Hierarchical Motion Planning for Self-reconfigurable Modular Robots

Preethi Bhat[1] James Kuffner[1] Seth Goldstein[1] Siddhartha Srinivasa[2]

[1] The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, USA
{preethi, kuffner, seth}@cs.cmu.edu

[2] Intel Research Pittsburgh
4720 Forbes Avenue, Suite 410
Pittsburgh, PA 15213, USA
siddhartha.srinivasa@intel.com

***Abstract*— Motion planning for a self-reconfigurable robot involves coordinating the movement and connectivity of each of its homogeneous modules. Reconfiguration occurs when the shape of the robot changes from some initial configuration to a target configuration. Finding an optimal solution to reconfiguration problems involves searching the space of possible robot configurations. As this space grows exponentially with the number of modules, optimal planning becomes intractable. We propose a hierarchical planning approach that computes heuristic global reconfiguration strategies efficiently. Our approach consists of a base planner that computes an optimal solution for a few modules and a hierarchical planner that calls this base planner or reuses pre-computed plans at each level of the hierarchy to ultimately compute a global suboptimal solution. We present results from a prototype implementation of the method that efficiently plans for self-reconfigurable robots with several thousand modules. We also discuss tradeoffs and performance issues including scalability, heuristics and plan optimality.**

I. Introduction

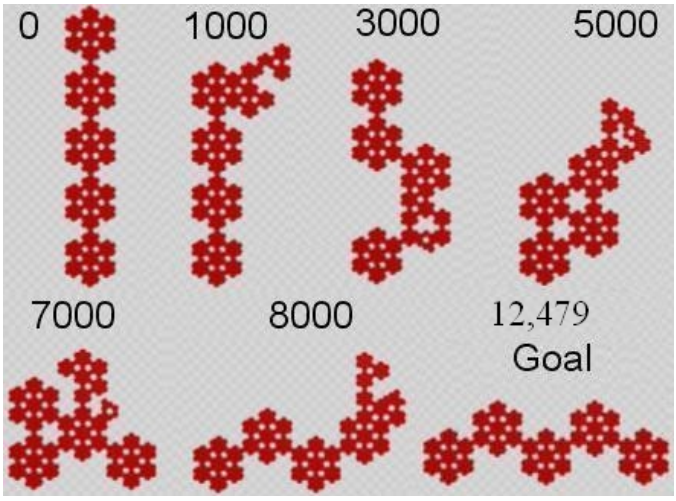


Fig. 1. An example plan for module motions on the roadmap with 12,005 catoms. The number labels indicate the number of module moves.

A self-reconfigurable modular robotic system comprises of a collection of homogeneous modules that can connect, disconnect, and move around adjacent modules. The system is capable of reassembling itself to a desired target configuration. Reconfiguration occurs for purposes of locomotion, manipulation, or for the creation of stable static structures. Motion planning for such systems is especially challenging as it involves coordinating the movement of a possibly large number of modules while enforcing motion constraints imposed by their physical design.

A motion plan is a sequence of module motions that changes the shape of the system from a start configuration to a goal configuration while enforcing constraints such as avoiding collision and maintaining connectivity. Optimality for such reconfiguration strategies can be measured in different ways such as minimizing the number of module moves, minimizing time for reconfiguration or minimizing energy consumption during reconfiguration. In this paper, we focus on generating plans that minimize the number of module moves.

In order to find an optimal solution, we can explore the discrete configuration space, guided by heuristics. This approach is feasible if the number of modules is small. However, because the configuration space grows exponentially with the number of modules, finding an optimal plan for a large number of modules quickly becomes intractable. Experimental results have shown that finding an optimal plan for more than *a few dozen* modules using classical search algorithms such as A* takes a few hours to compute [1]. In many cases, the planner fails to discover a feasible path. Thus, research efforts have primarily focused on determining good heuristics for suboptimal planners, or devising purely local methods for reconfiguration.

We present a hierarchical approach to motion planning for systems with many thousands of modules. Given the high cost of computing optimal plans using classical search algorithms, we focus on divide-and-conquer techniques and on the efficient reuse of existing plans. We impose a hierarchy by recursively reconfiguring groups of modules into meta-modules. Reconfiguration results in a fractal structure — a meta-module at level i of the hierarchy is comprised of a group of structurally identical but smaller meta-modules of level $(i-1)$. We exploit the self-similarity of the structure by pre-computing scale-invariant motion templates. A *template* describes a motion of meta-modules at level i of the hierarchy

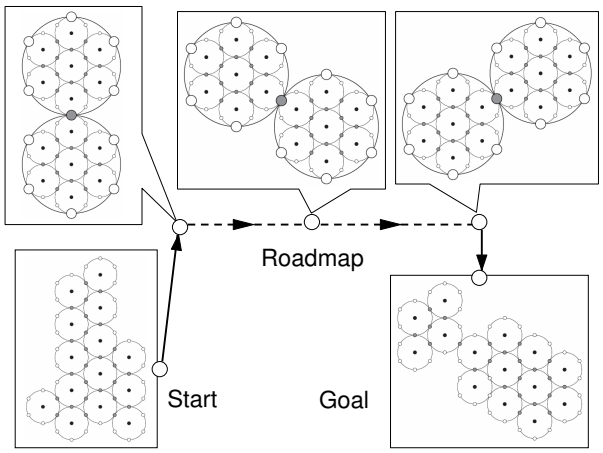


Fig. 2. Conceptual view of a high-dimensional configuration space roadmap.

in terms of the motion of the constituent meta-modules of level $(i-1)$. Applied recursively, a template describes a move at any level of the hierarchy in terms of the motion of the original modules (which constitute level 0 of the hierarchy). Likewise, a motion plan at any level of the hierarchy, comprised of a sequence of moves, can be translated to a motion plan for the modules with no extra planning cost. Since higher levels of the hierarchy contain fewer meta-catoms, a motion plan at that level can be computed quickly using classical A* search.

Conceptually, one can view the hierarchical planner as a special kind of roadmap in the very high-dimensional configuration space of the modules, as shown in figure 2. It provides a fast path from configurations *close* to the start and goal configurations, much like a highway connecting two cities. Admittedly, while getting to the highway is easy (in most cities), reaching and exiting the roadmap from an arbitrary configuration might potentially involve generating a plan for all of the modules. However, we believe that, for configurations spaced far apart in configuration space, this cost is generally far less than the cost of computing an optimal plan without using the roadmap. Figure I illustrates a plan for module motion on the roadmap. The path begins at the hierarchical start configuration labeled 0 (far left), and intermediate configurations after a few steps in the roadmap are shown. The labels represent the number of steps needed to reach that configuration. The path finally terminates at the hierarchical goal configuration in the roadmap (far right). Traversing the roadmap is fast and the total planning time is on the order of a few minutes. Our hierarchical algorithm is targeted at the *Claytronics* project. The goal of this project is to create dynamically deformable objects modeled using a large number of homogeneous modules called *catoms* (Claytronics atom). A Claytronics ensemble would be capable of dynamically rearranging itself to maintain the shape of the object at any instance of time. The current physical catom prototype is a cylinder with magnets on its surface (Figure 3). Catom movement is achieved using electro-magnetic forces.

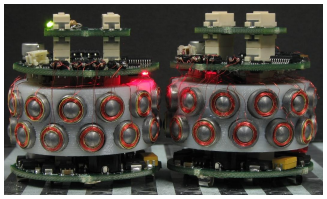


Fig. 3. Physical prototype of two dimensional catoms.

The catoms move about each other by turning successive magnets on or off. Our hierarchical planner has been used to plan for the reconfiguration of several simulated Claytronics ensembles from a given initial configuration to a target goal configuration (Figure 3).

II. Related Work

Different approaches have been proposed to address the problem of motion planning for self-reconfigurable systems. Biologically inspired approaches using hormones to direct reconfiguration has been well explored. Bojinov *et. al* [2] use local sensing and local control rules to achieve motion planning. Butler *et. al* [3] propose a generic algorithm based on cellular automata that uses a set of rules to perform tasks such as locomotion and navigating tunnels. Kubica *et. al* [4] use simple local rules inspired by local insects to produce complex behaviors, by local reactive path planning technique to achieve reconfiguration. Shen *et. al* [5] propose biologically inspired protocols for adaptive communication and distributed collaboration between modules to achieve global effects such as locomotion.

Decentralized planning has been investigated by many including Kubica *et. al* [4], Murata *et. al* [6], and De Rosa *et. al* [7]. De Rosa *et. al* [7] explore reconfiguration using distributed manipulation of regularly shaped voids in the structure. Stoy *et. al* [8] achieve reconfiguration by directing the growth of the configuration using seeds that communicate locally. A more centralized approach to motion planning has been explored by Yoshida *et. al* [9]. Nguyen *et. al* [10] explore some of the properties of modular robots in a 2D hexagonal lattice and offer an interesting approach to reconfiguration using scaffolding-like structures.

Two-phase strategy for motion planning uses a global planner to provide high-level plan and a local motion planner to translate it to the modules. Yoshida *et. al* [11] propose a local motion planner that uses local rules stored in a database. Walter *et. al* [12] initially find all admissible paths for reconfiguration and heuristically ranks them. In the second stage, a deterministic, distributed algorithm is used to achieve reconfiguration using little intermodule message passing. Zhang *et. al* [13] propose a general constraint-based control framework for modular self-reconfigurable robots where constraint solvers are goal oriented deliberative agents that can be used as control regulators or as information retrievers. Pamecha *et. al* [14]

discuss some useful heuristics and configuration metrics that can be applied to modular robot motion planning.

III. The problem

While our planning algorithm can be applied to any self-reconfiguring system, our specific implementation platform is the Claytronics system. In this section, we describe the system and its pertinent notation, describe the motion constraints that are specific to our system, and define the self-reconfiguration problem.

A Claytronics system is comprised of a large number of modules called *catoms* (short for Claytronics atoms). We envision the system to be comprised of many thousands of tiny spherical catoms (of diameter $2mm.$) whose surfaces are tesselated with mechanisms for attachment and detachment with other catoms. However, the current physical catom prototype is a cylinder of diameter $44mm.$ and height $40mm.$ constrained to move on a planar power strip. Electromagnets on the surface of the cylinder are turned on and off for attachment and detachment with other catoms. In this paper, we focus on motion planning for the planar Claytronics system.

In the plane, a catom is a circular entity. It attaches to other catoms via six electromagnets spaced uniformly on its circumference. A *configuration* is a collection of connected catoms. Due to the discrete number of connectors, catoms in a configuration occupy discrete nodes in a hexagonal planar grid. We describe a node by a pair of integers $C \in \mathbb{I}^2$. A configuration q of n catoms is a collection of occupied nodes given by:

$$q = \{C_1, C_2, \ldots, C_n\}$$

Furthermore, a configuration is *globally connected*, *i.e.*there exists at least one connected path from every catom in the configuration to every other.

A catom in a configuration can move about an adjacent connected catom if it satisfies certain motion constraints. For the planar system, we define the set of valid moves to be clockwise or counter-clockwise rotations:

$$\text{dir} = \{\text{CW}, \text{CCW}\}$$

Each move changes the configuration q of the system to q'. This is encapsulated in the function:

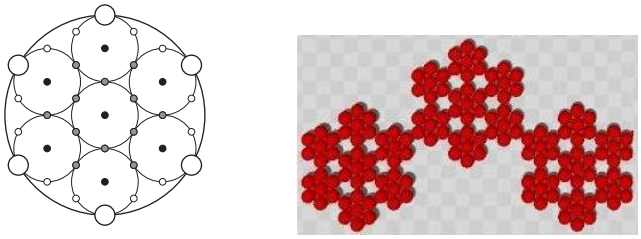
$$q' = \text{Move}(q, C_m, C_a, \text{dir})$$

where C_m is the catom which moves and C_a is the anchor catom about which the moving catom rotates. The move satisfies the following constraints.

1. C_m and C_a are neighbors.
2. q' is fully connected.
3. A move does not introduce collision among catoms.

A *plan* is a sequence of moves applied to a start configuration q_s to produce a goal configuration q_g. It is a solution to the self-reconfiguration problem.

If each configuration can be represented as a vertex in a *transition graph* [14] whose edges are moves, the number of vertices is exponential in the number of catoms. Hence, an exhaustive search of the space of configurations is infeasible, necessitating searches based on informed heuristics. An informed heuristic must be *permutation invariant*, *i.e.*, it must reflect the fact that if the catoms in a configuration q were numbered, all re-numberings of the catoms produce the same configuration q.



(a) Schematic representation of a meta-module

(b) Sample hierarchical configuration output from the planner

Fig. 4. Structure of a meta-module in a hierarchy

In Section IV, we describe a hierarchical planner that reduces searching by repeatedly reusing existing paths. As a result, we are able to plan for many thousands of catoms. We also describe two permutation invariant heuristics used in our searches and compare their effectiveness.

IV. Hierarchical planner

The hierarchy we utilize has a structural pattern enforced for motion template reuse. Planning is done at the top-level of the hierarchy and templates are reused at lower levels to ultimately generate the catom-level plans. The motion templates are precomputed.

A. The Hierarchy

We build hierarchies using catoms and abstractions of catom groups called *meta-catoms*. Figure 4(a) is a schematic representation of a meta-catom. A meta-catom MC at level i is a collection of seven meta-catoms at level $i-1$ (catoms if level $i-1$ = 0) forming a hexagonal structure as shown. The meta-catom at level i has six active magnets in its periphery, represented by the large circles. The other magnets do not take part in motion at level i. Hence, a meta-catom at any level is functionally similar to a catom. A configuration is a collection of catoms or meta-catoms. Figure 4(b) represents a sample hierarchical configuration output from the planner that has 3 meta-catoms at the highest level.

The hierarchical approach can be intuitively understood as getting on a roadmap from the start configuration, traversing this roadmap until we get close to the goal and then getting off the roadmap to the goal configuration. The process of getting on and off the roadmap corresponds to configuring in and out of a hierarchical template structure. This step could potentially be very difficult to compute. There are different approaches to solving this problem, such as utilizing carefully-defined local rules to quickly transition from a configuration to a hierarchy template configuration. Alternatively, a greedy approach could iterate through the hierarchy, greedily aggregating the first few modules to form meta-modules where each iteration results in

a new level of the hierarchy. The structure imposed on the hierarchy enables us to pre-compute optimal moves offline in the form of motion templates and reuse these templates at any level of the hierarchy. If we have motion templates for all moves, we need to plan only at the top-level of this high-dimensional configuration space with a few meta-modules using classical A* search techniques.

B. Motion templates

Motion templates are precomputed motions that are recursively reused to generate catom-level plans. A template takes as input a move at level i and outputs a sequence of moves at level $(i-1)$ that produce the required motion.

An instance of a motion template is shown in figure 5. The large circles represent meta-modules at level i of the hierarchy. Each meta-module is comprised of seven meta-modules of level $(i-1)$, as shown by the small circles. The input move is a CW rotation of the center meta-module about its anchor. The start and desired goal configurations of the move are shown in the top left and lower left of the figure, respectively.

The meta-modules at level $(i-1)$ of the moving meta-module are numbered from 0 to 6. The output of the template is a motion plan for these seven meta-modules. The motion plan is generated by an A* search for the seven meta-modules. Two intermediate configurations of the resulting motion plan are shown in the top right and bottom right of the figure.

To ensure global connectivity and maximum reuse, templates are designed for a worst case scenario where all adjacent meta-modules at level i are assumed to be present. By doing so, the same template can be reused regardless of the number of neighboring meta-modules actually present. Gray circles in the figure represent the neighboring meta-modules and the global connectivity of the entire structure is maintained.

For our implementation, we use two templates — the CW template described above and a CCW template. Generating templates is cheap as it involves a one-time cost of planning for the motion of seven meta-modules and a memory cost of storing the motion plan. Using more templates brings the final catom-level plan closer to the optimal solution.

C. Base Planner

The base planner plans at the top most level of the hierarchy and also generates motion templates. It uses classical A* search, guided by heuristics to plan for a small number of modules.

In the *greedy nearest neighbor* heuristic, we compute the cost of a configuration by comparing it to the goal configuration in the following manner: The cost of every module is the Euclidian distance between itself and its nearest neighbor in the goal configuration, normalized by the catom diameter. The cost of the entire configuration is the sum of the module costs for all modules in the configuration. This heuristic causes the search to expand nodes with increasing cost. This is a permutation independent heuristic that is always optimistic (and thus admissible). Hence, A* is guaranteed to produce an optimal path in terms of the number of module moves.

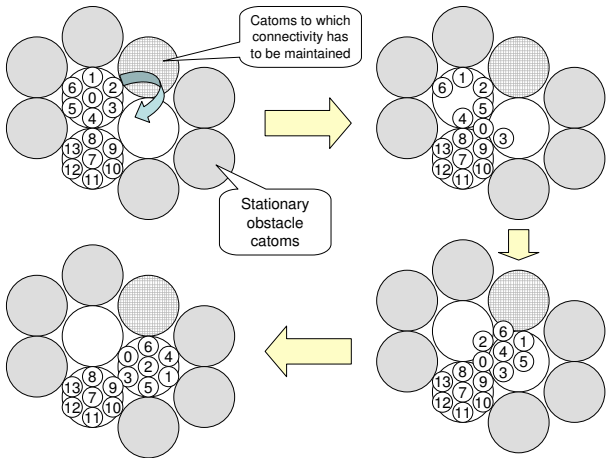


Fig. 5. Template for a clockwise move. Gray circles represent obstacle catoms and the hashed circle represents the catom to which connectivity has to be maintained during the move

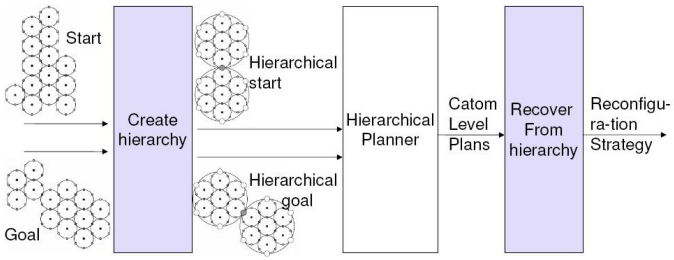
We have also experimented with another heuristic we call *greedy nearest mismatched neighbor*. In this heuristic, we initially preprocess the current and goal configurations such that all modules that are already in their goal position in the current configuration are removed and the corresponding module from the goal configuration are also removed. We then calculate the cost in the same manner as the greedy nearest neighbor. Intuitively, this means that the modules attempt to move towards the free goal positions and not towards positions that are already occupied. It also has the implication that the heuristic tries not to disturb modules that are already in their final goal positions. This heuristic is also admissible and hence A* returns an optimal plan.

We compare the performance of these two heuristics in section V. The main reasons for choosing the above two heuristics are simplicity, ease of implementation, speed and optimality. We are exploring other alternative heuristics that can be applied to this problem.

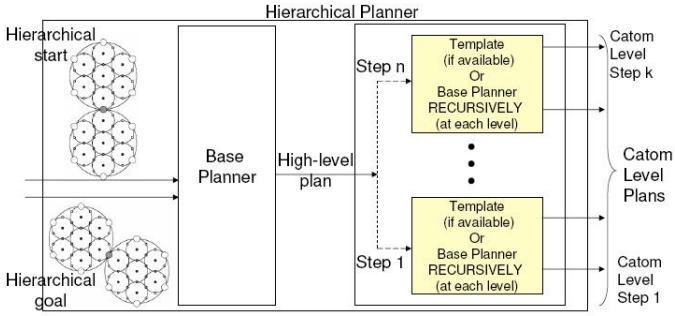
We prune the search space by applying only valid actions. Valid actions are moves that satisfy all of the constraints mentioned in Section III and generate a valid configuration. Experimental results have shown that constraining the expansion of a node to valid children configurations reduces the branching factor significantly.

D. Hierarchical Planning Algorithm

The flowchart representing the overall approach is shown in figure 6(a) and the flowchart representing the working of the hierarchical planner is shown in figure 6(b). The start and the goal configurations are initially converted to a hierarchical structure. This is not done automatically in our prototype planner. A perfect hierarchy can be formed with multiples of powers of 7 catoms. If, however, the configuration does not have multiples of powers of 7 catoms, motion templates need to be created for meta-catoms with less than 7 catoms. The



(a) Overview of the reconfiguration approach.



(b) Overview of the hierarchical planner.

Fig. 6. Flowchart illustrating the reconfiguration approach.

base planner is then called to plan at the top most level of the hierarchy to reconfigure the meta-modules from the start shape to the goal shape.

A single step at level i is translated to a sequence of steps at level $i-1$ by either reusing the plan given by a motion template, if available, or by calling the base planner to generate a plan. Every step of the plan generated at the top most level of the hierarchy is thus translated in a depth-first fashion till it represents catom level moves. Finally, we transform from the hierarchical structure to the goal configuration with the same techniques used for generating the hierarchies.

The base planner generates optimal plans, and the pre-computed motion template plans are also optimal. However, since we are combining the plans at each level of the hierarchy, the catom-level plans are not globally optimal. This loss of optimality is a tradeoff for the increase in speed of the planner, and the scale of the problems that can be solved practically. In addition, because our approach reasons about reconfiguration at a global level, our method has been observed to converge more consistently than purely local, rule-based methods. Further analysis needs to be done regarding the expected convergence rates.

V. Results

Experimental results show that the greedy nearest mismatched neighbor heuristic (branching factor of 5.4) increases the performance of the planner by expanding about 2.5 times less nodes than greedy nearest neighbor (branching factor of 7). However, the planner spends more time computing the greedy nearest mismatched neighbor (35.53 % of planning time) when compared to greedy nearest neighbor (3.77 % of planning time) heuristic. We ran experiments with four test

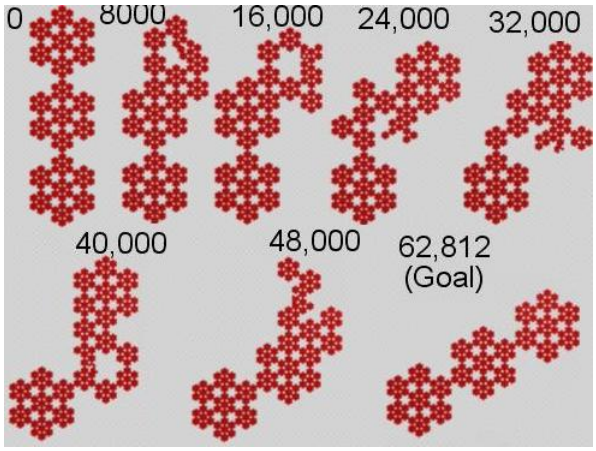


Fig. 7. Motion plan for 4 levels, 3 meta-catoms, 1029 catoms and 62,812 catom steps

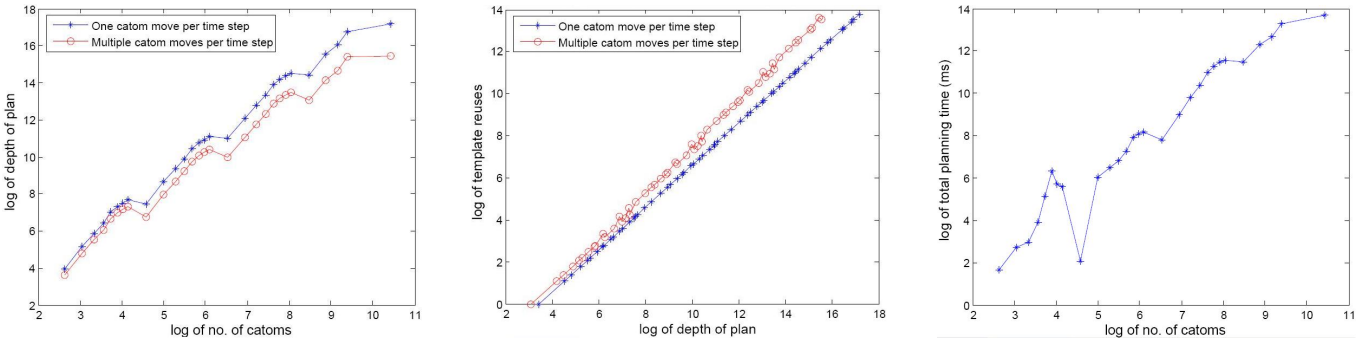
suites, each suite containing 50 problem instances. The planner reconfigured 33,614 catoms in 14.98 minutes with more than 28 million steps for reconfiguration. The planner can plan efficiently, even for systems with many thousands of catoms. In our experiments, we were limited to plan for configurations of at most 33,614 catoms due to a memory limit of 512MB in our implementation. The actual planning time is usually on the order of a few seconds.

Figure 7 shows an example of a hierarchical start configuration with 4 levels, 3 meta-catoms which translates to 1029 catoms. The transition from the start to the goal takes 62,812 catom steps.

In figures 8(a) and 8(b), the depth of a plan represents the number of module moves needed to reconfigure from an initial configuration to the goal. We identify some simultaneous moves by parallelizing the template and acknowledge that such a plan is not completely parallel. Figure 8(a) illustrates that as the complexity of the configuration increases, more moves are needed for the reconfiguration. It also illustrates that when multiple moves are permitted at each time step, the total number of time steps needed is reduced substantially. Figure 8(b) illustrates that as the number of moves in the plan increases, the number of times the templates were reused increases linearly. Figure 8(c) shows that as the number of catoms in the configuration increases, the time taken to translate the high-level plan generated at the top-most level of the hierarchy to the catom-level plan by the use of the templates also increases.

VI. Summary and Discussion

This paper presents a global hierarchical motion planning algorithm for self-reconfigurable modular robotic systems. We impose a hierarchical structure from input configurations, and utilize a recursive planner that plans at the top-level of the generated hierarchy by invoking a base planner. The base planner is implemented using classical A* search guided by novel admissible heuristics. The high-level plan is then translated to the catom-level plan by traversing through the



(a) No. of catoms Vs. Depth of plan (b) Depth of plan Vs. Template reuse (c) No. of catoms Vs. Total planning time

Fig. 8. Analysis of the hierarchical planner

hierarchy and generating intermediate plans that exploit precomputed template moves. These intermediate plans are then combined to produce the final low-level catom moves.

Experimental results have shown that the prototype hierarchical planner can easily handle planning for systems with a few thousand catom modules. The planner has also been designed to prune the search space to the space of feasible configurations and encodes the permutation independent nature of the problem to reduce the branching factor of the search.

There are several drawbacks to our approach, which form the basis of our future work. Generating a suitable hierarchical division and assignment from an arbitrary input configuration may be difficult. The number of meta catoms at the top level of the hierarchy should be small enough for the base planner to handle. Moreover, the final global plan generated by our planner is not optimal with respect to the number of catom moves. However, this loss of global optimality is acceptable, since generating truly optimal plans for such a large number of modules is currently intractable, and we have observed our method to consistently generate reasonably efficient reconfiguration plans. Our current implementation only plans for structures in two dimensions. However, it is relatively straightforward to extend the method to 3D. Heuristics affect the performance of the planner significantly, and hence more research should be done to identify better heuristics for this problem domain. Finally, our current implementation of the planner is oblivious of the physics involved and does not consider the structural stability of the resulting configurations. Robustness of execution and contingency planning has to be explored. These issues need to be addressed prior to porting the planner to run on physical prototype Claytronics systems.

VII. Future work

We intend to continue exploring efficient ways of generating hierarchies from input configurations. Determining good heuristics to guide the planner is difficult, but affect performance significantly. Other areas of future work include exploring ways to compute minimal sets of meta-moves for common intermediate configurations, incorporating stability of generated configurations with the constraint, and parallelizing and distributing the plan for simultaneous catom moves.

Acknowledgements

This work was supported in part by NSF grant CNS-0428738, DARPA Contract N66001-04-1-8931, Intel Research, and Carnegie Mellon University.

References

[1] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff, "Evaluating efficiency of self-reconfiguration in a class of modular robots," *Journal of robotic systems*, vol. 13, no. 5, pp. 317 – 338, 1996.

[2] H. Bojinov, A. Casal, and T. Hogg, "Emergent structures in modular self-reconfigurable robots," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2000.

[3] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for a class of self-reconfigurable robots," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2002.

[4] J. Kubica, A. Casal, and T. Hogg, "Complex behaviors from local rules in modular self-reconfigurable robots," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2001.

[5] W.-M. Shen, B. Salemi, and P. Will, "Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots," *IEEE transactions on Robotics and Automation*, vol. 18, no. 5, Oct 2002.

[6] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME transactions on mechatronics*, vol. 7, no. 4, Dec 2002.

[7] M. D. Rosa, S. Goldstein, P. Lee, J. Campbell, and P. Pillai, "Scalable shape sculpting via hole motion: Motion planning in lattice-constrained modular robots," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2006.

[8] K. Stoy and R. Nagpal, "Self-reconfiguration using directed growth," in *Int'l Symposium on Distributed Autonomous Robotic Systems*, 2004.

[9] E. Yoshida, S. Murata, A. Kaminura, K. Tomita, H. Kurokawa, and S. Kokaji, "Motion planning of self-reconfigurable modular robot," in *Proc. of Int'l Symposium on Experimental Robotics*, 2000.

[10] A. Nguyen, L. Guibas, and M. Yim, "Controlled module density helps reconfiguration planning," in *In Workshop on the Algorithmic Foundations of Robotics*, March 2000.

[11] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, "A motion planning method for a self-reconfigurable modular robot," in *Proc. of the IEEE Int'l Conf. on Intelligent Robots and Systems*, 2001.

[12] J. E. Walter, E. M. Tsai, and N. M. Amato, "Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots," *IEEE transactions on Robotics*, pp. 1042–296, 2005.

[13] Y. Zhang, M. P. Fromeherz, L. S. Crawford, and Y. Shang, "A general constraint-based control framework with examples in modular self-reconfigurable robots," in *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2002.

[14] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian, "Useful metrics for modular robot motion planning," in *IEEE Transactions on Robotics and Automation*, vol. 13, 1997.