Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots

Robert Fitch
National ICT Australia
University of New South Wales
Sydney, NSW Australia
robert.fitch@nicta.com.au

Zack Butler
Department of Computer Science
Rochester Institute of Technology
Rochester, NY USA
zjb@cs.rit.edu

I. Introduction

For large self-reconfiguring (SR) robots, any algorithm that requires linear space per module (with respect to the number of modules) or linear time computation or communication per actuation is undesirable. Unfortunately, many existing algorithms require linear time or space (e.g. to achieve arbitrary configurations, goal configurations must use linear space). However, for locomotion, sublinear algorithms are feasible – a complete goal specification is not necessary, reducing space requirements, and it is possible for the modules to move based on only local information, reducing the communication per move. There are two critical aspects of any such algorithm: all modules must be able to use local planning to find and execute paths that ensure that the robot does not become disconnected, and since many modules must move in parallel to enable time-efficient motion, paths must be maintained in the face of the changing topology of the robot. In this paper we describe a Reinforcement Learning technique that performs reliable locomotion among arbitrary obstacles. Planning is done for all modules simultaneously through distributed dynamic programming, while modules use local constant-time search and module locking to ensure physical integrity of the robot while following their paths. We also look after the dynamics of the system by modifying the planning to prefer locations closer to the ground. We expect this sublinear approach will enable locomotion planning at scales not previously possible, such as a million-module robot.

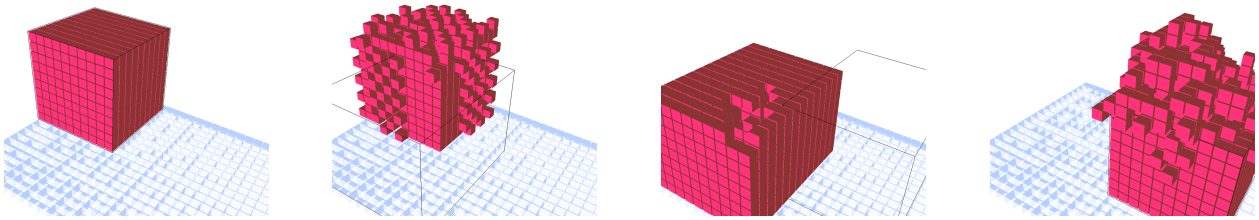
Related Work: Most algorithms for reconfiguration of lattice-based SR robots assume an exact specification of the desired shape and generate explicit module paths. One exception is the work of Stoy and Nagpal [4] which uses a representation independent of the number of modules in the system. Similarly, we use a simple bounding box to specify the goal of locomotion, though the goal could easily be made more detailed. In the work of Yim et al. [6], an exact shape is specified, but only local control is used to avoid explicit long-range path planning. Previous work in locomotion uses a cellular automata framework with constant information and computation requirements [2], but requires a restricted initial configuration and complex rule sets. Our technique uses more explicit planning to direct a robot from any configuration to any location over obstacles.

II. Parallel Path Planning

Planning for locomotion is similar to reconfiguration planning, in that modules move through the system to change the shape of the robot and reach a goal region at a distant location. Here, instead of finding a set of global paths that are guaranteed to work in concert (the space of such sets of paths is extremely large [1]), we build independent paths and continuously replan them in an efficient way as the system evolves. We define the *Parallel Path Planning* problem as finding a path from each mobile module to a position in a goal region, here defined by a bounding box. Borrowing techniques from Reinforcement Learning [5], we formulate this problem as a *Markov Decision Problem* (MDP) and solve it using dynamic programming (DP). The resulting policy yields a path from all open positions in the current configuration to a position in the goal region. Because the DP updates are executed in parallel, the policy converges in sublinear time in the size of the robot. As modules move, the underlying MDP also changes and we update the policy.

In our MDP formulation, a state s is a module face, or equivalently the adjacent lattice position, and an action a is a module actuation in any of the four cardinal directions with respect to s. The transition function maps each (s, a) pair to the resulting lattice position s'. State-action pairs that result in collision with an obstacle or another module transition to a state with a large negative reward. Otherwise, a reward of -1 is given for each action that does not transition into the goal region. For s' in the goal region, the reward is 0 plus a small negative value determined by the height of the lattice position above the ground, decreasing from 0 towards -1. This reward function results in modules moving first into the goal region and then towards the ground as far as possible.

Since the value function is defined over module faces, it is natural to store it within the modules and propagate value updates in a parallel distributed fashion. Updates are triggered by a change in the goal region as well as by module movement. Values propagate back from the goal, exploiting best case DP behavior, and the MDP converges quickly. A module moves toward the goal by querying its neighbors at each step and choosing an action with maximal value. This mechanism does not prevent disconnection or collisons with other moving modules, however. We describe a powerful method for local



(a) (b) (c) (d)

Fig. 1. Simulation of a thousand-module robot. Initial shape is shown in (a). After the first parallel actuation, the resulting configuration is (b). This illustrates a representative number of parallel actuations achieved by the algorithm. The goal region is shown filled in (c), along with the next position of the bounding box, which is nearly filled in (d). Due to our goal ordering heuristic, the goal region is filled from the bottom up.

motion control that solves these problems in the next section.

III. Parallel Actuation

When moving a single module in a fixed configuration, disconnection can be avoided with simple graph analysis: articulation points in the connectivity graph of the modules correspond to non-mobile modules (those which are not safe to move) and can be easily detected. However, this test takes linear time, and finding a set of mobile modules that can safely move in parallel is significantly more difficult. Here we describe a method that can be executed in a parallel distributed manner, prevents disconnection and collision, and allows modules to follow the specified paths.

If we remove a module from the connectivity graph and the graph remains connected, then the removed module is mobile, and its neighbors must be connected by a path not passing through it. We can therefore identify a module as mobile by using a local search that attempts to find a path connecting all neighbors. This search begins at each neighbor, using depth-limited depth-first search implemented with message passing.

The modules along the search path are locked to preserve connectivity. We must also lock the (one-step) destination position to prevent collision with another moving module. This is implemented using message passing to simulate a test-and-set operation. After actuation, the locked modules are free to attempt their own motions. All modules execute this process in parallel, allowing parallel actuation. This is a conservative test in that the depth limit means we do not find *all* mobile modules, but we expect to find many mobile modules in dense configurations. Also, the depth limit gives a tight constant bound on the time it takes to do mobility checking. In the case where we find no mobile modules due to a sparse configuration (e.g. a large ring of modules), we can increase search depth as in iterative deepening search.

Modules continuously attempt to move, guided by the value function. Motion control and path planning are executed until the robot achieves the goal configuration. Then, the bounding box is shifted either by a human operator or by some higher level process and locomotion continues.

Our goal is to implement this algorithm in a system with one million modules. We have implemented our approach in simulation using the SRSim simulator [3]. To experiment with large robots simulated on a serial computer, we used a centralized implementation. Fig. 1 illustrates locomotion with a thousand-module robot. Experimental data confirms the running time grows with $\sqrt[3]{n}$ for cubic robots with n modules.

IV. Discussion

Since our approach comes from a well-studied area, we can borrow much of that analysis to prove properties of our algorithm. For instance, the convergence properties of MDPs are generally well understood. In addition, the dynamics of any SR system are important, and we hope to show that the reward structure enforces good dynamic structure. We can also show that the rewards will allow the modules to densely fill the goal area (i.e. not leave any holes), since any hole will immediately be filled by the module(s) above it.

The development of this algorithm has pointed out interesting issues with large, dense systems relative to smaller, sparse ones. For example, the mobile module detection is ideally suited to dense configurations, and may work reasonably for regular structures like scaffolds, but gracefully degrades in sparse structures. Similarly, the DP propagation takes time proportional to the diameter of the shape, which can be anywhere from $\sqrt[3]{n}$ to n, depending on density. We believe it is critical for any algorithm for SR robots to consider its configuration dependence, that is, under what circumstances it performs well, and in general to not only shape the robot to the task, but choose an algorithm appropriate to the particular configurations and vice versa.

Acknowledgment

National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

[1] A. Abrams and R. Ghrist. State complexes for metamorphic robot systems. *Intl. J. of Robotics Research*, 23(7-8):809–24, 2004.

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized locomotion control for lattice-based self-reconfigurable robots. *Int'l Journal of Robotics Research*, 23(9):919–38, 2004.

[3] R. Fitch. *Heterogeneous Self-Reconfiguring Robotics*. PhD thesis, Dartmouth College, 2004.

[4] K. Stoy and R. Nagpal. Self-reconfiguration using directed growth. In *Distributed Autonomous Robotic Systems (DARS'04)*, 2004.

[5] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[6] M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.