An Efficient Algorithm for Self-Reconfiguration Planning in a Modular Robot

Tom Larkworthy[1] and Subramanian Ramamoorthy[1]

***Abstract*— An efficient planning algorithm for the hexagonal metamorphic self-reconfiguring system (SRS) is presented. Empirically, the algorithm achieves an time complexity of $O(n)$ averaged over random problem instances. The planning algorithm is capable of solving approximately 97% of planning tasks in the general state space of configurations containing less than 20,000 units.**

The state space is divided into two classes according to planning efficiency. The configurations belonging to the first class permit an Euler tour to be wrapped around the robotic aggregate. The existence of the Euler tour implies units are free to move around the perimeter of the SRS. Planning between configurations in this class can be performed in $O(n)$ using a specialized planning algorithm. The set of Euler tour configurations span a large volume of the general state space of the hexagonal SRS.

A second specialized planning algorithm plans from a general configuration to a nearby Euler tour configuration. While planning in the general configuration state space is computationally harder, the distance required to plan is short. Thus, the combination of both algorithms allows us to efficiently plan for a large proportion of possible reconfiguration tasks for the hexagonal metamorphic robot.

I. Introduction

We wish to build self-reconfiguring systems (SRSs) in order to overcome the limitations of monolithic robotic platforms. SRSs are comprised of numerous modular robotic subunits capable of actively altering their connectivity to their spatial neighbors. Through cooperation between subunits, the SRS aggregate is able to change morphology. The advantages of this approach include non-fixed workspace, flexible locomotion, economies of scale, scalable strength and scalable accuracy[18], [14]. However, in order for us to realize practical SRSs, two broad categories of technical issues need to be addressed, namely construction and control.

This work focuses on the issue of controlling the aggregate to reconfigure; determining a set of moves for the sub-units to execute in order to change from a current configuration into a desired configuration. Clearly the planning problem is dependent on the motion constraints of the particular robotic system. The algorithm developed here is for the hexagonal metamorphic robot. This abstraction of the problem constrains the SRS to reside on a discrete lattice. This is in contrast to other SRSs whose state contains continuous variables. While planning for a discrete SRS appears to be easier than for other types, as yet, there is no satisfactory solution that scales well with the number of units in the configuration.

Reconfiguration planning is a motion planning problem. While our specific motivation is to improve SRS technology, this particular motion planning problem is interesting to a broad community because of the combination of an exponential state space and non-trivial, yet exploitable, structure. Convincingly solving this problem may provide broader insights into the nature of difficult state spaces.

II. Previous Work

The original model of the hexagonal metamorphic robot was defined by Chirikjian[3]. Early planning work for this model used simulated annealing as a search technique, guided by a heuristic[16]. The plans formed by this methodology only permitted a single hexagonal unit to move per iteration of time (single-move planning), and the system was not scaled up to plan for configurations with more than 50 units.

Ghrist *et al.* [1], [8], [7] analyzed the problem using tools from topology and developed an alternative set of motion constrains for the hexagonal metamorphic system. The movement catalog was more constrained than Chirikjian's. Ghrist's definition excluded moves that could change the global topology of the system, such as introduction of enclosed space. These changes had non-trivial consequences on the reconfiguration space, and allowed Ghrist to show that the new reconfiguration space was of non-positive curvature. An immediate practical result that followed was the discovery of a plan optimization algorithm that was capable of changing a single-move plan into an optimal multi-move plan in the same homotopy class[8].

Rus and Vona advanced planning for a different class of SRSs, those that are comprised of unit compressible modules. In their work, units were assigned into small groups of virtual elements that lay on a coarse embedding lattice space[17]. A dictionary of moves was developed operating on the the coarser abstraction and planning was simplified. The overall time complexity for their planning algorithm was $O(n^2)$.

Constraining units to lie upon a coarse lattice undesirably reduces the permissible tasks that can be planned for from the total set of realizable configurations. Later work by Fitch, Butler and Rus relaxed the necessity for units to remain members of a fixed virtual group and applied the results to a SRS model that closely resembles the hexagonal metamorphic model[5]. The lack of mobility of units was overcome using a procedure called $TunnelSort$. A high level planner would request units to move between arbitrary locations, and the $TunnelSort$ sub-procedure would find a path through the SRS's volume and determine the local coordination of nearby units required to realize the move. Whilst the run-time of the algorithm was of the same complexity as before, $O(n^2)$, spatial resolution was not lost through

coarsening. The state-of-the-art of their continued work is a planning algorithm that also permits dense obstacles in the environment, and a degree of heterogeneity within the SRS[6], with a time complexity of $O(n^2)$.

Recent work by Aloupis *et al.[2]* created a planning algorithm for a unit-compressible SRS. They acheived linear time complexity in a distributed implementation, and the multi-move plans scaled linearly with the number of units. However, only configurations that lay on a coarse lattice could be planned for and, historically, compressible SRSs have been easier to plan for than SRSs whose modules can only move on the surface of the configuration.

There is an unfulfilled need for a reconfiguration algorithm that is sub-quadratic in time complexity, and capable of being applied to a large subset of configurations. This is the focus of our work.

III. Preliminaries

Let $\mathbb{P}$ denote the set of positions on a hexagonal lattice. $d : \mathbb{P} \times \mathbb{P} \mapsto \mathbb{Z}$ is defined as the hex distance (see [3] for details). We say two locations, $x_1 \in \mathbb{P}$ and $x_2 \in \mathbb{P}$ are adjacent as $isAdj(x_1, x_2) \Leftrightarrow d(x_1, x_2) = 1$. The undirected connectivity graph, $\mathcal{G}_{conn}$, of a set of locations, $V \in \mathcal{P}(\mathbb{P})$, is the graph constructed from $G(V, \{(e_1, e_2)|isAdj(e_1, e_2)\})$.

In all models of the hexagonal metamorphic system described here, the configuration space, Σ^* is a set of robotic unit locations, and a set of anchor locations, $\Sigma^* = \mathcal{P}(\mathbb{P}) \times \mathcal{P}(\mathbb{P})$, where $\mathcal{P}$ denotes the power set function. Access to the sets is through the overloaded unary function $\_.R : \Sigma^* \mapsto set\,of\,\mathbb{P}$ for robotic units, and $\_.A : \Sigma^* \mapsto set\,of\,\mathbb{P}$ for the anchor set. Often we will refer to the units of a configuration as $\_.U : \Sigma^* \mapsto set\,of\,\mathbb{P}$ where $c.U \triangleq c.R \cup c.A$. The unnocupied space immediatly adjacent to units in a configuration is denoted $c.Adj = \{p|p \notin c.U \wedge \exists q.q \in c.U \wedge isAdj(q, p)\}$

A move is an ordered pair of positions. If the positions are unrestricted we say it is a *long* move, $\mathbb{M}_L = \mathbb{P} \times \mathbb{P}$. Moves from a catalog are always *short* moves, $\mathbb{M}_S = \{x|x \in \mathbb{M}_L \wedge isAdj(x_1, x_2)\}$. A multi-move represents several moves taken in parallel, $\mathbb{M}_M = \mathcal{P}(M_s)$.

IV. Models

Our motivation is to computationally efficiently form plans for Chirikjian's definition of the metamorphic robot. The model, however, suffers from computational issues arising from its motion constraints. Ghrist's redefined motion constraints, however, do not suffer from the same set of issues and so we make gains in planning efficiency by using the Ghrist's move catalog wherever possible.

In this work we propose a new model, the unconstrained surface model (henceforth abbreviated to surface model), which adds further motion constraints to enable even more efficient planning. So we have a hierarchy of models, Chirikjian, Ghrist and surface. Each successive model is further constrained than the previous, so a plan executable by the surface model of reconfiguration can be executed by the Chirikjian model of reconfiguration. By planning for the most constrained model wherever possible, we can achieve a massive reduction in planning time. Each model is described in detail below.

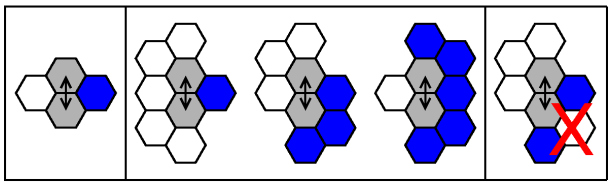

Fig. 1. Ghrist's diagrammatic notation for representing motion catalogs[8]. The trace of the move is shown in grey. The trace is where a single robotic unit is free to move between (shown by an arrow). The support set, the remaining labeled locations, denote what the local context must be in order for the move to be valid; blue for robotic units, and white for empty space. Catalog elements are matched to the embedding space with a rigid body translation. Left, the Chirikjian motion catalog[8], in addition, the moving robot must not be a cut vertex of $\mathcal{G}_{conn}(c.U)$. Middle, the Ghrist model's motion catalog[8]. Right, and example of a move that may be permitted by the Chirikjian catalog, but not Ghrist's. Ghrist's motion catalog implies gross topological changes in the robot morphology cannot occur.

A. Chirikjian

A configuration, c, belongs to the Chirikjian class of configurations, $\mathbb{C} \subset \Sigma^*$, if the units form a connected graph:- $c \in \mathbb{C} \Leftrightarrow isConnected(\mathcal{G}_{conn}(c.U))$

Robotic units pivot around a neighbor in order to move. There must be space for the robotic unit to pivot into[1] (Figure 1), and the moving unit must not be a cut vertex with respect to the connectivity graph. Anchor nodes behave like normal units from the perspective of constraint checking, but are not themselves permitted to move.

Pattern matching the catalog to a specific locale can be done in constant time. Checking a unit is a cut vertex, however, is a global constraint. Testing this constraint for all modules in a configuration, c, can be achieved in one pass by calculating the biconnected components of the connectivity graph, $\mathcal{G}_{conn}(c.U)$ at a time cost of $O(n)$[9].

B. Ghrist

The Ghrist version of the hexagonal metamorphic robot has more constraints to movement than the Chirikjian definition. For notational convenience we shall denote the permissible space of configurations as $\mathbb{G}$, albeit it is the same as $\mathbb{C}$. The difference between models lies only in their motion catalogs (Figure 1). The Ghrist catalog of moves ensures there is no possibility for the global topology of the robot to change. So if the aggregate is in a shape with no holes, it will remain so regardless of what moves are executed. Thus, the total reconfiguration state space of the Ghrist model is disconnected.

Ghrist showed the additional restrictions implied non-positive curvature of the reconfiguration state space. He then described how a single move planning solution can be changed to a multi move version at a time cost of $O(p^2)$[8], where p represents the path length.

The different move catalog also has useful computational benefits. The costly connectivity check that the Chirikjian

[1] Actually, the original definition did not require pivot space. However, the problem is harder and more interesting with it. The Claytronics hardware platform is an instantiation of these exact motion constraints[12].

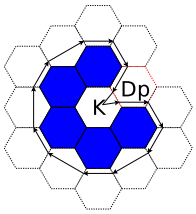

Fig. 2. The Euler tour, ET, around the surface for an example configuration. Robotic units are shaded blue. The two possible surface violations are shown. A kink is denoted K, and a dual path is shown as Dp.

model catalog requires is not necessary for the Ghrist model. Thus the set of applicable moves for a configuration can be updated after a move is taken dynamically at constant time cost.

C. Surface

Our third model is a further constrained version of the Ghrist model. The space is denoted by $\mathbb{S} \subset \mathbb{G}$. In order to define this space we require an additional data structure. An Euler tour (ET) around the adjacent external space is maintained using a spatially indexed linked list data structure. This data structure is represented by surface elements stored in a tree indexed by hex coordinates. A surface element is comprised of three pairs of *directional* pointers (Figure 2).

The first element of each pair of direction pointers represents the incoming direction of a Euler traversal visitation, and the second element represents the outgoing direction.

The ET is represented using indirect directional pointers, rather than absolute references to adjacent surface elements. This permits part of the tour to be rewritten without having to update all pointers. Therefore, the ET can be updated after a single-move at constant cost.

In addition to the tour, several other statistics about the tour are maintained; a set of kink violations, and a set of dual path violations. A kink violation is the location of where a Euler tour visitations enters and leaves through the same edge. A dual path violation is a location that is visited more than once on the tour. Access to the set of kink violations is denoted by the function $\_.K : \Sigma^* \mapsto set\,of\,\mathbb{P}$ and the dual paths by $\_.D : \Sigma^* \mapsto set\,of\,\mathbb{P}$

A configuration c is said to be a member of the unconstrained surface state space when there are no kink or surface violations i.e. $|c.K| = 0 \wedge |c.D| = 0 \Leftrightarrow c \in \mathbb{S}$. The motivation for this definition is to permit unconstrained motion for mobile units around the surface. If unit can move, then by definition from the Ghrist catalog it is on the surface of the robotic volume. Prevention of a move by lack of pivoting space is impossible because no dual path violations implies there is always space around the surface. A change in global topology cannot occur because no kinks exist with which a mobile unit could bridge. Thus, for the surface model, it is unnecessary to check the intermediate states when moving a unit from a location on the surface to another location on the surface.

For convenience we define a function $LongMove$ which moves a robotic component on a configuration to anywhere, updating the ET as it does so (Algorithm 1). There is no validity checking in this function, but this is done elsewhere in the algorithms presented later.

Algorithm 1 $LongMove$ moves a long distance, and updates the Euler tour in constant time.

$LongMove : \Sigma * \times (M_L) \mapsto \Sigma *$
$LongMove(c_s, (m_s, m_e)) \triangleq c_e$
$c_e \leftarrow ((c_s.R - \{m_e\}) \cup \{m_s\}, c_s.A)$
$c_e.ET \leftarrow UpdateEulerTour(c_s.ET, m_s, m_e)$

Plans executable in the Chirikjian reconfiguration state space can reach configurations belonging to the surface state space because $\mathbb{S} \subseteq \mathbb{G} \subseteq \mathbb{C} \subset \Sigma^*$. As moves are reversible (Figure 1), *i.e.* $move(c_s, m) = c_e \models move(c_e, m^{-1}) = c_s$, a path from $c \in \mathbb{C}$ to $s \in \mathbb{S}$ can be reversed to find a path from s to c.

V. Planning

Our target is to determine a plan between two arbitrary Chirikjian configurations. In this section we describe how surface-to-surface reconfiguration tasks can be solved efficiently. We then show how Chirikjian configurations can be converted to surface configurations. The two forms of previous plans can be combined into one single-move plan. Finally, we describe how the single-move plan is converted to a multi-move plan.

As some of our algorithmic claims are empirically derived, we describe our empirical experiments here, with the relevant data interleaved with the algorithmic descriptions later. The data from first set of experiments was used to test the most efficient sub-planning stages. A 100 random configurations belonging to $\mathbb{C}$ containing $1000x$ units were generated for $x = 1 \ldots 20$. For the second set of experiments, 100 configurations were generated containing $100x$ units for $x = 1 \ldots 35$.

Random configurations were generated iteratively starting with a single anchor node. Mobile robotic nodes were added by uniform randomly selecting a component, then uniformly randomly selecting an empty neighbor (if one existed) and placing a unit there until the desired number of units were placed. While our results would be more representative of average case behavior if configurations were sampled directly from the state space of $\mathbb{C}$, this itself is an open and challenging problem[15]. We do not believe that potential suboptimality in this random sampling affects the average case complexity in a qualitative way.

A. Surface-to-Surface

Surface-to-surface planning determines a set of moves that change from one surface adhering configuration to another. Surface-to-surface planning does not need to consider intermediate single-move motions.

The planning algorithm incrementally improves a current configuration c towards a goal configuration g. Locations in the embedding space are labelled from $L = \{PLACED, GROW, GROW^+, CONTRACT, \emptyset\}$(Figure 3).

A location labeled $PLACED$ denotes a robotic location that no longer needs to be considered in order to improve c. The anchors are considered $PLACED$ on initialization,

Algorithm 2 Updating the labeling for $PLACED$ and $GROW$ is initiated at a location, loc. If the loc label changes to $PLACED$, the function recurs.

$updateInc : \mathbb{P} \times \mathbb{S} \times \mathbb{S} \times (\mathbb{P} \mapsto L) \mapsto (\mathbb{P} \mapsto L)$
$updateInc(loc, c_{curr}, c_{goal}, labels) \triangleq labels$
  if($loc \in c_{curr}.U \wedge loc \in c_{goal}.U$)
    $labels(loc) \leftarrow PLACED$
    for($\forall q.isAdj(q, loc)$)
      $updateInc(q, c_{curr}, c_{goal}, labels)$
  if($loc \notin c_{curr}.R \wedge loc \in c_{goal}.R$)
    if($(c_{curr}.R \cup \{loc\}, c_{curr}.A) \in \mathbb{S}$)
      $labels(loc) \leftarrow GROW^+$
    else $labels(loc) \leftarrow GROW$

Algorithm 3 Updating the contraction labels is performed in patches of radius 2 around the location, loc. The $canMove$ function tests whether a given location on a configuration can move according to the Ghrist catalog.

$updateArea : \mathbb{P} \times \mathbb{S} \times \mathbb{S} \times (\mathbb{P} \mapsto L) \mapsto (\mathbb{P} \mapsto L)$
$updateArea(loc, c_{curr}, c_{goal}, labels) \triangleq labels$
  for($\forall q.d(q, loc) \leq 2$)
    if($canMove(q, c_{curr}) \wedge q \notin c_{goal}.R$)
      $labels(loc) \leftarrow CONTRACT$

and robotic units adjacent to placed units that are also found in the goal configuration are considered placed too. As the $StoS$ planner never moves $PLACED$ units, the number of placed locations only grows. As placed units are adjacent to already placed units, the labeling of placed units are updated incrementally in the function $updateInc$ (Algorithm 2).

$GROW$/$GROW^+$ locations are where robotic units could be placed. As such they are: always empty, adjacent to locations labeled $PLACED$, and where robotic units are located in the goal. The $GROW$ labels are updated incrementally by $updateInc$ (Algorithm 2).

$CONTRACT$ locations denote: locations where robotic units currently are that can move, and locations which are not occupied in the goal configuration. $CONTRACT$ locations provide a supply of units for moving into $GROW$/$GROW^+$ locations. After a move is applied to the current configuration, some units local to the move's start and end locations may become mobile or lose mobility. Thus, updating the $CONTRACT$ labels is a local operation to be applied after the configuration changes, at constant time cost by the function $updateArea$ (Algorithm 3).

The superscript $+$ is appended to the $GROW$ label when the addition of a robotic unit at that location results in a valid surface configuration. Movement to $GROW$ locations only results in a valid configuration if the removal from the corresponding $CONTRACT$ changes the local context of the $GROW$ area. Thus, we prioritize consideration of movements to $GROW^+$ locations because it is likelier that the move will result in a valid surface configuration; and reduces the amount of movements considered in $improve()$.

The $StoS$ planning algorithm iterates until all units in the current configuration are $PLACED$. It improves the current configuration by moving units from $CONTRACT$ locations to $GROW$ locations (Algorithm 4). As units become $PLACED$ by an incremental traversal of the connectivity graph, the total time of all calls to $updateInc$ is $O(n)$. The time cost of $improve$ depends on how many movement attempts are rejected because they fail to result in a valid surface configuration. Further, the number of calls to $improve$ depends on how many times the planning algorithm must iterate. Figure 5B, shows that empirically the number of times the planning algorithm iterates is approximately $k\sqrt{n}$. Figure 5A, shows that the number of $GROW$–$CONTRACT$ pairs considered by $improve$ grows very slowly with problem complexity.

Units, n	fails	trials	95% C.I. of P(fail)	
250	264	10000	.0233	.0297
500	7	10000	.0003	.0014

TABLE I
PROBABILITY OF $StoS$ FAILING

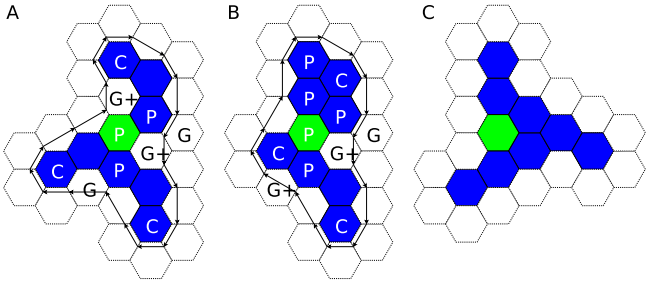

Fig. 3. The labeling of $PLACED$, $GROW$,$GROW^+$, $CONTRACT$ and the Euler Tour for two configurations (A and B) planning toward a goal (C). B is one possible *longMove* option $improve$ could suggest given A.

There are, however, cases where no improvements can be found and an error is generated (see Figure 4). If this occurs, an random intermediate configuration is generated for c_{start} and c_{goal} to be planned between. Failures become less likely as the number of units in the configuration increases (Table I) and presumably become irrelevant w.r.t. time complexity as $n \rightarrow \infty$.

The overall wallclock time of planning, including failure resolution, is shown in figure 5D. Computation time scales linearly with problem complexity in normal circumstances, but spikes are apparent where the system failed to find a solution in one attempt. As discussed, we believe these spikes to be irrelevant to time complexity, asymptotically.

B. Chirikjian-to-Surface

The Chirikjian configuration state space permits the overall morphology to contain holes. These cannot be removed

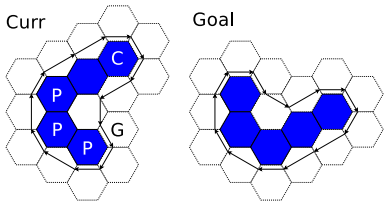

Fig. 4. Example of an error situation. Moving the unit at the only $CONTRACT$ location to the only $GROW$ location (thus forming the configuration in Figure 2) causes a kink violation *i.e.* the StoS planner is stuck

Algorithm 4 The $\mathbb{S}$urface-to-$\mathbb{S}$urface planner.

$improve : \mathbb{S} \times (\mathbb{P} \mapsto L) \mapsto \mathbb{S} \times \mathbb{M}_L$
$improve(c, labels) \triangleq$
  for $\{s, e | (labels(s) = CONTRACT^+$
    $\vee labels(s) = CONTRACT)$
    $\wedge (labels(e) = GROW^+$
    $\vee labels(e) = GROW)\}$
    $\hat{c} \leftarrow LongMove(c, (s, e))$
    if $(\hat{c} \in \mathbb{S})$
      return $(\hat{c}, (s, e))$
throw error

$StoS$: $\mathbb{S} \times \mathbb{S} \mapsto M_L^k \times \mathbb{S}^k$
$StoS(c_{start}, c_{goal}) \triangleq (M, S)$
$c \leftarrow c_{start}$
for $\{a | a \in c.A\}$
  $labels \leftarrow updateInc(c, g, labels)$
for $\{x | x \in c.R\}$
  $labels \leftarrow updateArea(x, c, g, labels)$
while $(\exists l. l \in c.R \wedge labels(l) \neq PLACED)$
  $((s, e), c) \leftarrow improve(c, labels)$
  $labels \leftarrow updateInc(e, c, g, labels)$
  $labels \leftarrow updateArea(s, c, g, labels)$
  $labels \leftarrow updateArea(e, c, g, labels)$
  $append(M, (s, e))$
  $append(S, c)$

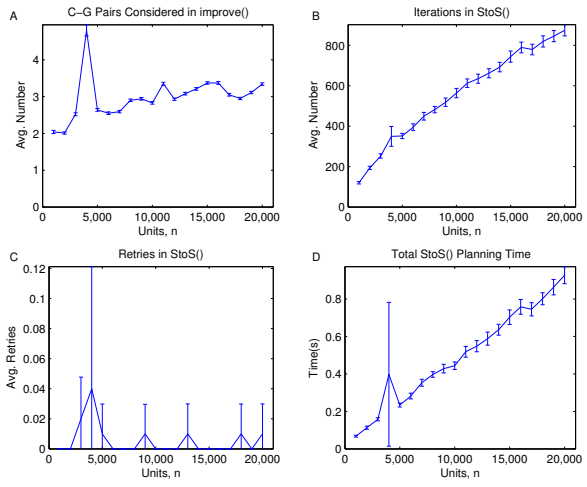

Fig. 5.

by the Ghrist move catalog, so the aim of the Chirikjian-to-surface conversion step is to plan a set of moves that removes any holes from the initial and goal configurations. Holes are removed by finding tunnels linking all holes and excavating units inside these tunnels to safe locations elsewhere on the configuration.

Firstly, it should be noted that from the moves catalog of the Chirikjian model it is clear that components cannot move along the walls of an empty tunnel one space wide. So a two wide tunnel needs to be determined that links all holes. This is done in a two step process. In the first step a one wide tunnel is found and in the second step, the one wide tunnel is expanded to be two wide.

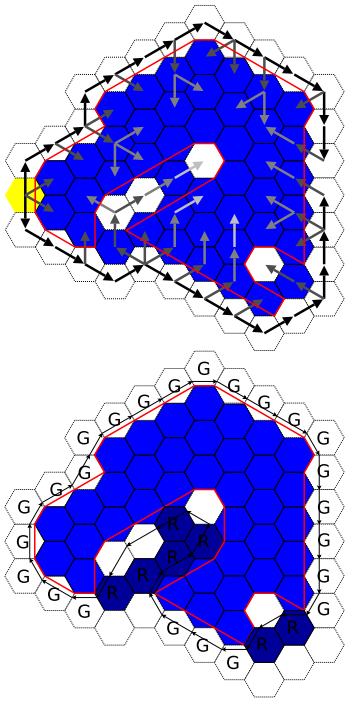

Fig. 6. Top, the one wide tunnel procedure. The large arrows denote the order of Dijkstra expansion. The shade of the arrows indicate the geodesic distance from the external unoccupied start node (yellow). The red highlights the nodes in the minimal spanning tree that connects all unoccupied space. Bottom, the expansion of the one wide tunnel to two wide.

To find the one wide tunnel for a configuration, $c \in \mathbb{C}$, a connectivity graph $G_t = \mathcal{G}_{conn}(c.R \cup c.Adj)$ is constructed that includes the adjacent unoccupied location but not the anchor nodes. The edges are given weights by the (directed) function $w(x_1, x_2) \triangleq if(x_2 \in c.Adj)\ 0\ else\ 1$

The weight function implies traversals into unoccupied locations on this graph "cost" nothing. To find a one wide tunnel we initiate a Dijkstra shortest path traversal initiated at an outside adjacent location. The traversal is terminated as soon as all unoccupied locations have been visited. The minimal spanning tree of the search graph containing all search nodes that expand unoccupied locations identifies the minimal set of units that connect all holes with a one wide tunnel (Figure 6). The time to perform the Dijkstra search is linear. Taking the minimal spanning tree is also linearly bounded. So overall determining what set of units lie within the one wide tunnel has a computation time of $O(n)$. We denote the operation as $tunnel_1(c \in \mathbb{C}) \mapsto R_1 \in \mathcal{P}(\mathbb{P})$.

The second step in the tunneling procedure is to expand the one wide tunnel to be two wide. Determining what side of a one wide section of tunnel to dig, in order to achieve the minimal set of nodes to remove, seems expensive to compute. Instead we chose a linearly bounded greedy procedure. First, all units of R_1 are removed from the configuration to yield a hole-free configuration $c' = (c.R - R_1, c.A)$. Then, every adjacent unoccupied location, $c'.Adj$, is checked to determine whether it is a one wide passage or a kink (figure 7). If it is, then the locally minimal set of neighbor units to undo its status are marked for removal, and c' is updated to reflect this (algorithm 5). Anchor locations cannot be part of

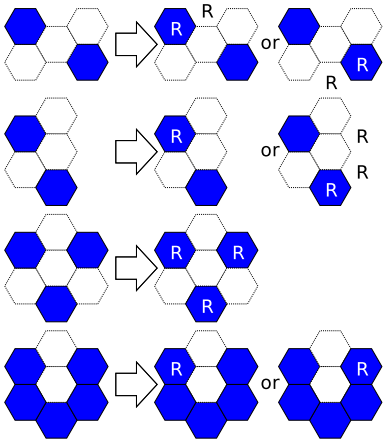

Fig. 7. The unoccupied locations which need to be corrected in order to create a two wide tunnel. On the left is the matching context, on the right possible fixes, where R denotes which locations should be added to the removal sets. Only the major classes of cases are shown, subject to rigid body transform.

Algorithm 5 Expansion of a one wide tunnel, (Figure 6) is achieved by matching unoccupied adjacent space to problem categories (Figure 7). If a problem is found, the solution that requires the minimal amount of additional R tunneling units is selected as the local solution.

$tunnel_2 : \mathbb{C} \mapsto set\,of\,\mathbb{P}$
$tunnel_2(c) \triangleq R$
$R_1 \leftarrow tunnel_1(c)$
$R \leftarrow R_1$
$c' = (c.R - R, c.A)$
for $\{e | e \in A_{c'}\}$
  if $(p \leftarrow matchProblem(e) \neq \emptyset)$
    $m \leftarrow \infty$
    for $\{s | s \in solutions(p)\}$
      $q \leftarrow |s \cap c'.R - R|$
      if $(q < m \wedge s \cap c.A = \emptyset)$
        $m \leftarrow q$
        $s_{min} \leftarrow s$
    $R \leftarrow R \cup s_{min}$
    $c'.R \leftarrow c'.R - s_{min}$

the removal set. This procedure is named $tunnel_2$.

Robotic units to be moved are identified by $tunnel_2$ as a set R (figure 6). Removal of R from $c.R$ yields a surface configuration with which an unviolated ET, W, (described in the definition of a surface robot) can be wrapped around. W provides a "roadmap" to determine what surface locations are safe for robotic units to be moved to. The growth set, G, is defined as locations where placement of a robotic unit does not cause a violation in the ET.

The planner solves the task by moving units at a location in R to a location in G. For speed, the planner tries to achieve these goals by using the Ghrist's motion catalog, and only uses the Chirikjian model's catalog when necessary.

The Ghrist catalog can be too limiting in some situations. The Ghrist catalog's motion constraints prevent a unit on the boundary of a hole from moving to open the hole (Figure 8). However, by definition, a hole is enclosed by a boundary of robotic units, so the global topological change of opening the hole cannot disconnect the overall connectivity of the robot. While the Chirikjian catalog would permit the move, it comes at a linear cost of a connectivity check. We can avoid this by filling all enclosed holes in the configuration with virtual robotic units, which will allow the move to be identified by the Ghrist catalog at constant cost. After the move has taken place, a virtual element will be in contact with empty space. All virtual elements in contact with empty space are removed recursively. As the Chirikjian-to-surface converter is only removing holes from the configuration, initial identification of virtual elements is $O(n)$ and the total time for removing all elements is also $O(n)$ (in much the same fashion of maintenance of the set P in the $StoS$ planner).

To improve efficiency further, rather than trying to move elements of R out directly, a subset of R is maintained, R^+, which denotes those locations of R that contain units that can move according to the Ghrist catalog.

Figure 9A, shows the that the number of times the Ghrist catalog is used to remove elements from R scales as $\sqrt{n}$. In comparison, the number of times the Chirikjian catalog is used scales very slowly, and in many problem instances is not needed at all (Figure 9B). The number of search substeps required to find a path for a unit from R to G appears to be independent of problem complexity (Figure 9C).

The algorithm only fails to remove a unit from R^+ when doing so disconnects the overall robot structure which, by definition, violates the Chrikjian's motion constraints. Failure occurs when the tunneling procedure creates a tunneling structure that partitions the robot. So far we have only implemented an *ad hoc* partial solution to this problem. We allow the planner to initially solve as much as possible, and rerun the tunneling algorithm using added noise to the weight function in $tunnel_1$ when the planner gets stuck. This permits the tunneling procedure to try different tunneling structures. The planner retries tunneling a maximum of 10 times before deciding the conversion is impossible. Whilst this means some conversion attempts fail, the noisy tunneler does allow some extra configurations to be solved that initially couldn't. The majority of randomly generated configurations are solvable by this procedure (Figure 9E). Retunneling $O(n)$ in time complexity, but the number of times it is required scales very slowly with problem complexity (Figure 9D).

The overall wallclock time to compute Chirikjian-to-surface plans is shown in Figure 9F. The data suggest that, for most cases, the computation time is scaling linearly. There is an exception in an experiment with 12,000 units. We cannot explain this anomaly, as it does not correlate with any other spikes in the other metrics. It may be due to a Java garbage collection pause.

C. Chirikjian-to-Chirikjian planing

The Chirikjian-to-Chirikjian planner uses each of the planning algorithms described above as sub-steps to creating an overall single move plan to between arbitrary Chirikjian configurations.

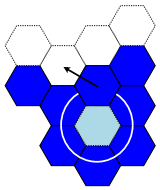

Fig. 8. The Ghrist catalog is unable to open an enclosed hole. By adding a virtual robotic element to the configuration (light blue), a Ghrist catalog move becomes applicable. After the move, the virtual robotic element can be removed from the configuration.

Algorithm 6 The $\mathbb{C}$hirikjian-to-$\mathbb{S}$urface planner

$C\_to\_S : \mathbb{C} \mapsto M_S^k \times \mathbb{C}^k \times \mathbb{S}$
$C\_to\_S(c) \triangleq (m, s, c)$
$m \leftarrow \emptyset$
$R \leftarrow tunnel_2(c)$
$W \leftarrow (c.R - R, c.A).E$
while$(|R| > 0)$:loop
  $R^+ = \{r | r \in \mathbb{P} \wedge canMove(r, (c.R \cup holes(c), c.A))\}$
  for $\{u | u \in R^+\}$
    if (p =findAnyPathGhrist(c, u, R)$\neq null$)
      $c, R, W, m, R^+ \leftarrow$update($u$,end($p$)), **goto** loop
  for $\{u | u \in R^+\}$
    if(p =findAnyPathChirikjian(c, u, R)$\neq null$)
      $c, R, W, m, R^+ \leftarrow$update($u$,end($p$)), **goto** loop
  if($tries = 10$) **return** error
  $tries =\leftarrow tries + 1$
  $R \leftarrow tunnel_2 noisy(c)$
  $W \leftarrow (c.R - R, c.A).E$

Given a start and end configuration, $s \in \mathbb{C}$, $e \in \mathbb{C}$, a single move plan is formed for each that transforms them into surface adhering configurations. A plan between these surface adhering configurations is found using $StoS$. This compressed plan contains *longMove*s, which is decompressed by searching the Ghrist state space. As $StoS$ plans, on average, contain $\sqrt{n}$ long moves, which require $\sqrt{n}$ Ghrist moves to realize, decompression costs $O(n)$ (Figure 10A).

Finally, the separate plans are concatenated together to yield a plan that is a set of single moves that changes s to e.

D. Single-Move to Multi-Move plan conversion

The planning algorithms discussed so far achieve their aims by moving one component at a time. However, the time

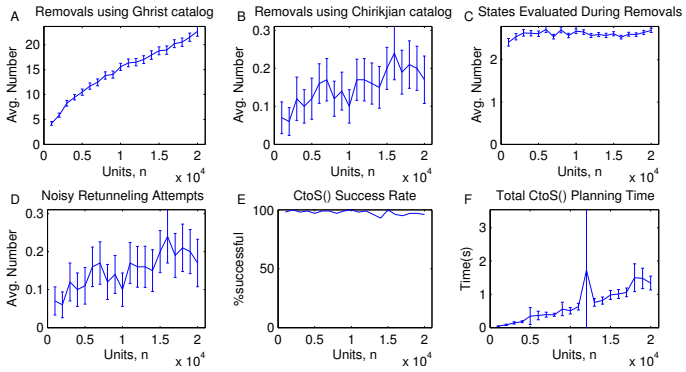

Fig. 9.

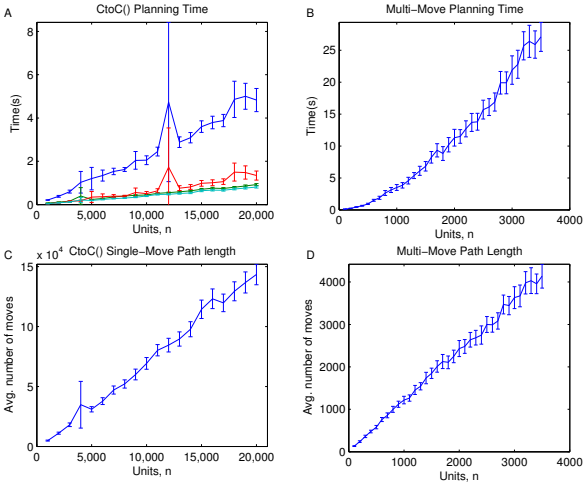

Fig. 10.

required to execute a plan on a hardware platform can be drastically improved if multiple units are permitted to move in parallel.

Ghrist provides[8] a method for converting a single-move plan into an optimal multi-move plan for moves using the Ghrist catalog. Once the single-moves plan is calculated, which contains moves from both Chirikjian and Ghrist catalogs, Ghrist's multi-move conversion algorithm can be applied. The existence of Chirikjian moves in the plan does not pose a problem. Ghrist's conversion algorithm is just applied to the solution sub-sequences that contain Ghrist moves.

The overall time to convert the single-move compressed plan to a multi move plan empirically appears to be bounded quadratic, in accordance with Ghrist's own analysis (Figure 10B). We note though, that when the multi-move conversion step is applied, if all intermediate states and moves are available then the conversion could be computed in a distributed fashion. So we propose representing the underlying state of configurations using *persistent* red black trees[4]. *Persistent* red-black trees can represent sets with all the normal operations taking $O(log_2(n))$ but modifications to the sets preserve the original versions at $O(log_2(n))$ in space cost. Using *persistent* red-black trees would increase the cost of computing a single-move plan from $O(n)$ to $O(nlog_2 n)$ but would permit all intermediate solution configuration states to be preserved. Then, a distributed version of Ghrist's multi-move conversion step could compute a multi-move plan in $O(nlog_2 n)$.

VI. Results

The total time to form a single move plan from two randomly generated Chirikjian configurations is shown in figure 10A. The total time to plan and convert into a multi-move plan is shown in figure 10B. Sometimes the planning algorithm can fail, caused by tunneling problems in the $CtoS$ converter. The failure rate is shown in Figure 9E. On average, for over 97% of randomly generated tasks with up-to $20,000$ units, our algorithm finds a single move plan in linear time. While some of the $CtoS$ computation times

spike, the magnitude of the spike is still linearly bounded because only a finite number of $O(n)$ retries are attempted.

The average single-move path length is shown in figure 10C. The single-move path length scales linearly with the number of units in the configuration. The multi-move path lengths are shown in figure 10D. Moving units in parallel reduces the number of time steps necessary to change from one configuration to another by several orders of magnitude. The multi-move plans scale sub-linearly with problem complexity, but not $\sqrt{n}$ as may be hoped.

Some authors have conjectured that Pamecha *et al.*'s optimal assignment heuristic[16] results in plans that are near optimal. Recent work [13] found that the combination of greedy search guided by the optimal assignment heuristic provides shorter solution paths than many other general purpose motion planning techniques. However, the plans formed by this technique grew faster than linear with problem complexity, so we can immediately deduce that the planner in our work produces better plans.

VII. Discussion

A common extension to the SRS reconfiguration planning problem is the addition of obstacles in the space. Our algorithm presented here is easily extended to this situation. The dual path violation case can be generalized to indicate when an obstacle blocks pivot space around the surface of the robot.

The simple solution to obstacles, avoiding movement in the vicinity regardless of the current planning task, is reminiscent of a conservative approach to calculating a subset of $\mathcal{C}_{free}$ by taking the Minkowski sum of a bounding sphere with obstacles in the environment [10]. Our approach in $StoS$ planning can be viewed as a utilization of this insight.

The SRS domain is difficult for planning because the robotic units are obstacles to each other. As we can control the robots though, this can be used to our advantage. Our approach has been to keep the surface of the configuration unconstrained for robot movement, in a sense, maximizing $\mathcal{C}_{free}$ for those robotic units who can move. This has enabled efficient planning over a large spanning subset of the Chirikjian reconfiguration state space.

We have described an algorithmic procedure for solving the self-reconfiguration problem and empirically shown that it has $O(n)$ complexity averaged over random problem instances. It would be desirable to characterize and analyze the reason for this improvement. At this time we conjecture that the state space of the unconstrained surface model is a discretization of Kendall's pre-shape space [11] for general shapes embedded in a Euclidean space. This would imply the surface state space is homomorphic to a lower dimensional topological hypersphere. We believe it is this structure that the $StoS$ planner is exploiting. In future work we will attempt to formally identify a qualitative difference between the general state space, $\mathbb{C}$ and the surface state space, $\mathbb{S}$, which would explain $\mathbb{S}$'s amenability for efficient planning.

VIII. Conclusion

Presented here is, what we believe to be, the first $O(n)$ centralized algorithm for single-move SRS planning, and a sketch of a $O(nlog_2n)$ distributed multi-move planning algorithm, that operates on the vast majority of the k^n sized reconfiguration state space for the hexagonal metamorphic robot.

Further improvements may be achieved by lowering the time complexity of the path optimizing algorithm of Ghrist[8], increasing the applicability of the algorithm to the entire state space of $\mathbb{C}$ or reaching $O(\sqrt{n})$ path lengths for multi-move plans.

References

[1] A. Abrams and R. Ghrist. State complexes for metamorphic robot systems. *Intl. J. of Robotics Research*, 23(7):809–824, 2004.

[2] Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Comput. Geom. Theory Appl.*, 42(6-7):652–663, 2009.

[3] G. S. Chirikjian. Kinematics of a metamorphic robotic system. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, volume 1, pages 449–455, 1994.

[4] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1), 1989.

[5] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. *Intelligent Robots and Systems*, 3:2460– 2467, 2003.

[6] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning among obstacles for heterogeneous self-reconfiguring robots. *Robotics and Automation*, pages 117 – 124, 2005.

[7] R. Ghrist and V. Peterson. The geometry and topology of reconfiguration. *Advances in Applied Mathematics*, 38:302–323, 2007.

[8] Robert Ghrist. Shape complexes for metamorphic robot systems. *Proc. Workshop in Algorithmic Foundations of Robotics*, 2002.

[9] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.

[10] Seth Hutchinson George A. Kantor Wolfram Burgard Lydia E. Kavraki Howie Choset, Kevin M. Lynch and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[11] D. G. Kendall, D. Barden, T.K. Carne, and H. Le. *Shape and Shape Theory*. Wiley, 1999.

[12] B.T. Kirby, B. Aksak, J.D. Campbell, J.F. Hoburg, T.C. Mowry, P. Pillai, and S.C. Goldstein. A modular robotic system using magnetic force effectors. *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2787–2793, 29 2007-Nov. 2 2007.

[13] Tom Larkworthy, Gilian Hayes, and Subramanian Ramamoorthy. General motion planning methods for self-reconfiguration planning. *Towards Autonomous Robotic Systems (TAROS 2009)*, 2009.

[14] Tom Larkworthy and Gillian Hayes. Utilizing redundancy in modular robots to achieve greater accuracy. *Robocomm*, 2009.

[15] Daniel Martins and Roberto Simoni. Enumeration of planar metamorphic robots. *ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*, pages 610–611, 2009.

[16] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *Robotics and Automation, IEEE Transactions on*, 13(4):531–545, Aug 1997.

[17] D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. *IEEE International Conference on Robotics and Automation*, 4:2513–2520, 1999.

[18] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems – challenges and opportunities for the future. *IEEE Robotics and Automation Magazine*, March:43–53, 2007.