Controlled Module Density Helps Reconfiguration Planning

An Nguyen, *SCCM Program, Stanford University, Stanford, CA 94305*
Leonidas J. Guibas, *Department of Computer Science, Stanford University, Stanford, CA 94305*
Mark Yim, *Xerox Palo Alto Research Center, Palo Alto, CA 94304*

In modular reconfigurable systems, individual modules are capable of limited motion due to blocking and connectivity constraints, yet the entire system has a large number of degrees of freedom. The combination of these two facts makes motion planning for such systems exceptionally challenging. In this paper we present two results that shed some light on this problem. First we show that, for a robotic system consisting of hexagonal 2D modules, the absence of a single excluded configuration is sufficient to guarantee the feasibility of the motion planning problem (for any two connected configurations with the same number of modules). We also provide an analysis of the number of steps in which the reconfiguration can be accomplished. Second, we argue that skeletal metamodules, which are scaffolding-like structures in 2D built out of normal modules, offer an interesting alternative. General shapes can be built out of these metamodules and, unlike the case for shapes built directly out of modules, a metamodule can collapse and pass through the interior of its neighboring metamodules, thus eliminating all blocking constraints. This tunneling capability makes the motion planning problem easier and allows faster reconfiguration as well, by providing a higher bandwidth conduit, the interior of the shape, through which the modules can flow. The conclusion of our work is that it is worthwhile to study subclasses of shapes that (1) approximate closely arbitrary shapes, while also (2) simplifying significantly the motion planning problem.

1 Introduction

A modular reconfigurable robot, sometimes known as a *metamorphic robot* consists of many identical modules, each with limited ability to connect, disconnect, and move around its neighbors. Originally proposed by Yim [Yi93], Chirikjian [Ch94], and Murata [Mu94], these robots have since been further studied by these and several other authors [Yi94, Yi97, Ca99, Ch96, Ko98, Ko98, Mu98, Pa96, Pa97, Ru99, Zh99]. A modular reconfigurable robot offers many advantages over conventional robots [Mu94]. Such a robot has high fault tolerance, since each module can be replaced by any other module in the robot; the homogeneity of the modules allows low cost mass production; finally, the robot can assume a wide variety of conformations or shapes, and thus can be easily adapted to performing numerous tasks.

A reconfigurable robot changes its shape by moving one or more of its modules locally. The motion of each individual module in the configuration must satisfy certain constraints, as illustrated in Figure 1. In general, there are two types of constraints: the connectivity constraint requires all modules to stay connected at all times to a fixed base (for power distribution, inter-module communication, etc), and the blocking constraint requires a module not to collide with other modules during its motion. A module may also need to support itself on its neighboring modules in order to move.

Though each module can have very limited motion, or none at all, the large number of modules present means that the system has many degrees of freedom — in general, roughly proportional to the number of modules present. The homogeneity of the modules further complicates the reconfiguration problem, as it is not clear which module in the original configuration should go to which location in the final configuration. Thus motion planning for modular reconfigurable robots is a challenging combinatorial task; in fact, under several scenarios, the local blocking constraints can make the task infeasible.

In this paper we propose to address this problem by focusing on selected subsets of the allowed module configurations. By imposing certain constraints that limit how locally packed the modules can be, we demonstrate that the reconfiguration problem can become always

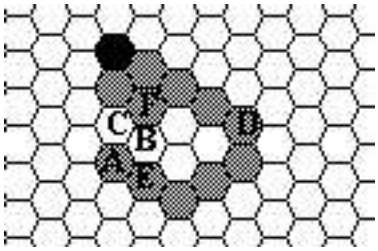

Figure 1: *This figure shows a reconfigurable robot composed of rigid hexagonal modules. Each module can move to a nearby empty space by rotating itself around a corner it shares with some other supporting module. The allowable motions of a module are restricted. The base, always shown in black in this and all subsequent figures, can never move. The module at D cannot move anywhere since such motion would leave the configuration disconnected. The module at A cannot move to B around supporting module E because such motion is blocked by F. The module at A cannot move to C either, because there is no module supporting the move.*

feasible, or a least significantly easier. We give two results, both for modular reconfigurable robots whose modules are rigid and correspond to a regular tiling of space. First, we show that for robots composed of 2D rigid hexagonal modules on a planar lattice, the reconfiguration problem is always solvable if both the initial and final configurations do not contain the excluded pattern (occupied, empty, occupied) parallel to any of the three axis of the lattice. We also analyze the complexity of our morphing algorithm in this case. Second, we propose the use of a *metamodule* structure for a 2D hexagonal lattice. A metamodule is effectively a scaled copy of the original module, built out of the original modules but with free space in its interior. Structures built entirely out of these metamodules are easier to reconfigure, as a metamodule has the ability to fold up and tunnel through any of its neighbors, thus removing the troublesome blocking constraints for the original modules.

In both cases there is still a very rich space of allowed robot configurations. Our intuition is that the shapes the robot needs to assume so as to perform various functions can be constructed or, at least, well-approximated by configurations satisfying the stated packing constraints. Thus by restricting ourselves to such less dense structures we gain simplicity in motion planning without any significant sacrifice in functionality.

2 Related Work

Here we survey briefly earlier related work on reconfigurable robots. Murata et al. built a "self assembling machine" [Mu94]. This machine is a robot constructed out of 2D fractum modules. The reconfiguration of such a robot is done with random local motions to improve the 'fitness' of a configuration with respect to the final configuration. Due to the randomness of the planning, the convergence is slow, and not practical for large problems [Mu94]. Pamecha et al. [Pa96] built a robot from regular hexagonal modules. The hexagonal module is designed so that it has no blocking constraints. They, too, solved the reconfiguration problem using simulated annealing to improve the matching distance to the final configuration. Upper and lower bounds on the optimal number of moves required were also given [Pa97, ChP]. Murata and Kotay et al. independently built 3D modular robots using 3D units [Mu98] and 3D molecules [Ko98] and discussed motion planning for their robots. Rus and Vona [Ru99] proposed building a system out of cubic compressible modules called *Crystalline*. They gave a *melt and grow* algorithm to do the motion planning between any two configurations. Another way to build 3D modular robots was proposed by Yim et al. [Yi97], using a 3D rhombic dodecahedron as the key element. Zhang et al. [Zh99] gave several heuristic algorithms for distributed parallel motion planning of such an robot.

The extant literature has not addressed the issue of the feasibility of the motion planning problem. Several of the proposed robots including Murata's fractum modules [Mu98] and Yim's rhombic dodecahedron [Yi97] (with 3, 5, or 7 sided constraints), admit of infeasible reconfiguration problems. Though in some cases the feasibility can be restored by building flexible modules to overcome the neighbor blocking constraints, this adds mechanical and control complexity, and therefore cost to the system.

3 The Reconfiguration Problem

A fundamental problem for a robot of this type is to plan how to accomplish its reconfiguration. Given initial and final configurations, we would like to compute the motions of the individual modules in the configuration that transform the initial configuration into

the final one. In theory, reconfiguration can be done by searching the graph of all possible configurations, with edges between two configurations representing the reachability from one to other in a single step of some module in the configuration. This searching approach is not practical since the number of possible configurations grows exponentially with the number of modules in the configuration. The approach is thus intractable, even for robots with modest numbers of modules.

Note that the optimal number of steps to transform one configuration to another is a metric defined on the space of all configurations. We refer to this number as the distance in the configuration space. Any optimal planner can be used to compute this distance, and conversely, any way to compute this distance function yields a way to do the motion planning. The computation of the optimal metric function would probably be as hard as the motion planning itself. Current approaches for this problem define heuristic approximations to this function that can be computed quickly and potentially decrease the distance to the final configuration.

4 Difficulty with Motion Planning

It is unfortunate that in the presence of motion constraints, two configurations that are very close in most intuitive distance notions may not be at all close to each other in the configuration space. For example, consider two solid disk-like shapes packed with modules except for a single-module hole near their centers, but in a slightly different location in the two disks, see Figure 2. These shapes are approximately the same by virtually all geometric and topological measures. Yet, the number of motion steps required to transform one disk to another is large, since any motion plan must unpack the disk before being able to move some module into the empty hole.

Before we discuss further difficulties, note that a robot may be *immobile*, with none of its modules able to move without violating their motion constraints. Such immobile configurations are typically the result of the blocking constraints (in practice the connectivity constraint always exists, but it cannot cause immobilization by itself.) An example of immobilization for a robot composed of rigid hexagons is shown in Figure 3. Robots made of fracta by Murata et al. or rhombic dodecahedron by Yim et al. with 7-sided blocking

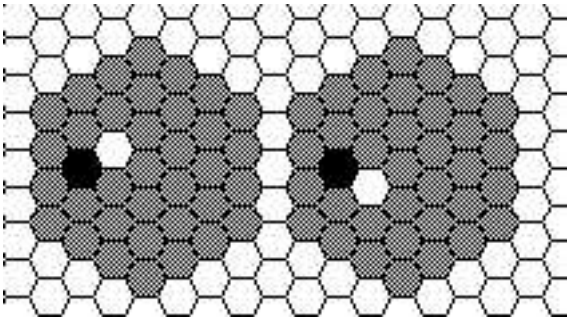

Figure 2: *Configurations that are approximately the same in geometric and topological sense may be far apart in the configuration space.*

constraints are known to have immobile configurations. Robots made of rhombic dodecahedra with 3- and 5-sided constraints can also be immobile. The construction of such immobile configurations is similar to that of the configuration shown in Figure 3.

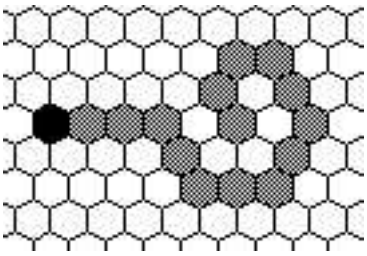

Figure 3: *Here is an example of an immobile configuration. The black module at one end of the chain is the fixed base, and the module at the other end cannot move because of blocking constraint. All other modules cannot move because of connectivity constraint.*

If the initial or final configuration is immobile, it will be impossible to find a motion plan between the two configurations. In general, the existence of an immobile configuration is likely to make the reconfiguration problem harder, even when it is feasible. There exist immobile configurations where a slight change in a configuration can make it mobile again — thus the distance in the configuration space between two configurations can be very sensitive to local perturbations. It would be hard for a heuristic measure to capture this delicate distinction. Another problem with immobile configurations is that there may exist an *immobile branch* even in a mobile configuration. Any mo-

tion planning involving immobile branches would require construction or destruction of the branches, and it would be hard to define a heuristic that captures that.

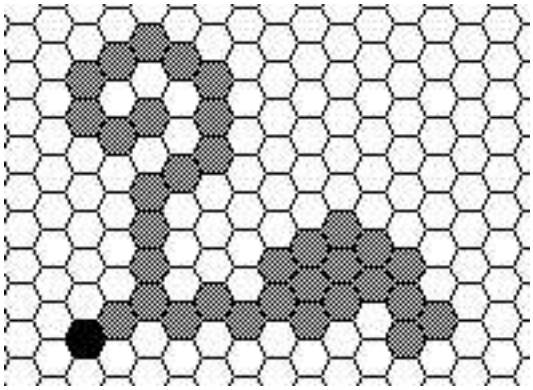

Figure 4: *An example of an immobile branch in a mobile configuration.*

Given the difficulty for solving the motion planning problem in general, it is desirable to restrict ourselves to a simpler problem. As already mentioned, we propose to restrict ourselves to a subset of all possible configurations, large enough to adequately approximate an arbitrary configuration, yet restricted enough to make the motion planning more tractable.

5 Reconfiguration for Planar Hexagonal Modules

We consider the case of a reconfigurable robot composed of rigid hexagonal modules in the plane, as illustrated in Figure 1. We assume that all configurations of interest must be connected, and that a particular module, called the base, is immobile. The base can be used for transmitting power and communication into the system. We will refer to a possible location for a module as a grid cell, or simply a *grid*. The only motion allowed for a module is the rigid rotation around a vertex it shares with some other module in the configuration, subject to the condition that such rotation is not blocked by any module nor leaves the configuration disconnected.

We call a robot configuration *admissible* if it is connected and there is no empty grid cell directly in between two occupied grid cells in the configuration; see Figure 5. Admissible configurations have a very nice property, as shown in the following proposition.

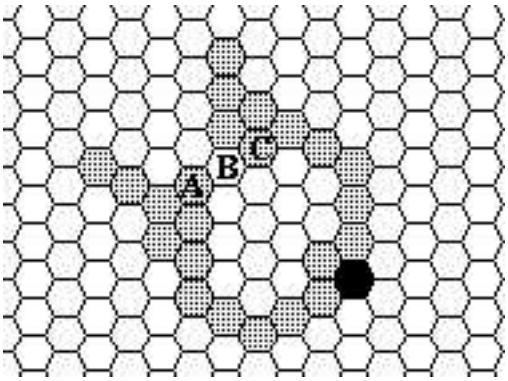

Figure 5: *The above configuration is inadmissible because the empty grid B is between two occupied grids A and C. If a module is added to the configuration at B, the configuration becomes admissible. The configuration also becomes admissible if the module at A or C is removed.*

Proposition 1 *Any admissible configuration can be transformed into a straight line configuration.*

As an immediate corollary of proposition 1:

Corollary 2 *Any admissible configuration can be transformed into any other admissible configuration with the same number of modules.*

Proof of proposition 1:

We construct a transformation from an admissible configuration to a straight configuration in two phases. First, we transform the admissible configuration to an admissible chain configuration (to be defined below), and then convert that chain to a straight line configuration.

Note that the complement of an admissible configuration in the plane consists of one or more connected components, exactly one of which is infinite. We call this infinite component the *outer free space* (with respect to the configuration). The module boundary line segments separating the outer free space and the configuration are called the *outer boundary*, and the modules in the configuration that border the outer free space are said to be on the outer boundary.

For ease of discussion, we assume that the hexagon lattice is oriented as in the previous figures. We use lat-

tice coordinates with the x-axis in the minus 30 degree direction, and the y-axis in the vertical direction.

In the first phase, we use a greedy algorithm. Let us choose an extreme module in the configuration, say choose among all modules with maximal x coordinate the one with maximal y coordinate. Clearly, this module is on the outer boundary of the configuration. We grow a 'tail' by repeatedly locating available modules that can be moved towards the extreme module we just selected, moving them there and then appending them to the end of this growing tail. This process ends when there is no module available for extending the tail.

The search for a module to add to the tail is done greedily. Starting from the extreme position, we search among all modules on the outer boundary of the configuration, say in the counter-clockwise direction, for the first module that can move clockwise without violating any motion constraints. We then let that module roll into the free space, along the outer boundary toward the extreme position, and then along the existing tail until it reaches the tip. The feasibility of such a motion is guaranteed with the help of the following lemma:

Lemma 3 *A module moving along the outer boundary of an admissible configuration in counter-clockwise (clockwise) direction can only be blocked at a position at which its blocker module is capable of moving in the that same direction.*

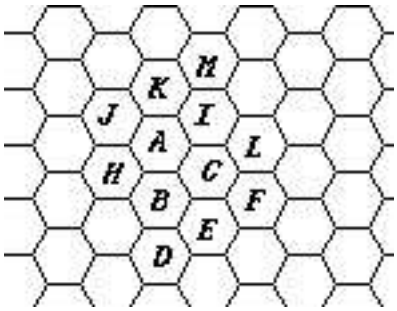

Figure 6: *This figure is used in the proof of lemma 3. A is a grid on the outer boundary of a configuration, and B and C are on the outer free space. If a module, moving in the counter-clockwise direction along the outer boundary of the configuration, can reach B but cannot reach C, then the lemma asserts that the module at A can move counter-clockwise to either K, or I. Other grid labels in the figure are used in the proof of the lemma.*

Proof of lemma 3:

We only consider the case when the direction is counter-clockwise. The clockwise case is similar. See Figure 6.

Say that in the original configuration, before the moving module moves to B, grid A is occupied, and grids B and C are empty. The admissibility of the configuration implies that grids D and F are empty. Thus the only reason that a moving module cannot move forward from B to C is that E is occupied. The admissibility again dictates that H and I are empty. There are now two possibilities:

- If K is occupied, then L is empty, and thus the module at A can roll counter-clockwise over K to I.
- If K is empty, then there must be a module at J so that A is not disconnected from the rest of the configuration. Grid M must then be empty, and the module at A can roll counter-clockwise over module at J to K.

In both situations, the module at A can move counter-clockwise, and thus lemma 3 is proved. $\diamond$

Since we choose the first module that can move counter-clockwise, the lemma guarantees that such a module can move counter-clockwise without ever being blocked toward the extreme module. The module can then move to the tip of the constructed tail. We need to make sure, however, that the configuration is admissible at the end of each move.

Note that the removal of a module from an admissible configuration always leaves it admissible as long as it is still connected. We only need to ensure that when we add the module to the tip of the tail, the configuration is still admissible. Since the tail points right up from a module at an extreme position, most modules in the tail are too far to affect the admissibility of the rest of the configuration. We need to be cautious only when building the first few modules in the tail.

Let the coordinates of the extreme module be (x_0, y_0). It is easy to verify that the following scheme for building the tail will keep the configuration admissible, see Figure 7:

- if $(x_0 - 1, y_0 + 1)$ is empty, the tail can be build at $(x_0 + 1, y_0 + 1), (x_0 + 2, y_0 + 2), (x_0 + 3, y_0 + 3), \ldots$;

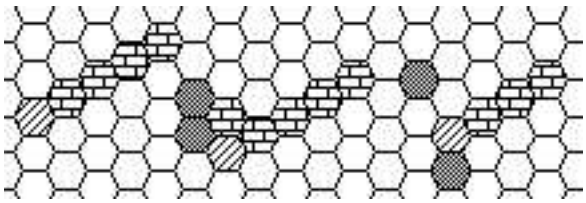

Figure 7: *There are three different types of tails. In each sub-figure, the grid shaded with slanted lines is the extreme module, gray grid(s) are other occupied grids, and grids shaded with bricks are grids in the tail to be constructed.*

- if (x_0-1, y_0+1) is occupied, and (x_0-1, y_0) is also occupied, the tail can be build at $(x_0, y_0+1), (x_0+1, y_0+1), (x_0+2, y_0+2), (x_0+3, y_0+3), \ldots$;
- if (x_0-1, y_0+1) is occupied, but (x_0-1, y_0) is empty, then (x_0-1, y_0-1) is also empty, and in order for (x_0, y_0) to be connected, (x_0, y_0-1) is occupied. The tail can be build at $(x_0+1, y_0), (x_0+2, y_0+1), (x_0+3, y_0+2), \ldots$.

The first phase eventually ends. At that time, no piece in the configuration is free to move except the module at the end of the constructed tail.

Lemma 4 *The configuration at the end of phase 1 is a chain with one end at the end of the newly constructed tail and with the other end at the fixed base.*

Before we prove lemma 4, let us prove:

Lemma 5 *Any admissible configuration has a module on the outer boundary that can move without any motion constraint.*

Proof of lemma 5:

We call a module, say X, in a configuration a *connector* if the removal of X from the configuration makes the configuration disconnected. For a connector X, we define the *branch* at X as the subconfiguration consisting of X and the component that becomes disconnected from the base upon the removal of X.

Consider an extreme module in the configuration that is different from the fixed base. This module is on the outer boundary, and its motion (clockwise or counter-clockwise) into the outer free space doesn't violate any blocking constraint. If this module is not a connector, it is the desired module.

If it is a connector, consider the branch at this connector as a configuration, with the connector as the fixed base. By an inductive argument, we can assume that there is a module capable of moving without constraints. This module is on the outer boundary of the new configuration and thus is on the outer boundary of the original configuration. $\diamond$

Proof of lemma 4:

In this proof, we use the term *loop* for a circular list of distinct modules, each being a neighbor to the modules located right before and after it in the circular list. See Figure 8.

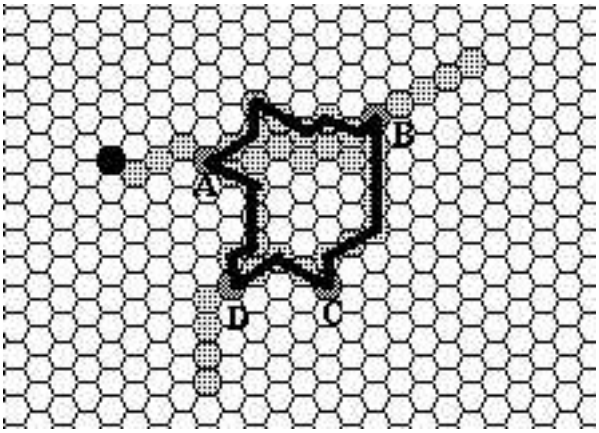

Figure 8: *The black loop represents the largest loop in the configuration. A is the loop connector for the fixed base. B is the loop connector for the the tail. C is an extreme module on the loop that has no motion constraints for its motion to the outer free space. D is an extreme module on the loop, but is a connector. A and B happen to be extreme modules of the loop.*

Consider the configuration obtained after phase 1. Assume, for the sake of contradiction, that this configuration is not a chain, so there is a loop in the configuration. Among all the loops, choose the one enclosing the largest number of grids. Call this loop $\mathcal{L}$. ¿From any module, say X, outside of loop $\mathcal{L}$, there is one or more paths connecting the module to the loop, all of which must share a common module Y contacting the loop (otherwise, the loop can be augmented to enclose more grids inside.) We call Y the *loop connector* for X. The removal of Y would make X disconnected from $\mathcal{L}$, and thus make the configuration disconnected. Thus, a loop connector is a connector as previously defined.

Consider an extreme module M in loop $\mathcal{L}$. Without loss of generality, we can assume that M is neither the loop connector for the tip of the tail constructed, nor the loop connector for the fixed base (if the loop connectors for those modules are defined). We can make this assumption, since a loop has more than two extreme modules, and thus we can always choose an extreme module different from the above two connectors. Note that with our choice of $\mathcal{L}$, module M must be on the outer boundary of the configuration.

There are two possible cases:

- If M has no neighboring module outside of the loop then, due to its extremality, M can move into the outer free space without any motion constraints.
- If M has as a neighbor module N outside of the loop, then M is a loop connector for N. Consider the branch at M as a configuration with fixed base M. By lemma 5, there is a module on the boundary of this new configuration that can move into the outer free space of the new configuration. It is clear that this module can move into the outer free space of the original configuration.

Thus, in both cases, we have a contradiction: there is a module capable of moving to the tail built at the end of phase 1. This proves that the assumption we made is false, and thus lemma 4 holds. In other words, the greedy algorithm in phase 1 transforms any admissible configuration into an admissible chain. $\diamond$

In phase 2, we show that it is possible to transform an admissible chain configuration into a straight line configuration. The chain configuration may be winding around the fixed base, and a simple way to make it straight is to unwind it. Starting with the extreme module we used in phase 1, we repeatedly choose a different extreme module on the chain, then build a tail at this location. The tail will have more and more modules after each iteration, since the previous extreme module and all the modules in the previous tail are in the new tail. Eventually, the fixed base becomes one of the extreme modules, and the tail contains all modules other than the fixed base.

If the original configuration has n modules, the first phase would take $\mathcal{O}(n^2)$ steps since each module moves at most $\mathcal{O}(n)$ steps. In the second phase, the cost for each tail relocation is $\mathcal{O}(n^2)$. The number of tail relocation can be bounded by the following lemma:

Lemma 6 *The number of tail relocation steps in phase 2 is* $\mathcal{O}(n^{1/2})$

Proof: At any moment, consider the configuration excluding the tail. Assume that this configuration has m, $(m \leq n)$ modules, then its diameter d (measured in terms of grid modules) is at least $\Theta(m^{1/2})$. Let P and Q be two modules with distance d. It is easy to see that in two tail relocations, by relocating the tail to P then to Q (or to Q, then to P), the tail has at least d more modules; the remaining configuration has $\Theta(m^{1/2})$ less modules. Straightforward analysis shows that the number of tail relocations must be at most $\mathcal{O}(n^{1/2})$. $\diamond$

¿From lemma 6, the number of steps in the second phase is $\mathcal{O}(n^{5/2})$. This worse case can be achieved for a winding chain of n modules around the fixed base.

This completes our proof of proposition 1. $\diamond$

6 Robots Composed of Metamodules

In this section, we explore a different approach to reconfiguration planning. Instead of restricting ourselves to admissible configurations, we consider arbitrary configurations that can be built out of a different building block, a *skeletal metamodule* or in short a *metamodule*. A skeletal metamodule is a collection of modules in a scaffolding-like structure, designed with the purpose of relaxing the basic module blocking constraints. The metamodules themselves pack in a lattice structure; the ones we consider have an overall shape that is a scaled-up copy of the basic module shape.

Figure 9 shows a robot of rigid hexagons, built from skeletal metamodules. The metamodule in Figure 9 still has a hexagonal shape, but it is an enlarged copy of the original module shape and has an empty interior.

A metamodule can decompose itself into modules, which can move to another nearby location, then reassemble themselves back into a metamodule. This can happen on the surface of the structure, just like with regular modules, but now with no blocking constraints. More interestingly, in a configuration of metamodules that are sufficiently large, a metamodule can move to a neighboring empty position *through the interior* of

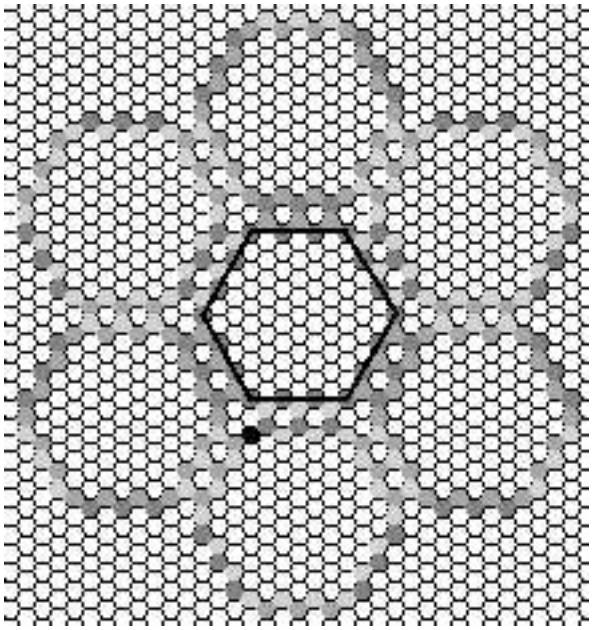

Figure 9: *This figure illustrates a configuration of several hexagonal metamodules. Each metamodule consists of 36 hexagonal modules, yet collectively behaves like a single hexagonal module. Unlike a regular hexagonal module, a metamodule have no blocking constraints, and can move by tunneling through other metamodules.*

a supporting metamodule neighboring both the metamodule and the nearby empty location. This can be achieved by moving out of the way certain modules of the supporting metamodule, thus creating a gate to its interior, and then allowing modules of the moving metamodule to go inside. The supporting metamodule can then close the gate and open another gate toward the empty location, letting the modules inside it move out and to assemble back to a metamodule again; see Figure 10. This neighbor to neighbor motion is again without any blocking constraint. In the same way a metamodule can actually tunnel through the interior of the structure and then reassemble itself when it reaches the boundary on the other side. Thus the metamodule can be thought of as a module of a new metamorphic robot, but now this module has no blocking constraints and can pass through the interior of other modules.

As the above example shows, metamodules can be built out of regular reconfigurable robot modules and then they themselves can be treated as building blocks

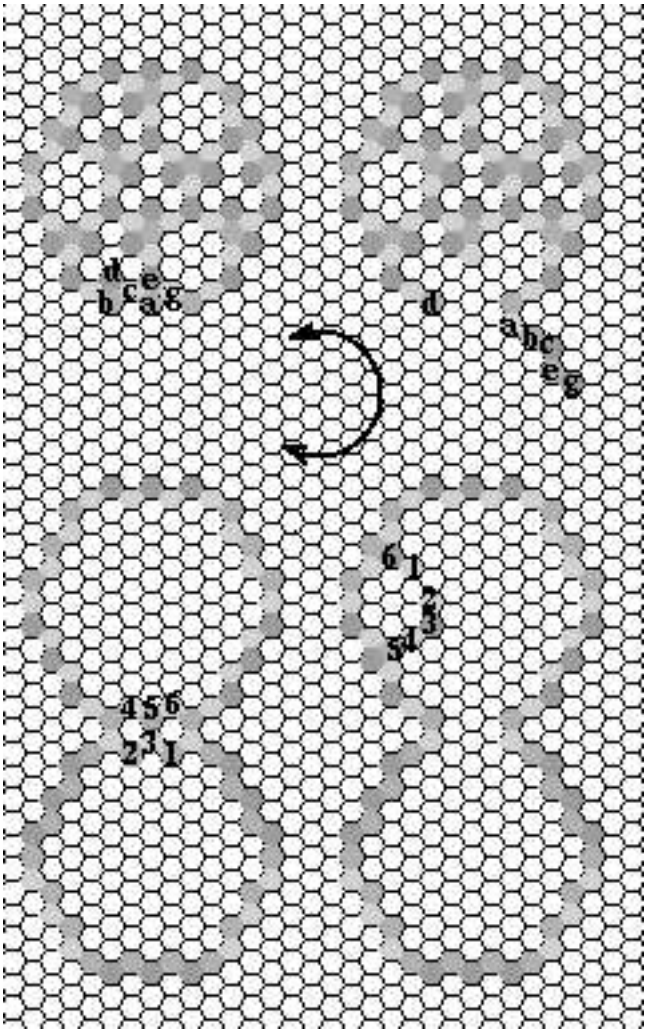

Figure 10: *This figure illustrates the process in which a metamodule, collapsed inside another metamodule, comes out to form a metamodule again. On the top we see a metamodule with another collapsed metamodule inside it. The grids labeled by letters form the gate that opens to let the collapsed metamodule out. On the bottom we see the second metamodule almost fully assembled. Its final modules are in the grids labeled by numbers — after they come out the transformation is complete.*

for a reconfigurable robot. Any sufficiently large shape built out of basic modules can be well approximated by metamodules, or an exact scaled version of the original shape can be constructed directly out of metamodules (if scale is not important) by using a metamodule in-

stead of a module as the basic unit. Thus metamodules configurations can be used in lieu of basic module configurations, with little loss in modeling power.

Metamodules have several advantages over standard modules. If we have a heuristic algorithm to do motion planning for a reconfigurable robot of hexagon modules, that algorithm can be used directly for a robot of hexagonal metamodules. Furthermore, heuristic measures for the distance between two configurations will most likely result in better quality plans in the metamodule case, as the blocking constraints have been eliminated. For example, the heuristics proposed by Chirikjian and Pamecha to do motion planning [Pa97], and the bound on the number of steps [ChP] are valid for metamodule configurations.

Metamodules also have the additional tunneling capability. Besides moving on the boundary of the configuration, like regular modules, metamodules have the flexibility to go through the interior of other modules and thus they can make shortcuts through the configuration. If we let many modules move at the same time, the interior is now available to provide a much higher bandwidth channel for the motion than is possible on the surface alone. This tunneling capability of metamodules can be also used to create compressible modules, such as those proposed by Rus and Vona [Ru99].

6.1 Restricted Configurations of Metamodules

In this section, we propose a new way to do parallel motion planning for a restricted class of metamodule configurations. This combines the idea of metamodules and that of restricted configurations. We allow modules to move concurrently to speed up reconfiguration. The approach is motivated by the work of Rus and Vona [Ru99].

Suppose that we have two layers of metamodules as shown in Figure 11. If we keep the metamodules on the bottom layer fixed, we can collapse all metamodules on the top layer, move the modules in parallel to the right, then reconstruct the metamodules on top. This can be thought of as a shifting of the top layer over the bottom layer by one module, in one time step. The overall effect of the operation is to move the leftmost metamodule of the top layer all the way to the right in one time step, independent of the number of metamodules on the layers.

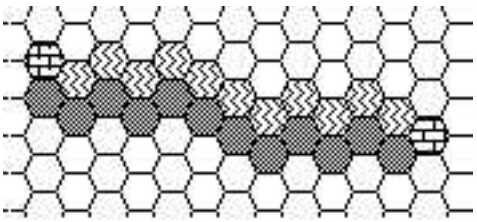

Figure 11: *The top layer of metamodules can move to the right over the bottom layers in one parallel time step. This is equivalent to moving the metamodule on the top layer from the extreme left position to the extreme right position in one parallel step.*

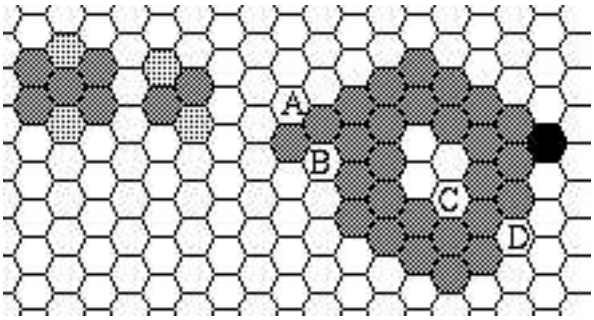

Figure 12: *The two local combinations of modules on the left are prohibited in a* fat *configuration. The dark and light gray grids represent occupied and unoccupied grids respectively. The configuration on the right is not* fat*, because the fat condition is violated in two different places. If two modules are added to the configuration at B and C, the configuration will become* fat*.*

We will use the above operation as a basic step for our motion planning. It is preferable that the configuration we deal with is not too skinny. We restrict ourselves to a new class of *fat configurations* in which two local combinations of metamodules, as explained in Figure 12, are prohibited. This restriction essentially requires that a *fat* configuration has locally, width two or more; again, it should be clear that this restriction has little affect on the class of shapes that can be approximated. We will use the term *nonfat* for configurations that are not *fat*.

Fat configurations have a nice property, as shown in the following proposition:

Proposition 7 *Any* fat *configuration of n metamodules can be transformed into a straight line* fat *config-*

uration in $\mathcal{O}(n)$ parallel steps.

As an immediate corollary of proposition 7:

Corollary 8 *Any* fat *configuration of n metamodules can be transformed into any other* fat *configuration with the same number of modules in $\mathcal{O}(n)$ parallel steps.*

Proof of proposition 7:

Without loss of generality, it suffices to show that we can transform a *fat* configuration to a *fat* straight line configuration with a rounded end (shown shaded with slanted lines in Figure 13.) Furthermore, we can assume that the six metamodules surrounding the base metamodule are already in the original configuration. This is so we can preprocess the original configuration and postprocess the final configuration to the desired form in $\mathcal{O}(n)$ time.

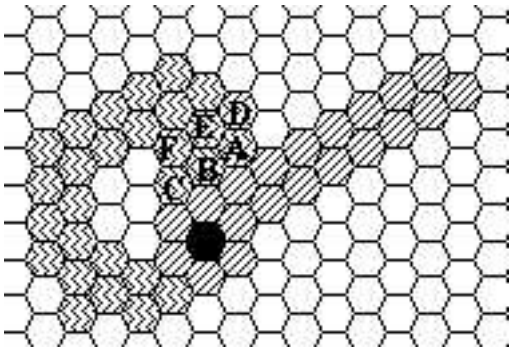

Figure 13: *The metamodules in a* fat *configuration can move to generate the rounded straight line to the right. The algorithm applied on this configuration first moves A, then moves both B and C (and breaks the only loop in the configuration), then moves D, F, and E to the tail.*

We again will use a greedy algorithm. Starting from the tip of the tail constructed so far, we search in the counter-clockwise direction along the boundary of the configuration to find the first metamodule not already in the tail satisfying either of the following two conditions:

- The removal of the metamodule leaves the configuration connected and *fat*. In this case, we use the parallel shearing operation to shift an entire layer of the configuration on the boundary, with the net effect of moving the selected metamodule in $\mathcal{O}(1)$ steps to the tip of the tail. If the layer on the boundary includes metamodules in both the upper and lower layers of the tail, we do the shearing in two steps, the first shearing uses only one of the two layers of the tail, and the second shearing uses the remaining layer.
- The removal of the metamodule leaves the configuration connected but *nonfat*, and further removal of one of its neighboring metamodule leaves the configuration still connected and *fat* again. In this case, we move both modules to the tip of the tail in two consecutive time steps.

Note that the fixed base should not be in any layer being shifted, and this is guaranteed since there are modules surrounding the fixed base at all times. Also, although the configuration at the end of each step may not be *fat*, this only happens at the tip of the tail (when the tail first touches some other part of the configuration.) For the analysis, it is safe to assume that the configuration is always *fat*, since we can pretend that the tip of the tail is not yet there when we search for the next metamodule.

Before further analysis, let us give some intuition behind the two cases above. In the first case, the operation takes away a metamodule on the boundary, making the configuration thinner. Since the configuration is *fat*, the topology of the configuration does not change. In the second case, the configuration has some thin cross section in a loop. The removal of a module in that thin section makes the configuration *nonfat*, yet the additional metamodule removal eliminates the skinny cross section. The loop is disconnected, and the topology of the configuration changes, see Figure 13.

We claim that the above greedy algorithm moves all metamodules in the configuration to the tail. Let us assume for the sake of contradiction that there are some metamodules not in the tail when the greedy algorithm fails to find any metamodules to move. Consider the free space, the complement of the configuration. If there are two or more connected components of the free space, consider the smallest distance between the component contacting the tail and all remaining components. This distance is not one, since we assume that the configuration at the beginning of each search is always *fat*. If this distance is two, the second condition would be applicable to remove the two metamodules, merging the two components of the free space. If

the distance is three or more everywhere, clearly the first condition would have been used to remove further metamodules. Thus the free space is connected. In order for the first condition not to be applicable, the metamodules not in the tail are on one or more thin chains originating from the tail. This is a contradiction, since the metamodules at the tip of these chains would have been removed by the first condition.

The $\mathcal{O}(n)$ number of parallel steps follows trivially, since the remaining configuration has one less metamodule after every $\mathcal{O}(1)$ time steps. $\diamond$

6.2 Reconfiguration by Tunneling

If the metamodules are large enough to allow tunneling, then another method can be used for reconfiguring an arbitrary configuration into a line, in $\mathcal{O}(n)$ steps. The intuition here is that a metamodule at a boundary of a configuration can follow a tunneling path through a configuration emerging at a growing tail that is forming a line (and consequently between two arbitrary configurations). Every metamodule in that path can immediately follow this leader metamodule and emerge at the end of the tail. The appropriate metaphor here is of a turning a rubber glove inside out by pushing in at the fingers and having them come out the other end, one at a time, appended to each other in a straight line, see Figure 14.

The first step is to impose a tree structure on the configuration. This is done by labeling the metamodules with increasing numbers in a breadth first traversal sequence, starting with the metamodule where the tail will grow as number 0. The breadth first numbering can be achieved with simple incremental message passing.

The following algorithm is purely local, in that every metamodule runs the same program (a small finite state machine) and no communication is required between metamodules except for knowledge of their neighboring metamodules and their numbers (in the labeling).

There are four states that every metamodule can be in:

- S_1 — the metamodule is part of the original structure (not tunneling nor part of the tail)
- S_2 — the metamodule is tunneling towards the 0 metamodule.
- S_3 — the tunneling metamodule has reached the 0 metamodule and is now tunneling in the tail.
- S_4 — the metamodule has grown the tail and so has stopped tunneling.

There are three rules for transitioning between states:

- T_1 ($S_1 \mapsto S_2$): if the metamodule is or becomes a leaf metamodule, it starts to tunnel toward 0.
- T_2 ($S_2 \mapsto S_3$): if the metamodule has tunneled into the 0 metamodule, it starts to tunnel to the tip of the tail.
- T_3 ($S_3 \mapsto S_4$): if the metamodule has reached the end of the tail, it exits and forms a normal non-tunneling metamodule growing the tail.

Any metamodule that is a leaf will move up the tree by tunneling inside the branches towards 0. Moving up the tree is accomplished by examining the number of the metamodules neighboring the current metamodule that the tunneling metamodule is contained in and targeting the neighbor with the lowest number as the direction to tunnel. When a leaf tunnels up a branch of the tree, the former second-to-last metamodule becomes the last metamodule and thus a leaf, so it starts to move up.

There are three cases where a metamodule in state S_2 will not be able to tunnel into its neighbor: 1) when it is being tunneled itself, 2) when another metamodule is inside the target metamodule and has not exited because it is waiting for another metamodule to move, and 3) when a metamodule with a higher number wants to enter the same target metamodule. The last case is an arbitrary prioritization (or right of way) convention. Once a metamodule reaches the 0 metamodule, it attempts to form the tail by tunneling down the tail until it emerges at the end, thus growing the tail.

At every step, at least one metamodule is either moving up the tree, or growing the tail. In fact, every metamodule will either be a conduit for other metamodules to pass through or will itself be tunneling through another metamodule after $\mathcal{O}(n)$ steps. While there is always a convoy tunelling towards the root or growing the tail, some convoys may be waiting at branch nodes for other convoys to finish their tunneling. Clearly a

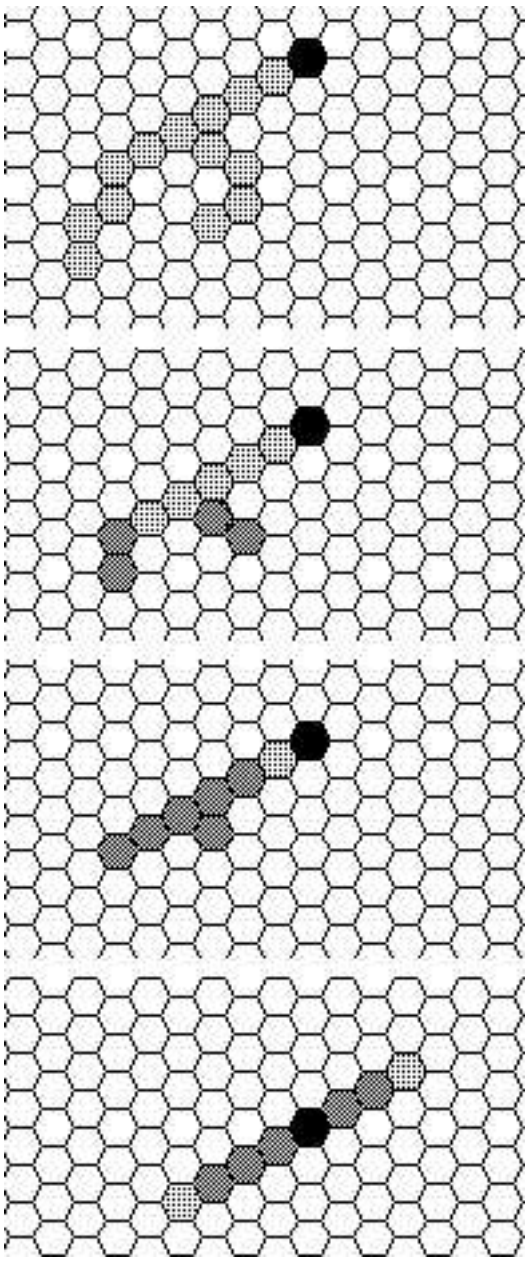

Figure 14: *Several snapshots of the tunneling process as it transforms the top configuration to a straight line. Metamodules tunnel toward the base, then extend out to form a straight line. In these figures, light gray grids contain regular metamodules, and dark gray grids contain metamodules with another metamodule tunneling inside them.*

metamodule only moves $\mathcal{O}(n)$ steps before it reaches its eventual goal position in the tail, as its motion follows a path in the tree and through the tail. What is more interesting is that a metamodule M also never waits a total of more than $\mathcal{O}(n)$ steps before reaching its destination. We can see this as follows. After the initial $\mathcal{O}(n)$ steps, metamodule M waits only when it is being tunneled through, or when it is part of a tunneling convoy that is waiting at a branch node up the tree for another convoy to pass through. In both cases another module, the *blocker* of M from a different subtree, is moving though the branch node for every time step that our module M has to wait. But once a metamodule has been the blocker of M, it can never block M again at a branch node, as it has now become an ancestor of M in the tree. Thus no node needs more than $\mathcal{O}(n)$ moving or waiting steps to reach its final destination.

Since we move from the leaves of the branch inward, connectivity is never broken. Furthermore this algorithm operates in a very simple and purely local fashion. All of this assumes that only one metamodule can be "tunneled" inside another metamodule at one time. Metamodules with larger interiors would have more "bandwidth" which would mean less traffic problems and fewer steps.

7 Conclusion

In this paper we have explored certain restrictions on the allowed structure of reconfigurable robots with the aim of making the motion planning problem easier, without unduly confining the set of shapes the robot can assume. Though all our results are in 2D, we believe that appropriate extensions to 3D are possible, and in some cases straightforward. For example, we can build 3D rhombic dodecahedron metamodules that consist of an edge skeleton of a scaled copy of the basic RD module. Because the parallelogram facets of these metamodules are actually solid only along their edges, other collapsed metamodules can enter, tunnel through, and exit a given metamodule easily, without any of the gate opening or closing operations we need in 2D. In 3D as well, the traffic management of convoys of modules tunneling through the robot interior seems easier than in 2D. Since the tunneling reconfiguration algorithm imposes a tree structure which is independent of the dimensionality of the physical modules, it will work for this 3D RD case as well.

Acknowledgments

Leonidas Guibas and An Nguyen were supported in part by National Science Foundation grants CCR–9623851 and IRI–9619625, as well as by US Army MURI grant DAAH04–96-1-0007. An Nguyen was also supported by a Stanford Graduate Fellowship.

References

[Ca99] Casal, A., Yim, M., "Self-Reconfiguration Planning for a Class of Modular Robots", SPIE Symposium on Intelligent Systems and Advanced Manufacturing, Sept. 1999.

[Ch94] Chirikjian, G., "Kinematics of a Metamorphic Robotic System," in *Prc. 1994 IEEE Int. Conf. Robotics and Automation, San Diego, CA.*

[ChP] Chirikjian, G., Pamecha, A., "Bounds for Self-Reconfiguration of Metamorphic Robots," *JSU Technical Report, RMS-9-9 5-1.*

[Ch96] Chirikjian, G. et al., "Evaluating Efficiency of Self-Reconfiguration in a Class of Modular Robots," in *Journal of Robotic Systems, June 1996.*

[Ko98] Kotay, K., et al., "The Self-reconfiguring Robotic Molecule: Design and Control Algorithms", Algorithmic Foundations of Robotics, 1998.

[Ko98] Kotay, K. et al., "The Self-reconfiguring Robotic Molecule," in *Proceedings of the 1998 IEEE International Conference on Robotics & Automation, May 1998.*

[Mu94] Murata, S. et al., "Self-Assembling Machine", in *Proceedings of the 1994 IEEE International Conference on Robotics & Automation, May 1994.*

[Mu98] Murata, S. et al., "A 3-D Self-Reconfigurable Structure", in *Proceedings of the 1998 IEEE International Conference on Robotics & Automation, May 1998.*

[Pa96] Pamecha, A. et al., "Design and Implementation of Metamorphic Robots" in *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, Aug 1996.*

[Pa97] Pamecha, A., Chirikjian, G., "Useful Metrics for Modular Robot Motion Planning," in *IEEE Transactions on Robotics and Automation, Vol. 13, No 4, August 1997.*

[Ru99] Rus, D., Vona, M., "Self-reconfiguration Planning with Compressible Unit Modules," in *Proceedings of the 1999 IEEE International Conference on Robotics & Automation, May 1999.*

[Yi93] Yim, M., "A Reconfigurable Modular Robot with Many Degrees of Locomotion," in *Proc. 1993 JSME Int. Conf. Advanced Mechatronics.*

[Yi94] Yim, M., "Locomotion With a Unit-Modular Reconfigurable Robot", Stanford University PhD Thesis, 1994.

[Yi97] Yim, M. et al., "Rhombic Dodecahedron Shape for Self-Assembling Robots," Xerox PARC, SPL TechReport P9710777, 1997.

[Zh99] Zhang, Y., et al., "Distributed Control for 3D Shape Metamorphosis," submitted *Autonomous Robots Journal, special issue on self-reconfigurable robots*, 1999.