-S8 concerning the rule that common edges with node 1 can only get increased.

Step 2: Hierarchical graph re-indexing

This procedure is done once for the node with maximum power centrality, which we call the *master node*. The master node performs VVMP followed by re-indexing according to the mapping provided by VVMP. Then, the edges incident to the master node with their new labels are frozen so that no other node could modify their labels. This process is repeated for a neighbor of the master node and each time some edges are frozen. Neighbors are prioritized according to their vicinity value. When all edges are frozen, the re-indexed graph is ready for signature calculation. Fig. 5 shows an example of re-indexing.

It should also be noted that during NVMP some cases happen where for two different symmetric positions, calculated node values of a specific module are equal and it cannot decide which symmetric position *s* to choose for re-indexing. In such cases we simply look at the power centralities of the nodes that the connector tends to connect, and choose the symmetric position in which the connectors with higher indices are going to connect to nodes with higher power centralities. We think in cases of equal power centralities choosing either one of the nodes would not change the overall outcome (not proved yet, left for future work).

Fig. 5 Hierarchical re-indexing of 6-Module climber configuration graph. Module 3, the master starts VVMP and freezes its edges, the process is continued by 2, 4, 1, 5, and 0.

Step 3: Graph signature calculation

The graph signature is generated by performing a DFS starting from the master node on the frozen graph of step two. If more than one master node (and consequently more than one re-indexed graph exists), graph signature is generated for each case by starting from the corresponding nodes and the one with maximum lexicographical order is chosen. It is easy to verify that the worst time complexity of this new method is $O(S \times |V|^2)$ which is much better than [12].

3.2.3 Improvement of Searching for Feasible Attach Actions

As mentioned earlier, reconfiguration takes place by performing feasible edit actions, i.e. attach/detach, on initial configuration hoping to make it closer to final one. Finding feasible detach actions is easy; it can be done by searching for modules that form loop (so they can be disconnected from either side). This can at worst be done in $O(|E|^2|V|)$. However, computing all possible attach actions, especially in case of modules with high DOF, is very difficult. Asadpour et al. [11] [12] used a brute force approach and checked all permutations of discretized servo movements for possible attach action i.e. $O(p^{|V|\times|M|})$, for p being the number of discretized servo movements and $|M|$ being the DOF of modules (so the dimension of the *joint state space* is $|V|\times|M|$).

We tackle this problem by applying a sample based method called Rapidly-exploring Random Trees (RRT) [20]. The main property of RRT is its tendency to search unexplored regions of the space, while insuring that the whole space will be explored if sampling runs for a long time. Here, instead of finding all feasible attach actions which takes a lot of time, the sampling is continued until either "enough" attach actions are found or no part of space remains unexplored.

4 Simulation Results

Our method is tested on simulated M-TRANs and SuperBots, each having 6 genderless connectors, and 2 and 3 rotational servos respectively. Fig.2A, B and Fig.6 depict the configurations we study. The SRP tasks for MTRAN are Quadruped-to-snake with 4 and 8 modules. Then the scalability of our method is verified on stool-to-snake reconfiguration (with 12 M-TRAN). The SRP tasks on SuperBots are line-to-climber configuration with 4, 6, and 8 modules. Simulations are repeated on 30 random seeds and are continued until 30 solutions are found.

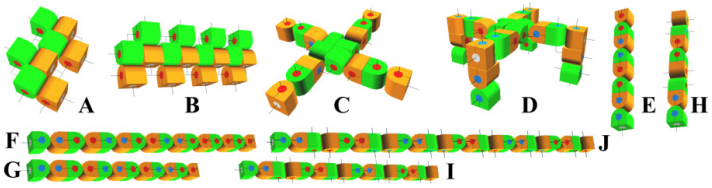

Fig. 6 (A, B) 4 and 8-module climber (C) 8-module quadruped (D) 12-module stool (E, F, G) 4, 6 and 8-module line (H, I, J) 4, 8 and 12-module snake

4.1 Reconfiguration with M-TRAN

Fig.7 (left) shows the number of graphs examined before finding the first and the best solutions of 4-module quadruped-to-snake reconfiguration. In about 70% of simulations the first solution is found before less than 500 graphs are examined, in less than 5 seconds. Moreover the first solution is always the best solution with 9 attach/detach actions. This is much better than Asadpour et al. [12] where the first solution was among 4,000 visited graphs and only about 17% of best solutions were within the first 2,500 examined graphs.

Fig.7 (right) shows the results for 8-module quadruped-to-snake task. The best solution of this task has 7 actions which is equal to results of [12]. The first solution is found by visiting 7000 or fewer graphs in 50% of simulations in less than 300 seconds. The best solution is found in 7 simulations after visiting 15000 or fewer graphs in less than 450 seconds. This is almost similar to [12], but is gained in significantly shorter time.

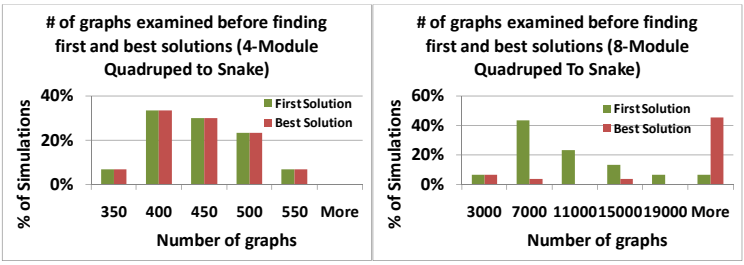

Fig. 7 Reconfiguration result for (Left) 4-Module quadruped to snake and (Right) 8-Module quadruped to snake

To test the scalability of our framework, we solved the stool-to-snake reconfiguration with 12 M-TRAN modules. The best solution that has 27 actions is found by examining around 120000 graphs in about 85 minutes.

4.2 Reconfiguration with SuperBot

Fig.8 shows the number of graphs examined before finding the first and the best solutions of line-to-climber reconfiguration with SuperBots. Fig.8 (top-left) shows the result for 4-module reconfiguration. It is seen that more than 70% of the first solutions and about 20% of the best solutions are found after visiting 2800 or fewer graphs. The first and best solutions are found in less than 10 seconds.

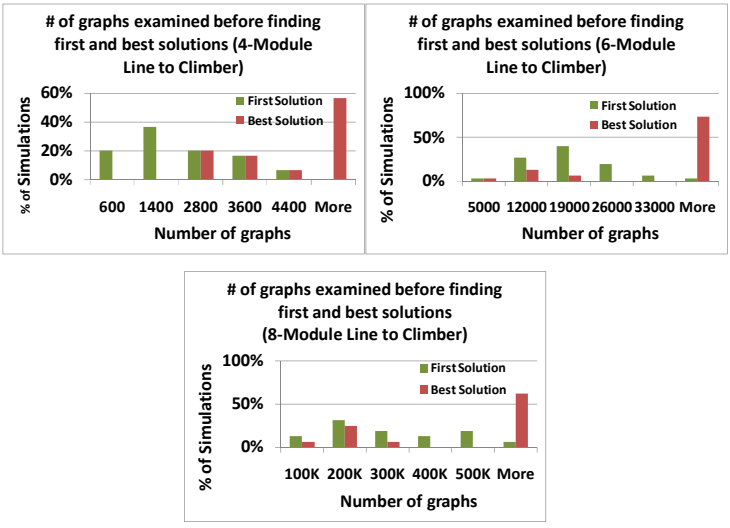

Fig. 8 Result for line-to-climber reconfiguration with (top-left) 4, (top-right) 6 and (bottom) 8 modules

Fig.8 (top-right) shows the results for 6-module reconfiguration. It is seen that more than 50% of the first solutions are found by examining 19000 or fewer graphs in less than 30 seconds. The best solution to this task has 12 actions and is found in about 23% of our simulations.

Fig.8 (bottom) shows the results for 8-module reconfiguration. This is the hardest reconfiguration task in our simulations (the configuration has totally 24 DOFs). About 63% of the times, the first solution is found after examining 300000 or fewer graphs in less than 120 minutes. The best solution that has 24 actions is found only in 34% of simulations always before examining 300000 or fewer graphs in less than 200 minutes.

5 Conclusion

We proposed an isomorphism-invariant graph signature calculation based on power centrality. We could enhance the time complexity of signature calculation to polynomial time even for symmetric modules. We also tackled the problem of finding feasible attach actions by using the sample based RRT method. The results showed an impressive drop in reconfiguration time for both M-TRAN and Super-Bot modules.

As future works, we think finding the feasible attach action by sampling can be improved through parameter tuning. Physical restrictions during reconfiguration should be mentioned and finally the cases where power centralities or vicinity values of some nodes are equal need more investigation.

References

[1] Rus, D., Vona, M.: Crystalline robots: Self-reconfiguration with compressible unit modules. Autonomous Robots 10, 107–124 (2001)

[2] Jørgensen, M., Østergaard, E., Lund, H.: Modular ATRON: Modules for a self-reconfigurable robot. In: 2004 IEEE/RSJ Int. Conf. on Intelligent. Robots & Systems (IROS), pp. 2068–2073 (2004)

[3] Vassilvitskii, S., Kubica, J., Rieffel, E., Suh, J., Yim, M.: On the general reconfiguration problem for expanding cube style modular robots. In: Proceedings - IEEE International Conference on Robotics and Automation, pp. 801–808 (2002)

[4] Kotay, K., Rus, D., Vona, M., McGray, C.: Self-reconfiguring robotic molecule. In: Proceedings - IEEE International Conference on Robotics and Automation, pp. 424–431 (1998)

[5] Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K., Kokaji, S.: M-TRAN: Self-reconfigurable modular robotic system. IEEE/ASME Trans. on Mechatronics 7, 431–441 (2002)

[6] Castano, A., Shen, W., Will, P.: CONRO: towards deployable robots with inter-robot metamorphic capabilities. Autonomous Robots 8, 309–324 (2000)

[7] Sproewitz, E., Billard, A., Dillenbourg, P., Ijspeert, A.J.: Roombots—Mechanical Design of Self-Reconfiguring Modular Robots for Adaptive Furniture. In: IEEE International Conference on Robotics and Automation, pp. 4259–4264 (2009)

[8] Yim, M., Duff, D.G., Roufas, K.D.: PolyBot: a modular reconfigurable robot. In: Proceedings - IEEE International Conference on Robotics and Automation, pp. 514–520 (2000)

[9] Moeckel, R., Jaquier, C., Drapel, K., Dittrich, E., Upegui, A., Ijspeert, A.: Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface. Industrial Robot 33, 285–290 (2006)

[10] Salemi, B., Moll, M., Shen, W.: Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In: IEEE International Conference on Intelligent Robots and Systems, pp. 3636–3641 (2006)

[11] Asadpour, M., Sproewitz, A., Billard, A., Dillenbourg, P., Ijspeert, A.: Graph signature for self-reconfiguration planning. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 863–869 (2008)

[12] Asadpour, M., Ashtiani, M., Sproewitz, A., Ijspeert, A.: Graph signature for self-reconfiguration planning of modules with symmetry. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 5295–5300 (2009)

[13] Bonacich, P.: Power and Centrality: A Family of Measures. The American Journal of Sociology 92, 1170–1182 (1987)

[14] Yim, M., Goldberg, D., Casal, A.: Connectivity planning for closed-chain reconfiguration. In: Proceedings of SPIE - The Int. Society for Optical Engineering, pp. 402–412 (2000)

[15] Casal, A., Yim, M.: Self-reconfiguration planning for a class of modular robots. In: Proceedings of SPIE - The International Society for Optical Engineering, pp. 246–257 (1999)

[16] Hou, F., Shen, W.: Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots. In: Proceedings - IEEE International Conference on Robotics and Automation, pp. 3135–3140 (2008)

[17] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W.H. Freeman (1979)

[18] Luks, E.: Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer and System Sciences 25, 42–65 (1982)

[19] Jiang, X., Bunke, H.: Optimal quadratic-time isomorphism of ordered graphs. Pattern Recognition 32, 1273–1283 (1999)

[20] Lavalle, S.M.: Rapidly-Exploring Random Trees: A New Tool for Path Planning (1998)