Graph Signature for Self-Reconfiguration Planning

Masoud Asadpour, Alexander Sproewitz, Aude Billard, Pierre Dillenbourg, and Auke Jan Ijspeert

Abstract— **This project incorporates modular robots as building blocks for furniture that moves and self-reconfigures. The reconfiguration is done using dynamic connection / disconnection of modules and rotations of the degrees of freedom. This paper introduces a new approach to self-reconfiguration planning for modular robots based on the graph signature and the graph edit-distance. The method has been tested in simulation on two type of modules: YaMoR and M-TRAN. The simulation results shows interesting features of the approach, namely rapidly finding a near-optimal solution.**

Keywords: Modular self-reconfigurable robots, adaptive furniture, graph isomorphism, graph signature, graph edit distance.

I. Introduction

Future working and living environments will be composed of places where people and new technologies cohabit seamlessly. Thanks to the recent progress in tangible interaction with computers [1], ubiquitous computing [2], and augmented reality [3], a movement is observed towards integrating technologies in everyday artifacts, ranging from tables to walls and even carpets or kitchen furniture. This new field is referred to as “roomware” [4] or interactive furniture. It addresses the design and the evaluation of computer-augmented room elements like doors, walls, furniture with integrated information and communication technology.

Although roomware projects deal with user interaction, users have few possibilities to contribute to the design. We aim at developing roomwares able to adapt their morphology to the users needs. We therefore envision scenarios where parts of the furniture are capable of locomotion, self-assembly, self-reconfiguration, and self-repair, depending on user’s preferences. We believe this is where modular robotics can contribute.

Modular robots are robots made of multiple simple robotic modules that can attach and detach. Connectors between units allow creation of arbitrary and changing structures depending on the task to be solved. Using such simple modules as building blocks, constructing a variety of furniture is possible. We call these robotic modules Roombots for roomware-robots. Fig.1 shows some possible furniture.

These pieces of furniture should not fill a big part of the available space, therefore they must be reusable as much

M. Asadpour is with Control and Intelligent Processing Center of Excellence, ECE Dept. University of Tehran, Iran. masoud.asadpour@ieee.org

A. Sproewitz and A.J. Ijspeert are with Bio-Inspired Robotics Group (BIRG), A. Billard is with Learning Algorithms and Systems Laboratory (LASA), P. Dillenbourg is with Centre de Recherche et d’Appui pour la Formation (CRAFT), all in Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland {alexander.sproewitz, auke.ijspeert, aude.billard, pierre.dillenbourg}@epfl.ch

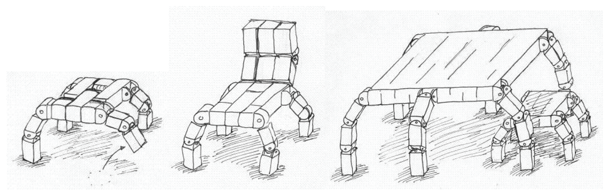


Fig. 1. Potential configuration of modules in (1) stool, (2) chair, (3) table and stool forms. The actuated joints can be used to walk.

as possible. Ideally they should be able to modify their structure upon request and reconfigure as another piece of furniture which is really required. Reconfiguration is done autonomously by sequences of movements, attachments and detachments of the modules. Therefore, modular or self-reconfigurable robots are a natural choice.

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 attaching and detaching over a lattice of other modules), thereby giving the impression that the cluster “flows” on the ground and around obstacles. The Crystalline robot [5], Telecube [6], and ATRON [7] are examples of such robots. Chain-type robots normally locomote in a static configuration (i.e. without doing reconfiguration), using powered joints. See e.g. M-TRAN [8], CONRO robot [9] and Polybot [10]. Reconfiguration is usually used to adapt to a new environment or task. We work on chain-type robots.

In this paper, we tackle the Self-Reconfiguration Planning (SRP) problem on YaMoR [11] and M-TRAN [8] modules. The goal of SRP is to find the optimal reconfiguration steps from structure A to B, given by the user. We propose a framework for SRP based on graph signature and graph edit distance. A configuration is represented in a labeled directed graph form. Graph signature provides an isomorphism-invariant code which enables us to compare different configurations and find isomorphic ones. The signature is used to prune the redundant paths and avoid solving some repeating sub-problems for multiple times. The graph edit-distance metric provides a means to judge the amount of difference (or similarity) between different configurations. The distance measure is used to guide a stochastic gradient descent method.

This paper is organized as follows: The state of the art in SRP is explained in the next section. In the third section our proposed method is described. The simulation results is presented in the forth section. The paper is finalized with conclusions and remarks for future developments.

II. Literature

A *configuration* is defined as a particular arrangement of connectivity between independent modules. A configuration can be represented in graph form, called *configuration graph* (from now on, it is referred to as graph simply) where, the vertices represent the modules and the edges represent the connection between the modules. *Self-reconfiguration* is a transition between two configurations by a series of atomic movements (attach, detach [1]). The goal of SRP is to design an optimal algorithm that minimizes the number of steps (or other measures of optimality e.g. time) required to reach a final configuration, starting from an initial configuration .

SRP for the chain-type robots is more difficult than for the lattice type because of the mechanical limitations. Individual modules must be strong enough to perform motion while lifting the weight of module chains, taking care of collision avoidance, and maintaining the stability of the whole structure. As a consequence finding the optimal solution is very difficult. Here, we prefer to find a feasible near-optimal solution in a reasonable time instead of an optimal solution. Hence, we look for a heuristics that can be executed rapidly and respond with “some” guarantee of performance.

Casal and Yim [12], [13] present a divide-and-conquer strategy to solve reconfiguration for closed-chain robots. The configuration is first decomposed into a hierarchy of small *substructures* belonging to a finite set. The sets of substrates must be topologically non-homomorphic, must occur often in the possible configurations, and reconfiguration between them must be simple. Reconfigurations between the substructures in a 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 substructures. 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 build the final configuration from that intermediate structure. The second algorithm tries to match the initial and final configuration in a hierarchical manner, i.e. first matching the number of levels, then the number of sub-structures per level, and then the size of substructures, etc.

Yoshida et al [14] presents a centralized planner for reconfiguring a group of M-TRAN modules. The planner uses macro-actions with a block of modules instead of one. Due to dealing with smaller number of substructures the planning procedure is simplified. The authors propose a two-layered motion planner consisting of a global *flow planner* and a local *motion-scheme selector*. The global flow planner searches the possible module-paths and motion-orders to provide the global cluster movement. The local motion-scheme selector verifies that the paths generated by the global planner are valid for each member of the block according to the possible motion orders.

Most approaches to SRP use stochastic optimization methods like Simulated Annealing and Genetic Algorithm, guided by a heuristic. The heuristics are usually fitness functions that reflect the amount of similarity between the configuration currently being evaluated and the desired one. Murata et al. [15] uses weighted probabilities based on potential fields to guide the stochastic search procedure.

Pamecha et al. [16] introduces the concept of distance between configurations. The distance metric is used as a cost function in conjunction with Simulated Annealing to guide the reconfiguration process. Different distance metrics are proposed: The *overlap metric* counts the number of non-overlapping modules in the configurations. The second distance metric is the *minimum number of moves* required for reconfiguration. This is not useful in practice since it needs solving the reconfiguration problem first. The third and the best distance metric in terms of performance and computation time is the *optimal assignment metric*. It tries to optimally assign the modules of the initial configuration to the ones of the final configuration so that a cost function is minimized. The algorithm that solves the optimal assignment problem, called Hungarian method [17], needs $O(n^3) \times O(d(u,v))$ where n is the number of modules. $d(u,v)$ is a function that calculates the cost of assigning a module u to another module v. If d is not calculable in $O(1)$ the total complexity would be higher than $O(n^3)$.

III. Our method

We propose a graph theoretic approach to SRP. Like many others, we use stochastic optimization methods. However, we guide the search process with two new heuristics that are showed to be simplifying the planning process: a *graph isomorphism test*, and a *similarity metric*.

The isomorphism test is a binary test that specifies whether two labeled graphs are isomorphic or not. This test enables us to cut some redundant branches in the search space and avoid solving some repeating sub-problem for multiple times.

The similarity metric provides a means for direction. It shows how much a graph looks topologically like another one. It is based on the relative size of the Maximum Common Sub-graph (MCS) of two graphs. It calculates an upper-bound for the relative size of MCS in linear time (fixed order if calculated incrementally). The similarity metric is used to assign priority to different branches such that the ones ending to graphs more similar to the goal graph have higher priorities.

A. Search Strategy

Given the initial and the final configuration graphs, connections between the modules are represented by a directional edge from male to female connectors. Genderless connectors are treated as a special case(sec. III-D). Fig. 2 shows a sketch of the search strategy that we follow in SRP. The vertical axis shows the similarity metric (ranging from 0 to 1). Each point on the paths is a configuration graph. The initial configuration is processed in order to find the list of feasible {attach,detach} actions (in conformance with mechanical limits). By executing an action on the initial

[1] *Rotations* does not change the configuration so they are not considered as an action in SRP. Otherwise, different shapes of the robot during locomotion must be considered as different configurations.

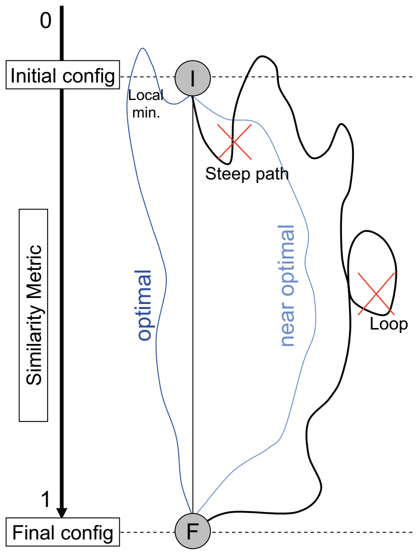


Fig. 2. A sketch of the search strategy in the configuration space. For illustration purpose, the space is shown in 2D; the real one is multi-dimensional.

graph, some new graphs are achieved (e.g. 4 new graphs are achieved from I in fig. 2). Similarity of the new graphs with the final one is calculated. A priority is assigned to each one and the one with the highest priority is selected for further expansion. In case of tallying, one of them is randomly selected. These steps are repeated until some suitable solutions are found (or another termination criteria is met).

Putting priority on the branches leads sometime to ignoring an optimal solution which includes a local minimum, like the leftmost path shown in fig. 2. In this case the planning procedure would find a near-optimal solution. This is fine for our application as far as this solution is found in a reasonable time.

During exploration of the configuration space, loops in the paths from the initial graph to the final one are inevitable. First, because the attach and detach actions are reversible (at least in theory) i.e. if two modules are connected in a planning step, one possible action in the next step would be detaching them and returning back to the configuration in the previous step. Second, because some sequences of actions can be executed in different orders and result in the same configurations.

The loops are redundant paths that have already been traversed once. In order to avoid them, we record what we call a *signature* of the previously encountered graphs. Whenever the signature of a graph is found in the records, processing that graph is not continued anymore.

B. Graph Signature

Graph isomorphism is one of the important problems in graph-theory. It is not proved yet whether it is NP-complete [18] or not. However for special cases polynomial time algorithms exist e.g. for planar graphs [19], graphs with bounded genus [20], graphs with bounded degrees [21], trees [22], ordered graphs [23], convex graphs, permutation graphs, and interval graphs. The algorithm for ordered graphs is best suited to our application due to its lower complexity. Jiang and Bunke [23] prove that these type of graphs have quadratic-time isomorphism test.

It is known that connected undirected graphs are Eulerian if and only if all of their vertices have even degrees. Degree of a vertex in an undirected graph is the number of edges incident to it. A graph is called Eulerian if an Eulerian circuit exists in the graph. An Eulerian circuit in an undirected graph is a cycle that visits each edge exactly once. In a finite connected undirected graph, it is always possible to construct a cyclic directed path passing through each edge once and only once in each direction [23]. By replacing undirected edges with two directed edges in opposite directions we can construct an Eulerian circuit. The sequence of the visited vertices while traversing an Eulerian circuit is isomorphism invariant. Therefore, the Eulerian circuit can be used to generate a *code* or *signature*.

To adopt this theory, we need to provide a means for ordering the edges of the graph. We consider the fact that each module in our application has a fixed number of connectors. Also, we assume the connectors can be connected to with a finite number of orientations. Therefore, the number of connection points and angles is finite. If a unique identifier is assigned to each case, it can be used as a label for the edges. The labeled graph is trivially transformable to an ordered graph by sorting the out-edges of the vertices in lexicographic order.

For example Fig. 3 shows a possible indexing for the connectors of a simulated YaMoR [11] module. Each module has one rotational servo in front and 6 connectors (white circles) that accept connections in 4 different orientations ($0°$, $90°$, $180°$, $270°$). So totally 144 (=$6{\times}6{\times}4$) different relative positions exist. In Fig. 3, the front connector (index 0) of the module at behind is connected to the rear connector (index 5) of the front module with $0°$ orientation, so their connection must be labeled with 20 (=$0{\times}6{\times}4{+}5{\times}4$).

Assume that out-edges of the vertices are sorted in descending order and in-edges of the vertices are sorted in ascending order. This is done only once, while inserting the edges in the graph. Computing the signature of a graph consists of the following steps: First, the biggest label in the graph is found, say l_{max} (time order: $O(|V|)$ where V is the set of vertices). Then a *vertex signature* is computed for every vertex that has out-edge with label l_{max} (best case: $O(|E|)$, worst case: $O(|V||E|)$ where E is the set of edges). The vertex signature that has the highest lexicographic order is called the signature of the graph.

Computing the vertex signature consists of a depth-first-search (dfs) with two modifications: edges can be traversed through either normal direction (from source to target) or reverse direction, and out-edges of the vertices should be traversed before their in-edges. Needless to say that the edges are traversed only once. Each vertex is assigned an index. The index reflects the order of visit i.e. the first vertex is indexed with one and each time a new vertex is discovered

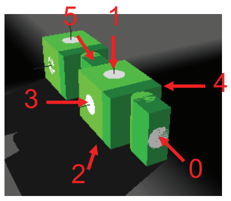


Fig. 3. Labeling the connections.

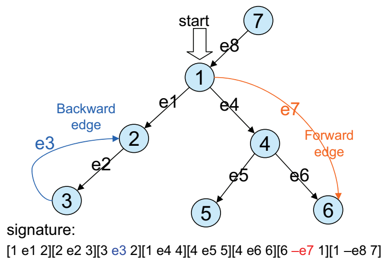

start
7
e8
1
e1
e4
e7
Backward edge
e3
2
e2
3
4
Forward edge
e5
e6
5
6
signature:
[1 e1 2][2 e2 3][3 e3 2][1 e4 4][4 e5 5][4 e6 6][6 –e7 1][1 –e8 7]

Fig. 4. An example of computing a vertex signature.

the index is incremented by one. The sequence of the encountered vertices and edges during a visit is proved to be isomorphism invariant [23]. The vertex signature is a string. When traversing an out-edge, the string is concatenated with $[v_s, l, v_t]$ where v_s and v_t are the indices of the source and target vertices and l is the edge label. For in-edges, $[v_t, -l, v_s]$ is concatenated.

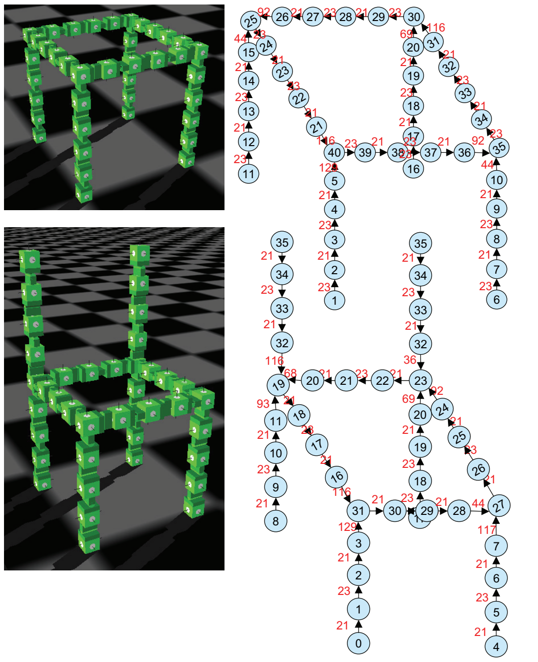
Fig. 4 shows an example of computing a vertex signature. We assume that the edges are sorted according to their label, out-edges in descending order and in-edges in ascending order. The vertex shown as start point is indexed with 1. The out-edge e_1 is traversed first. The newly visited vertex is indexed with 2 and $[1, e_1, 2]$ is added to the signature. Then vertex 3 is visited via e_2. From vertex 3, vertex 2 is reached via e_3. Vertex 2 was already visited, so we back-track to 3. From vertex 3 we have no other edges to traverse. We have to back-track to 2, and then to 1. In vertex 1 the next edge i.e. e_4 is followed. Similar steps are repeated until vertex 6. It has two in-edges, e_6 and e_7, and no out-edge. We had arrived to it from vertex 4 via e_6; so the next unvisited edge would be e_7. Since we are traversing e_7 in reverse order, $[6, -e_7, 1]$ is added to the signature. The procedure is continued until all vertices and edges are visited once and only once.

C. Similarity Metric and Graph Edit Distance

The similarity metric that we use is defined on the basis of *graph edit distance*. The graph edit distance is defined as the shortest sequence of graph edit operations, i.e. {deletion, insertion} of edges or vertices, that transform an initial graph to a final graph. Edit distance is proved to have the following relation with MCS of the input graphs, I and F [23]:

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

where in our application $|MCS|$ is the number of edges in the maximum common *edge-induced* subgraph [24], and $|I|$ and $|F|$ are the number of edges in I and F.



Label	#edges
21	16
23	16
44	2
69	1
92	2
116	2
129	1

Label	#edges
21	20
23	10
44	1
45	1
68	2
92	1
93	1
116	2
117	1
129	1

Fig. 5. An example of calculating upper-bound for the size of maximum common sub-graph. Common labels are marked with grey background in the tables.

Our desired similarity metric is one minus this distance metric: $\sigma(I, F) = \frac{|MCS(I,F)|}{\max(|I|,|F|)}$. The similarity σ is maximum (i.e.1) if the number of edges in MCS is equal to the maximum number of edges of I and F, i.e. I is isomorphic to F.

Unfortunately, finding the MCS is proved to be NP-complete [18]. However having a rough estimation is enough for our application. We calculate an upper-bound for its size that is computable in linear time. The idea of the upper-bound comes from [24] however, since we deal only with labeled graphs our formulation could be simplified.

For calculating the upper-bound we categorize the edges based on their label. We know that the necessary but not sufficient condition for two vertices to be matchable in isomorphism test is that the label of their incident edges are matchable. So, before calculating the upper-bound we fill a table for each graph (fig. 5). The columns of the table are the labels that exist in the graph and the number of edges with that specific label.

Let C_l^1 and C_l^2 be the number of edges of the input graphs that have label l. The upper-bound is:

$$\sigma_{UB}(I, F) = \frac{\sum_{l=0}^{l_{max}} \min(C_l^1, C_l^2)}{\max(|I|, |F|)} \qquad (2)$$

Needless to mention that only the labels that exist in both tables can be matched to each other. Using a hash table the upper-bound is calculated in linear time $O(\max(|I|, |F|))$.

D. Special cases

Some cases need special treatments: *genderless* connections and *symmetric* modules. Genderless connections are undirected however our method needs a direction for the

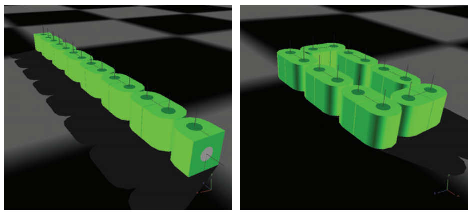


Fig. 6. (left) Line configuration (right) Ring configuration

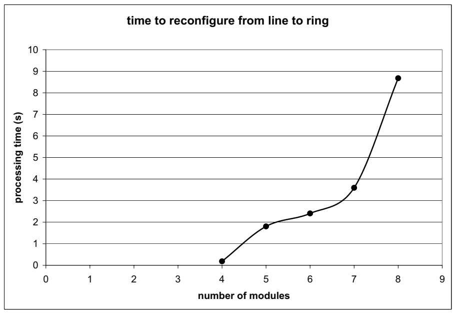


Fig. 7. Computation time to reconfigure from line to ring

edges. We choose a direction that maximizes the graph signature. That is to say, edge direction should be from connectors with higher index to connectors from lower index. When two connectors have the same index (e.g. front to front connection) both directions should be tried to see which direction leads to a bigger signature.

Some modules have symmetry in their geometry e.g. YaMoR modules are symmetric around the line that connects the connector 0 to 5 (fig. 3). So, the configuration in fig. 3 looks similar to the case where the front module is rotated 180° around this line. In this case, an orientation that maximizes the signature is selected. So when running isomorphism test on two symmetric modules, the largest signature that can be acquired from a graph like them should be compared to each other.

IV. Results

The proposed method has been tested on the simulated versions of YaMoR [11] and M-TRAN [8]. M-TRAN modules are composed of 2 rotational servos and 6 connectors. YaMoR modules are like half a M-TRAN; so due to this similarity and the sake of saving the space we mention only the results on M-TRAN. Different reconfiguration problems have been executed. Except some very special cases, the results are similar to the ones that are presented here. The connectors were set to be genderless. The planning process is expected to run even faster on male-female connectors because of the limitation on the number of possible actions and therefore smaller search space.

A. *Reconfiguration from line to ring*

In order to differentiate the performance of our planner from some overhead procedures (e.g. finding the next feasible actions), which have to be executed anyway regardless of the planner, we let the program solve the reconfiguration

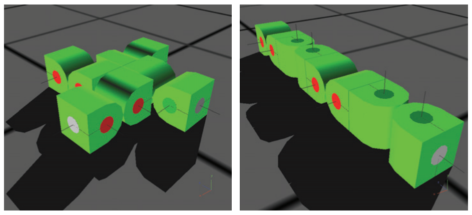


Fig. 8. (left) Quadruped configuration (right) Snake configuration

problem from a line configuration (Fig.6 left) to a ring (Fig.6 right). We know that the line configuration can be reconfigured to the ring by only one attachment. It needs examining only one configuration graph. So only one iteration of the algorithm is sufficient. This can reflect the overhead time.

We have executed the program several times for different number of modules[2]. Fig.7 shows that the processing time increases exponentially with the number of modules (number of servos, precisely speaking). The reason is that, in order to find the next possible actions we have to find feasible detachment / attachment points. Finding possible detachment points is relatively simple; we need only to find circuits in the configuration graph. Finding the possible attachment points is time consuming; we have to iterate over all possible shapes (servo positions) within the same configuration, and find the connectors that are perfectly aligned. Although we discretize the servo positions at 90°s (so servos can be in -90°, 0°, and +90° positions), the combination of possible positions is still a lot, increasing exponentially with the number of servos (3^n, n being the number of servos). This is one of the overheads that imposes a big delay on the reconfiguration process. If it could be replaced with another more efficient method, the whole procedure could speed up a lot.

For the configurations composed of more than 5 modules, the timing changes a bit. This is due to the growth of the search space. The main memory is rapidly filled. After that, the whole procedure is delayed due to working with virtual memory.

B. *Reconfiguration from quadruped to snake*

The problem that we select as a benchmark for the rest of the experiments is reconfiguration of four modules from a quadruped configuration (Fig.8 left) to a snake (Fig.8 right).

We show the performance of our algorithm in processing time and quality of the found solutions. A set of around 500 experiments (with different random seeds) were executed. For each experiment, the number of encountered graphs before finding a solution, and the number of actions in the solution were recorded. In each experiment, the planner continued running until at least 20 different solutions were found.

1) The best sequence: The best solution ever found for this reconfiguration problem consists of 9 attach/detach actions, however none of the experiments could find it at first iteration. This means the similarity metric has came across a

[2]Computer spec: Intel® Core™2, CPU 6600 @ 2.4 GHz, 2 GB RAM

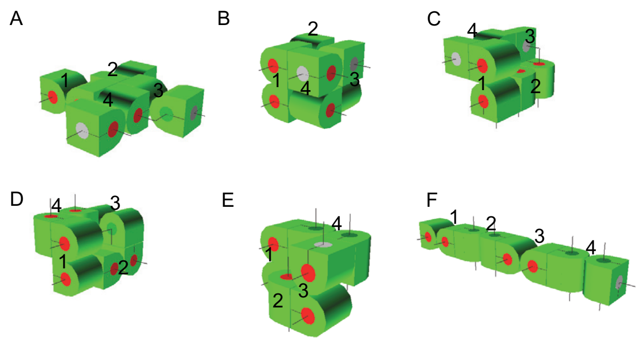


Fig. 9. The best solution ever found for reconfiguration from quadruped to snake: A→B: attach module 4 to 1 (from the green-colored connector on module 4 to the grey-colored connector on module 1 with 0° orientation); B→C: detach 2 from 3, detach 4 from 1, attach 2 to 3 (grey to grey, 90°); C→D: detach 4 from 1, attach 4 to 1 (grey to grey, 90°); D→E: detach 1 from 2, attach 1 to 2 (grey to grey, -90°); E→F: detach 3 from 4

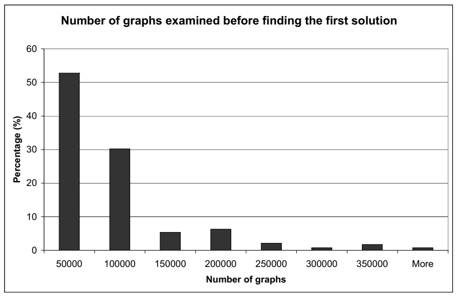


Fig. 10. Number of graphs examined before finding the first solution

local minimum in the path from the quadruped to the snake configurations. The sequence of reconfiguration is shown in Fig. 9 (servo positions are not mentioned). The found solution can be applied to both genderless and male-female connectors.

The solution that is provided here is not guaranteed to be the optimal solution. In order to prove its optimality we had to do a complete search in the configuration space. Our analysis showed that for this specific problem the configuration space grows by a rough factor of 16 (from each configuration graph, around 16 new graphs are generated). So, to be sure at least $16^{9-1} \approx 4.3 \times 10^9$ graphs must have been evaluated. This means more than 25 years of computation time, which was impossible!

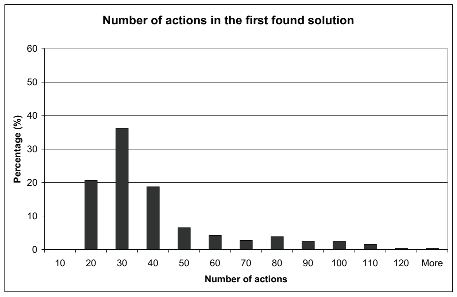


Fig. 11. Number of actions in the first found solution

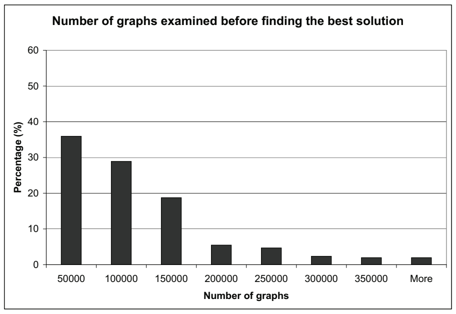


Fig. 12. Number of graphs examined before finding the best solution among the twenty found solutions

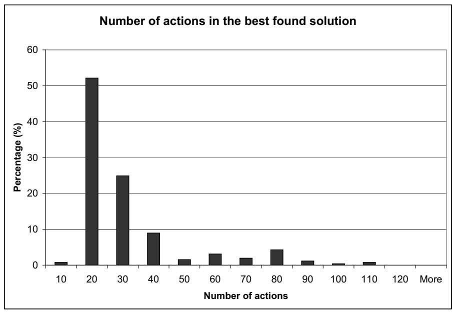


Fig. 13. Number of actions in the best found solution among the twenty found solutions

2) The first found solution: Fig. 10 shows the histogram (shown as percentage of the whole experiments) of the number of graphs encountered before finding the first solution. The first solution is found 83% of the times before examining up to 100k graphs. Moreover, in 53% of the cases the solution is found earlier, before examining 50k graphs. This is a good news; it means we have a solution available in a short time. The planner continues running in order to enhance the found solution. Meanwhile, we can either wait for the next solutions to come or stop the planner and survive with the available ones.

Fig. 11 shows the histogram (percentage) of the number of actions in the first found solution. As mentioned earlier none of the experiments could find the best solution at first time. 21% of the time the solution consists of up to 20 actions; 57% of the solutions consist of up to 30 actions; and 76% consist of up to 40 actions. Most of the experiments end up with a first solution that consists of 20-30 actions. This means the first found solution is not too far from the optimal solution.

From these results we can argue if the proposed algorithm could not find any solution in a "short" time, two cases could be imagined: either (1) no solution with a "short" sequence of actions exists, or (2) the planner is going in a wrong direction and if any solution is found it would be far from optimality. Hence, we have to change the search path or even change the planning strategy.

3) The best found solution: Fig. 12 shows the histogram (percentage) of the number of graphs encountered before finding the best solution among the 20 found solutions.

Compared to Fig. 10, percentage of 50K bin is decreased and percentage of the other bins are increased. The largest increment belongs to the 150K bin. However 50K bin is still in majority. Fig. 13 shows the histogram (percentage) of the number of actions in the best solution among the the 20 found solutions. This time most of the experiments end up with a best solution that consists of 10-20 actions. The results show that the planner is getting very close to the optimal solution, however the price that we have to pay is a little bit more computation time. Considering the quality of the found solutions, a bit more computation time is negligible.

V. Conclusion

We have developed a framework for self-reconfiguration planning which is based on the graph signature and the graph edit-distance. The graph signature method proved to be a very fast test for isomorphism of the configurations. Recording the signature of the encountered configurations enabled us to cut some redundant paths. A similarity metric was introduced based on the graph edit-distance. The metric rapidly calculates an upper bound for the size of the maximum common sub-graphs. The results showed that a near-optimal solution could be found rapidly. This means the similarity metric creates a good gradient for the search procedure.

The notions of graph signature and graph-edit distance are very general and can be used with any other search strategies. A possibility would be converting the priority among the branches to probabilities. This is sometimes a good way of escaping from local optimums.

However the computation time is still slow due to the exponential growth of the configuration space, inverse kinematic problem, and collision avoidance issue. We are looking for effective solutions to these problems.

We are looking for improving the design of the YaMoR modules as the current version is not suited to serve as a basis for the future Roombots. We believe the reconfiguration planner would gives us some hints about the necessary mechanical characteristics of the future module, e.g. shape, degrees of freedom, and torque limits.

VI. ACKNOWLEDGMENTS

We would like to thank Y. Bourquin for his help on the 3D simulations. This work made possible thanks to funding from Microsoft Research Labs and the EPFL (FIFO grant).

References

[1] S. Jordà and G. Geiger and M. Alonso and M. Kaltenbrunner, "The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces," in *Proceedings of the first international conference on Tangible and Embedded Interaction (TEI07)*, Baton Rouge, Louisiana, 2007.

[2] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 66–75, Sep 1991.

[3] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 34–47, 2001.

[4] Norbert Streitz and Peter Tandler and Christian Müller-Tomfelde and Shin'ichi Konomi, "Roomware: Towards the next generation of human-computer interaction based on an integrated design of real and virtual worlds," in *Human-Computer Interaction in the New Millenium*, J. Carroll, Ed. Addison-Wesley, 2001, pp. 553–578.

[5] 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.

[6] 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.

[7] 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.

[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, 2006, pp. 115–124.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[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] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji, "A 3-d self-reconfigurable structure," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1998, pp. 432–439.

[16] 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.

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

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

[19] I. S. Filotti and J. N. Mayer, "A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus," in *Proc. of the 12th Annual ACM Symposium on Theory of Computing*, 1980, pp. 236–243.

[20] G. Miller, "Isomorphism testing for graphs of bounded genus," in *Proc. of the 12th Annual ACM Symposium on Theory of Computing*, 1980, pp. 225–235.

[21] 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.

[22] A. V. Aho, J. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

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

[24] 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.