Generalizing Metamodules to Simplify Planning in Modular Robotic Systems

Daniel J. Dewey
Michael P. Ashley-Rollman
Michael De Rosa
Seth Copen Goldstein
Todd C. Mowry
School of Computer Science
Carnegie Mellon University

Siddhartha S. Srinivasa
Padmanabhan Pillai
Jason Campbell
Intel Research Pittsburgh

***Abstract*— In this paper we develop a theory of metamodules and an associated distributed asynchronous planner which generalizes previous work on metamodules for lattice-based modular robotic systems. All extant modular robotic systems have some form of non-holonomic motion constraints. This has prompted many researchers to look to metamodules, i.e., groups of modules that act as a unit, as a way to reduce motion constraints and the complexity of planning. However, previous metamodule designs have been specific to a particular modular robot.**

By analyzing the constraints found in modular robotic systems we develop a holonomic metamodule which has two important properties: (1) it can be used as the basic unit of an efficient planner and (2) it can be instantiated by a wide variety of different underlying modular robots, e.g., modular robot arms, expanding cubes, hex-packed spheres, etc. Using a series of transformations we show that our practical metamodule system has a provably complete planner. Finally, our approach allows the task of shape transformation to be separated into a planning task and a resource allocation task. We implement our planner for two different metamodule systems and show that the time to completion scales linearly with the diameter of the ensemble.

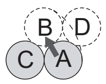
I. Introduction

There is growing interest in using self-reconfigurable modular robots (MRs) as metamorphic systems: i.e., robots that approximate arbitrary 3D shapes via physical rearrangement of modules for 3D visualization, programmable antennas, entertainment, or general-purpose robotics. To achieve macroscale objects with good spatial resolution an MR system requires a very large number of very small modules: e.g., thousands to millions of centimeter- to millimeter-scale modules [?]. One of the main challenges for these systems is an efficient planner. It has long been recognized that traditional methods are unsuitable due to the large search space and the blocking constraints imposed by realizable module design. To ease the planning problem, many groups have proposed different kinds of metamodules, groups of modules that act as a unit for planning or motion execution purposes, each specific to a particular module design [?], [?], [?].

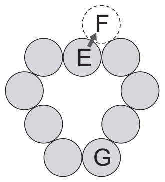
In this paper we develop a theory of metamodules and an associated planner which generalizes previous work on metamodules for lattice-based MR systems. We demonstrate how metamodules can be derived from the structure of the *reconfiguration graph*—whose vertices are feasible configurations and edges are unit physical motions that produce reconfiguration. Our goal is to construct a metamodule system where two points are adjacent in the space of possible configurations if and only if they can be reached by a single move in the reconfiguration graph. In other words, if two configurations are legal and differ by only one module position there is a one step transition between those configurations. This makes it much simpler to construct a heuristic which effectively explores the possible paths between configurations.

We begin by examining the types of constraints that make planning hard for MR systems. These constraints fall into two general categories: (1) local constraints, introduced by the specific design of the modules, e.g., motion or blocking constraints and (2) non-local constraints imposed on the overall system, e.g., that the system remain connected. We show that blocking constraints are a form of discrete nonholonomic constraints. Building on [?] we develop a set of movement primitives and an ideal metamodule system which eliminates local constraints. This ideal metamodule system, which we call *pixel*, has but one such primitive: a module can be created or destroyed at any point in the lattice. It is trivial to show that in this ideal system state-space adjacency guarantees reconfiguration-graph adjacency, and thus it is clearly holonomic. We use *pixel* to show that practical realizable metamodule systems are holonomic by showing that a practical system can be reduced to *pixel*, thus showing that it is also holonomic.

We develop a class of practical metamodule systems based on *pixel* that use voids in the lattice to eliminate local constraints. This can be seen as a generalization of previous work which moved voids around to form shapes [?], [?] and work which built scaffolds to ensure their are empty spaces in the lattice in which to move units [?]. The generalized metamodule system allows a metamodule to hold a variable number of constituent modules while maintaining connectivity with all its adjacent neighbors. Motion in the system is accomplished by the exchange of modules between metamodules while maintaining connectivity and structural stability. We show that this general system is isomorphic to the idealized *pixel* system in terms of reconfiguration capability, allowing us to prove properties of the system in *pixel* and have those proofs hold in the practical system.



(a) Local Constraint



(b) Global Constraint

Fig. 1. **Local and global constraints:** (a) A's movement to position B requires a module in position C to pivot about and empty space in positions B and D to move through. This is a local constraint. (b) E can only move to F if the group as a whole remains connected. This means that removal of any module, G for example, makes the E-to-F movement illegal.

Using our generalized metamodule system we develop a distributed asynchronous planner which enforces non-local constraints and is provably complete; if there exists a plan which maintains global connectivity, then the planner will find that plan. One of the convenient properties of our metamodule planner is that it separates the task of reconfiguration planning (i.e., determining which metamodules should move where) from resource allocation (i.e., determining where a module should come from when one is needed). Using two different, but equivalent, metamodule systems we show a lower bound on the amount of work that needs to be performed to achieve a reconfiguration and then compare this to an implementation that uses a random allocator. More sophisticated resource allocators can be substituted into the same planner to improve the performance while still maintaining the provable properties of the planner.

Finally, we show that our metamodule systems is general in that we describe how the same metamodules and planners can be used for three vastly different MR systems: a robot arm, a cubic telecube-style [**?**] robot, and a 2D hexagonal sphere packing.

A. Related Work

Some of the early results on reconfiguration planning are presented in Chirikjian *et al.* [**?**]. They show that the search space is exponential in the number of modules and stress the importance of good heuristics, such as the Linear Sum Assignment Problem (LSAP) [**?**] and those generated by the *Hungarian Method*. Subsequent papers [**?**], [**?**], [**?**] propose other heuristics which exhibit similar properties for specific metamorphic systems. Walter *et al.* [**?**] offers a survey of reconfiguration planning algorithms and heuristics. Yoshida *et al.* [**?**] present an alternative method using a cluster flow algorithm and generators to bypass nonholonomic constraints.

Nguyen *et al.* [**?**] observed that heuristics perform poorly due to *blocking constraints*; in a tightly packed ensemble motion of the modules in the interior is blocked by their neighbors. To resolve this, Nguyen *et al.* and Kotay and Rus [**?**] independently suggested the construction of *metamodules*—groups of modules behaving as a single unit, with sufficient space in their interiors to absorb other metamodules. Alternatively, Kotay and Rus [**?**], and Stoy and Nagpal [**?**] suggested the used of open static scaffolding structures in cubic modules which permit modules to pass through the scaffolding, while De Rosa *et al.* [**?**] suggested a hole-motion algorithm which moves empty space randomly within the interior of packed structures. The basic idea behind all of these approaches is to alleviate blocking by making structures porous, and the effect is to improve planning speed at some cost in physical resolution. Our work presents a generalization of these approaches.

Other uses of metamodules include Prevas *et al.* [**?**] who focused on generating optimal paths with a metamodule structure, Ünsal *et al.* [**?**] who used metamodules to remove constraints on surface motion, and Vassilvitskii *et al.* [**?**] who used the concept to demonstrate feasibility and lower bounds for motion. All of these examples are specific to a particular MR system.

B. Contributions

The main contribution of this paper is a generalized metamodule system and associated planner. This follows from or entails the following contributions:

- Identification of blocking constraints as a form of discrete nonholonomic constraints.
- The development of an ideal metamodule system, *pixel*, which is useful for proving properties about practical metamodule systems and planners that operate on them.
- We show how our metamodule system can be used for such diverse modules as robot arms, expanding cubes, hex planar modules, and cubic packed rolling spheres.
- A distributed, asynchronous planner which we proved complete.
- The separation of concerns between motion planning and resource allocation to make shape transformation tractable in metamorphic systems.

II. Reconfiguration and Motion Constraints

A. Nonholonomic constraints for discrete systems

A continuous system can be described by a configuration q and a velocity $\dot{q}$. Such a system has a nonholonomic constraint if it possesses a constraint on q and $\dot{q}$ that cannot be reduced (integrated) to a constraint on q alone. A consequence of a nonholonomic constraint is that any continuous heuristic function $f(q, q')$ for computing the distance between two configurations can wildly underestimate the actual distance because the velocity constraints can prohibit motions from q to an infinitesimally close neighbor $q + \delta q$, i.e., their true distance is very large, even though $f(q, q + \delta q)$ is small by definition [**?**].

In discrete systems, velocity is represented by atomic movements. A *discrete nonholonomic* constraint is a constraint on configurations and atomic movements that cannot be reduced to a constraint on configurations alone. As in the continuous system, the consequence of a discrete nonholonomic constraint is that configurations that are adjacent to each other can have a prohibitively high true distance in the reconfiguration graph, i.e., a distance that no heuristic can accurately estimate.

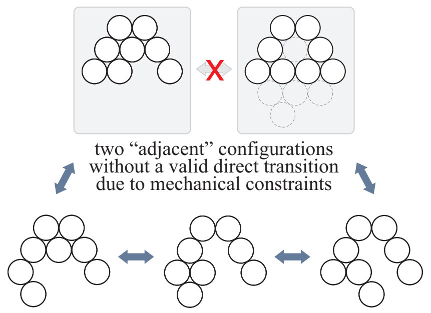

two "adjacent" configurations
without a valid direct transition
due to mechanical constraints

Fig. 2. **The reconfiguration graph as a state-space constraint:** The hex-packed system from Fig. **??**, here showing that configurations adjacent in state-space (boxes) can be substantially further apart in reconfiguration space due to blocking constraints. Since this constraint separates two legal, state-space-adjacent configurations, it is nonholonomic. Dotted circles in the top right configuration must be empty and available for modules to move through during this four-step reconfiguration.

B. Configuration Space and the Reconfiguration Graph

The *configuration space* of a metamorphic system is the set of all possible distinct configurations of that system. For a given module design with particular motion capabilities there is also a *reconfiguration graph*, whose vertices are distinct configurations and whose edges connect (minimal) transitions from one configuration to another. A motion planner must often consider the distance between two configurations—ideally measured in terms of the shortest path on the reconfiguration graph. Because the reconfiguration graph is intractably large for any reasonable size metamorphic system, planners resort to heuristics to estimate these distances.

Fig. **??** is an example of a nonholonomic constraint illustrated by a part of a reconfiguration graph. The two boxed states differ by the motion of one module to a neighboring lattice cell and would be considered directly adjacent to one another by, for instance, an LSAP heuristic [**?**] based on module locations. However, due to mechanical blocking constraints, four transitions would be needed to go from the first boxed configuration to the second. This is an example of a discrete nonholonomic constraint.

C. Local and Non-Local Constraints

A useful formalism for reasoning about certain constraints was introduced by Abrams and Ghrist [**?**], who model local constraints using a *catalog of generators*. For our purposes, the important feature of each generator is that it contains an unordered pair of local configurations (where a local configuration is, roughly, a template for a small arrangement of modules) and all of the information required to transition from one state to the other by means of an *action*. For example, Fig. **??**(a) displays one four-module generator for hex planar modules. The local constraint expressed by this generator is that the action of changing configuration AC to BC requires the absence of a module at D.

We extend this formalism with the notion of a *non-local constraint*, which is a predicate on a configuration that determines if the configuration is admissible or inadmissible. In contrast to local constraints, non-local constraints may consider any part of the system in order to determine admissibility. Such constraints can, for example, be used to maintain global properties such as connectivity and physical stability. Given a particular configuration of the system, all possible next configurations can be found by applying each generator at each admissible location, and then filtering the resulting set of configurations through all non-local constraints.

Note that non-local constraints are holonomic as they reduce legal configurations, not legal transitions. Such constraints can make the planning problem easier, since they reduce the search space. However, for large ensembles, checking configurations for admissibility with respect to non-local constraints can be prohibitively expensive.

This extended formalism is used to evaluate whether an *action* is legal. Even when a direct movement to a configuration-space-adjacent configuration is not legal, the configuration may be reachable via a series of legal actions. Constraints in this formalism are holonomic exactly when each adjacent state-space configuration is either reachable by applying a single generator to the current state, or is unreachable by the constrained system.

Systems with mechanical blocking constraints (i.e., where one module physically blocks another from entering an empty lattice position) are susceptible to nonholonomic constraints; modular robots generally fall into this category. An example of such a situation is shown in Fig. **??**, and part of the reconfiguration graph for the same situation is given in Fig. **??**.

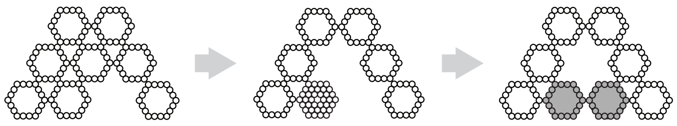


Fig. 3. **Motion plan in a metamodule systems:** The configuration in Fig. **??** again, scaled up and constructed from metamodules. The motion plan using metamodules is much simpler because of their ability to absorb/recreate other metamodules. This plan relies only on one neighbor for execution.

III. Avoiding Nonholonomic Constraints

A. A Purely Holonomic System

Our goal is to devise a holonomic lattice-style metamorphic system. Each lattice point is either empty, or holds a metamodule, which in turn is composed of a group of basic modules. At the metamodule level, the system provides a set of templates for manipulating metamodules and the state of the lattice points, and, below this abstraction, provides preplanned motion sequences for individual modules to implement the templates on metamodules. Holonomic metamodule templates free the system of blocking constraints, thereby ensuring that distributed rules guided by simple, local heuristics are guaranteed not to get stuck.

We first consider an idealized, simple system called the *pixel* system. Here, there is only one defined template: create/destroy, which switches the state of a lattice point between "empty" and "metamodule present." This can be written as:

$$\mathsf{create/destroy} : \square \Leftrightarrow \mathsf{M}$$

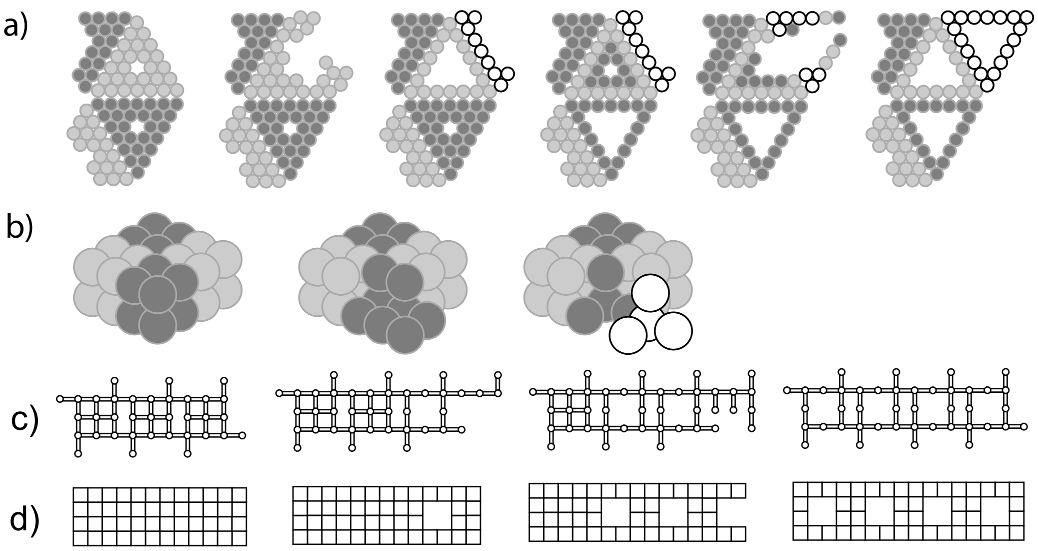


Fig. 4. Example metamodule creation sequences for (a) *general*^{2} on disks, (b) *general*^{1} on spheres, (c) *general-lock*^{3} on modular robot arms and (d) *general-lock*^{3} on expanding cubes. Each system's metamodule destruction sequence runs the creation sequence in reverse.

Like pixels being turned on and off, metamodules can spontaneously appear and disappear anywhere, independent of the presence or absence of metamodules at other lattice points.

The *pixel* system has two non-local constraints: the first, *connectivity*, states that only configurations in which all present modules form a connected group are allowed; the second, *conservation(min, max)*, only allows configurations with no more than *max* and no fewer than *min* metamodules at a time.

This system's constraints are holonomic. This can be seen from the fact that the system's generator allows unconstrained movement through state space (since any module can appear or disappear as long as the non-local constraints are not violated.)

B. Towards a Realizable Holonomic System

The pixel system's metamodule template is not realizable in practice, as it is not mass-conserving. Furthermore, pixel relies heavily on a non-local constraint to enforce connectivity. We can construct an intermediate system that simplifies the latter requirement by changing the create/destroy template to:

$$\text{create/destroy} : \mathsf{M}\square \Leftrightarrow \mathsf{MM}$$

This system, called *intermediate*, requires that creation and deletion of metamodules occur only at lattice spaces adjacent to existing metamodules. This rule makes the system appear more like a compressible fluid that can expand and contract arbitrarily, than like a set of pixels or voxels. Although complete independence no longer holds, this template still requires only local support. Furthermore, given the *connectivity* constraint, any metamodule that is created or destroyed in the *pixel* system is guaranteed to have a neighbor, so any reconfiguration that can be accomplished in that system can also be done in *intermediate*.

To make the system mass-conserving, we define a new system, *general*^{1} with the following templates:

$$\begin{aligned} \text{create/destroy} &: \mathsf{C}\square \Leftrightarrow \mathsf{DD} \\ \text{transfer} &: \mathsf{CD} \Leftrightarrow \mathsf{DC} \end{aligned}$$

Here, metamodules can be in two different states: in state D, the metamodule is composed of a minimum number of modules, and has internal room to hold additional modules; and in state C, the extra space is filled, such that the metamodule is composed of twice as many modules. The create/destroy template in now mass conserving, as the number of modules (not metamodules) remains unchanged. An extra template to transfer extra modules (resources) between adjacent metamodules without changing the number of metamodules is also introduced.

Assuming mass of 1 and 2 for D and C, respectively, the number of metamodules during reconfiguration in *general*^{1} can range from $m/2$ to m, where m is the total mass of a given starting configuration. Given the *conservation(min, max)* global constraint, assuming $min = m/2$ and $max = m$, any reconfiguration that can be accomplished in *intermediate* can also be accomplished in *general*^{1}, assuming appropriate application of the resource transfer template. In contrast to *intermediate* and *pixel*, this system can be physically realized. Fig. **??** shows an example of a 2-dimensional *general*^{1} metamodule system applied to the same reconfiguration problem as in Fig. **??**. Here, the system does not suffer from blocking, nonholonomic constraints, and reconfiguration is a straightforward task. An example of a 3D *general*^{1} metamodule system is shown in Fig. **??**(b).

C. A Generalized Holonomic System

Although *general*^{1} is realizable, the mass conserving property in conjunction with the single-step create/destroy template may impose unwieldy requirements on metamodule design. In particular, a metamodule must have sufficient internal volume to hold all of the modules required to create a second metamodule. As this may not be desirable, or even possible, for some systems, we introduce *general^n*, a generalization of *general^1* that uses extra modules from n metamodules to create a new one. Here, metamodules are constructed/destroyed gradually over multiple steps. This requires n different create/destroy templates to handle all degrees of completion in a partial metamodule, and multiple applications of the transfer rule to deliver/remove all of the resources. The templates for *general^n* are as follows:

$$\begin{aligned}
\mathsf{create/destroy}_1 &: \mathsf{C}\square &&\Leftrightarrow \mathsf{DB}_1\\
\mathsf{create/destroy}_i &: \mathsf{CB}_{i-1} &&\Leftrightarrow \mathsf{DB}_i\\
\mathsf{create/destroy}_n &: \mathsf{CB}_{n-1} &&\Leftrightarrow \mathsf{DD}\\
\mathsf{transfer} &: \mathsf{CD} &&\Leftrightarrow \mathsf{DC}
\end{aligned}$$

Here, additional states $\mathsf{B}_1 \ldots \mathsf{B}_{n-1}$ represent partially constructed metamodules, such that B_i has a relative mass of i/n. D and C are as in *general^1*, but with mass 1 and $1 + 1/n$, respectively.

An interesting observation is that the addition of an n-step process requires the participation of just one additional metamodule, whose modules can be scavenged to construct new metamodules. An example of a 2-step create/destroy process is shown in Fig. **??**(a). One issue in implementing such a metamodule system is that partially constructed B_i metamodules must connect to all neighboring metamodule lattice positions. This ensures that the B_i are functionally equivalent to all neighboring metamodules, and thereby, consistent with our rule set. Unfortunately, this greatly restricts the metamodule designs that will work.

To overcome this, we introduce the notion of locking at the metamodule level: starting a creation or destruction sequence locks the partial metamodule and the adjacent complete metamodule together until the sequence completes. The locked pair can only participate in create or destroy templates with the partnered metamodule, although the transfer template is allowed with the other adjacent metamodules. The templates for *general-lock^n* are:

$$\begin{aligned}
\mathsf{create/destroy}_1 &: \mathsf{C}\square &&\Leftrightarrow \mathsf{D} \bowtie \mathsf{B}_1\\
\mathsf{create/destroy}_i &: \mathsf{C} \bowtie \mathsf{B}_{i-1} &&\Leftrightarrow \mathsf{D} \bowtie \mathsf{B}_i\\
\mathsf{create/destroy}_n &: \mathsf{C} \bowtie \mathsf{B}_{n-1} &&\Leftrightarrow \mathsf{DD}\\
\mathsf{transfer} &: \mathsf{CD} &&\Leftrightarrow \mathsf{DC}
\end{aligned}$$

Here, "$\bowtie$" represents the pairwise lock between the metamodules. This system now allows a much broader range of implementations, where a B_i metamodule only needs to maintain connectivity with the adjacent module that initiated the creation or destruction process. One potential concern is that multiple locked metamodules performing simultaneous creation sequences may result in a resource-starved livelock situation. However, given the *conservation(min,max)* global constraint, any parallel templates that could have succeeded under *pixel* would have sufficient resources to complete under *general-lock^n*. Examples of metamodule systems implementing *general-lock^n* are shown in Fig. **??**.

To summarize, we described the construction of a general and realizable metamodule system that arose from the desire to maintain holonomic constraints in the system. While several other researchers [**?**], [**?**] have suggested metamodules for specific systems, we believe that the rule set for *general^n* can serve as a blueprint for metamodule construction on any modular robotic system. We must admit that creating the metamodules is not easy; it requires domain knowledge of the underlying modular robotic system. However, it provides a unifying grammar that enables the construction of shape planning algorithms capable of working on any hardware platform that implements *general^n*.

IV. Shape Planning

In this section, we present a distributed asynchronous algorithm for shape reconfiguration, the planner, that runs on top of our metamodule abstraction. Given a starting configuration of metamodules, the planner controls the creation and deletion of metamodules to reconfigure the ensemble into a target shape. The planner preserves the global connectivity property, but does not directly address the *conservation(min,max)* constraint. However, it does provide provable guarantees of completeness: if there exists a globally connected plan to achieve a target shape, it will be found. Although the algorithm is written based on an *intermediate* system, we show how to extend it to run on realizable *general^n* and general-lock^n systems.

A. The planning algorithm

The planner, based on the *intermediate* system, produces a sequence of rearrangements to reach a target shape, $\mathcal{T}$, while maintaining global connectivity. Initially, all metamodules are in a known start shape. We assume each metamodule is aware of its coordinates in the lattice (e.g., by using techniques from [**?**]) and knows the configuration of the target shape.[1] Metamodules are only allowed to communicate with their lattice neighbors. The planner incrementally deletes metamodules that are not in the target shape and creates metamodules at empty spaces in the target shape. While creating metamodules cannot break global connectivity, deleting metamodules can. The planner ensures that deletion does not affect global connectivity by generating logical trees that connect each metamodule to be deleted to the sections that will be preserved inductively through its parent in the tree. When deletion is limited to the leaves of the trees (i.e., the metamodules that do not have children), all other metamodules will remain connected.

To distribute the work of generating and modifying these logical trees, each metamodule stores and communicates a variable, which we term its *label*, to its lattice neighbors. Our planner uses three label values—U (undecided), P (path), and F (final)—to denote metamodules that are not part of a tree, those that are in a tree, and those that are in the target shape, respectively. The algorithm proceeds to achieve $\mathcal{T}$ with every module in state F denoted $\mathcal{T}_{\mathsf{F}}$.

[1]In this work, we do not consider how $\mathcal{T}$ is represented, and only assume that there is some function that indicates whether a given coordinate is in the target shape. Efficient shape representation is beyond the scope of this paper.

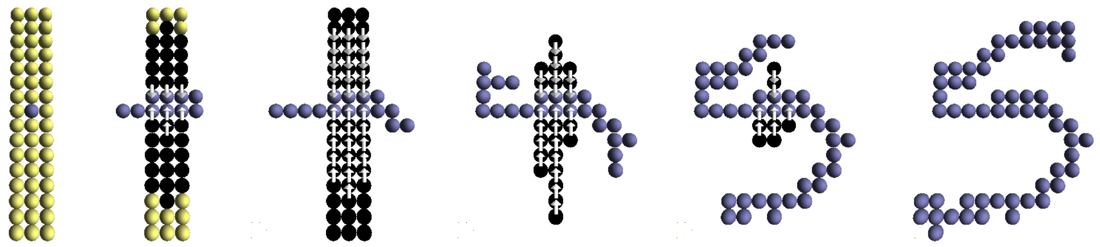


Fig. 5. The planning algorithm running. Metamodules are created in the target shape and deleted outside the target shape; logical trees are formed to make sure deletion will not disconnect the group. The U start state is indicated by the white metamodules (spheres), the F final by the gray (blue) metamodules, and P state by the black metamodules. Trees are indicated by white arrows among the P state metamodules.

The plan starts with an ensemble, $\mathcal{E}$, consisting of a set of metamodules in the start shape. Exactly one seed metamodule in the intersection of $\mathcal{E}$ and $\mathcal{T}$, is labeled F, while all of the others are labeled U. If it is necessary to make the distinction, we mark the labels of metamodules in $\mathcal{T}$ with (ˆ), and of those not in $\mathcal{T}$ with (ˇ). An F-labeled metamodule recruits every neighbor in $\mathcal{T}$ to also become F. It marks every neighbor in state $\check{\mathsf{U}}$ as a candidate for removal called P. These nodes recursively recruit other U metamodules. Every P is connected to its parent (the metamodule that recruited it to state P); the connection from parent to child is indicated with $\rightarrow$. As long as the sequence of parent-child connections remains unbroken, the P metamodules will remain connected to the goal shape. Eventually, the P trees will have no further space to expand, at which point, the leaves (metamodules with no children) can be safely deleted without loss of connectivity. Fig. **??** shows snapshots from an execution sequence of this planner.

The above plan reconfigures $\mathcal{E} \Rightarrow \mathcal{E}'$ when one of these transitions occurs:

RELABELING RULES:

FINALIZATION	$\mathsf{F}\hat{\mathsf{U}}$	$\Rightarrow$	FF
PATH FINALIZATION	$\mathsf{F}\hat{\mathsf{P}}$	$\Rightarrow$	FF
PATH CREATION	$\mathsf{F}\check{\mathsf{U}}$	$\Rightarrow$	$\mathsf{F}\rightarrow\mathsf{P}$
PATH PROPAGATION	PU	$\Rightarrow$	$\mathsf{P}\rightarrow\mathsf{P}$

MOTION RULES:

CREATION	$\mathsf{F}\hat{\square}$	$\Rightarrow$	FF
DELETION	P	$\Rightarrow$	$\square$,

$$\text{if } \nexists\, \mathsf{U} \in \mathrm{nbr}(\mathsf{P}) \text{ and } \nexists\, \mathsf{P}' \in \mathrm{nbr}(\mathsf{P}) : \mathsf{P}\rightarrow\mathsf{P}'$$

where 'nbr' returns the neighbors of a metamodule. Note that only CREATION and DELETION produce actual module motion; all of the other rules, collectively called RELABELING RULES, merely update labels.

B. Resource allocation

The planner as described above runs directly on a *intermediate* abstraction. To run the planner on an *general*^{n} system it is necessary to include an additional component called the *resource allocator*. The resource allocator shifts resources around (moves C and D states, as defined in §**??** through multiple invocations of the transfer operation) so creations and deletions indicated by the planner can occur. Since there is a global constraint on the total number of C and D labels, a good resource allocator must distribute them well, sending the D to regions of anticipated deletion and the C to regions of anticipated creation. A simple, highly suboptimal allocator is a randomizer which transports resources by randomly switching adjacent C and D labels.

Note that except when a purposely malicious resource allocator is used, the algorithm is provably complete as long as the start shape contains a viable number of resources for the target shape. This works because the planner permits creation and deletion in parallel, preventing intermediate states from becoming stuck due to resource constraints. It is also worth noting that while any resource allocator results in eventual completion, the better the allocator, the faster the time to completion.

We have implemented both the basic planner for *intermediate* systems, and a version with a randomized resource allocator suitable for *general*[1] systems. The results of using these in shape change experiments are presented in §**??**.

C. Proof of Correctness

Theorem 1 (Correctness): Running the shape change algorithm on $\mathcal{E}$ containing a single seed will result in $\mathcal{T}_{\mathsf{F}}$ being reached while maintaining connectivity provided $\mathcal{E}$ is connected, $\mathcal{T}$ is connected, and there exists some configuration of metamodules in the target shape with the same mass as $\mathcal{E}$.

The reconfiguration planner is correct if, for a given starting configuration and target shape, $\mathcal{T}$, it produces a plan that successfully transforms the start shape into the target shape in finite time and maintains connectivity provided the starting configuration contains adequate resources to produce the target shape. The proof of Theorem **??** follows from the lemmas defined below. The proof has been written in Twelf[**?**], a language for proving properties of deductive systems. The full Twelf proof is available in [**?**] and a brief sketch is included below.

The proof centers around the existence and maintenance of a spanning tree of the metamodule system. This tree is said to be well-formed:

*Definition 1 (*well-formed*):* A spanning tree is said to be well-formed if the following 2 conditions hold

1) If a metamodule has state F and it is not the root then its parent has state F
2) If a metamodule has state P then its parent in the tree is its parent in the algorithm

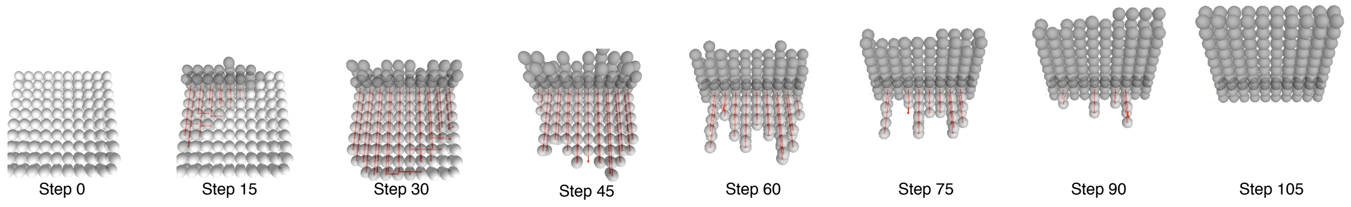


Fig. 6. **Example execution of *general*^{1} planner:** 400 metamodule system transitioning between two rectangular solids.

Corollary 1: If $\mathcal{E}$ has a well-formed spanning tree then $\mathcal{E}$ is connected.

Next, we will define a few mechanisms for comparisons between shapes:

Definition 2 ($|\mathcal{E}|_S$): the number of metamodules in state S in $\mathcal{E}$.

Definition 3 ($\#(\mathcal{E})$): The number of resources contained within $\mathcal{E}$. $\#(\mathcal{E}) = |\mathcal{E}|_\mathsf{D} + (1 + \frac{1}{n})|\mathcal{E}|_\mathsf{C}$ in *general*^{n}.

Definition 4 (κ supports $\mathcal{E}$): We say that κ supports $\mathcal{E}$ if there exists a distribution of C and D resources in $\mathcal{E}$ such that $\#(\mathcal{E}) = \kappa$.

Definition 5 ($<_\mathcal{T}$): For a given target shape $\mathcal{T}$, $\mathcal{E} <_\mathcal{T} \mathcal{E}'$ if $|\mathcal{E}|_\mathsf{F} = |\mathcal{E} \cap \mathcal{T}|_\mathsf{F}$ and one of the following hold:

- $|\mathcal{E}|_\mathsf{F} > |\mathcal{E}'|_\mathsf{F}$
- $|\mathcal{E}|_\mathsf{F} = |\mathcal{E}'|_\mathsf{F}$ and $|\mathcal{E}|_\mathsf{U} < |\mathcal{E}'|_\mathsf{U}$
- $|\mathcal{E}|_\mathsf{F} = |\mathcal{E}'|_\mathsf{F}$ and $|\mathcal{E}|_\mathsf{U} = |\mathcal{E}'|_\mathsf{U}$ and $|\mathcal{E}|_\mathsf{P} < |\mathcal{E}'|_\mathsf{P}$

By this definition, $\mathcal{E} <_\mathcal{T} \mathcal{E}'$ if $\mathcal{E}$ is "closer" to $\mathcal{T}_\mathsf{F}$. Observe that if $\mathcal{E}$ is closer to $\mathcal{T}$ than $\mathcal{E}$' then $\mathcal{E}$ contains more metamodules in the target shape, or else it contains fewer metamodules in the start state which have made no progress or else it contains fewer metamodules which need to be deleted. As a corollary, there is no shape $\mathcal{E}$ such that $\mathcal{E} <_\mathcal{T} \mathcal{T}_\mathsf{F}$. This is because no configuration can have a greater number of F metamodules in $\mathcal{T}$, fewer U metamodules, or fewer P metamodules than $\mathcal{T}_f$.

Using these definitions, the proof of correctness follows simply from 3 lemmas:

Lemma 1 (Safety): If $\mathcal{E}$ has a well-formed spanning tree and $\mathcal{E} \Rightarrow \mathcal{E}'$ then $\mathcal{E}$' has a well-formed spanning tree.

Lemma 2 (Progress): If $\mathcal{E}$ has a well-formed spanning tree and $\#(\mathcal{E})$ supports $\mathcal{T}$ then $\mathcal{E}$ is $\mathcal{T}$ or $\exists \mathcal{E}'$ such that $\mathcal{E} \Rightarrow \mathcal{E}'$ and $\#(\mathcal{E}) = \#(\mathcal{E}')$.

Lemma 3 (Termination): If $\mathcal{E} \Rightarrow \mathcal{E}'$ then $\mathcal{E}' <_\mathcal{T} \mathcal{E}$.

Proof Sketch of Lemma **??**: This proof proceeds by case analysis on the rules of the algorithm, showing that after each rule is applied, a new well-formed spanning tree can be constructed based on the old one.

Proof Sketch of Lemma **??**: This proof proceeds by analysis on the states of the modules. If there is are modules in state U or P, then it is possible to find a rule that can be applied to affect one of these modules. Otherwise, either the target has been achieved or a new module can be created.

Proof Sketch of Lemma **??**: This proof proceeds by case analysis on the rules.

Proof Sketch of Theorem **??**: Observe that a spanning tree rooted at the seed is well-formed. We inductively apply Lemma **??**, Lemma **??**, and Lemma **??** to show that if $\mathcal{T}_\mathsf{F}$ has not yet been reached then $\mathcal{E} \Rightarrow \mathcal{E}'$ where $\mathcal{E}$' still has a well-formed spanning tree, is connected, and is "closer" to the target shape.

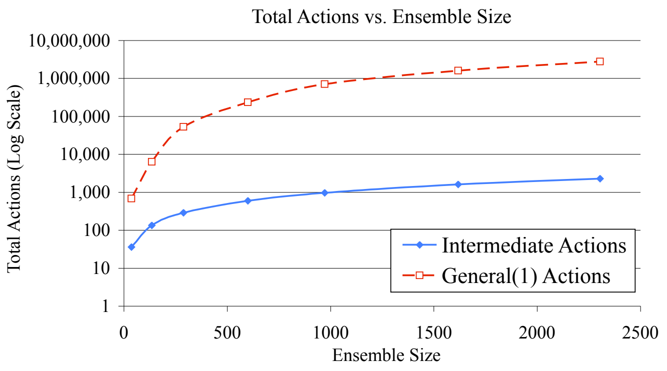


Fig. 7. **Total required actions of *intermediate* and *general*^{1} vs. ensemble scaling:** The total number of creations, deletions, and (in the case of *general*^{1}) *transfer*s required during shape change. Note that the y-axis is log-scale.

V. Implementation

To verify that planning using our metamodule formalism is tractable, we implemented the planner under *intermediate* and *general*^{1} in DPRSim [**?**], a simulator for distributed modular robotic applications. Using the high-level language LDP [**?**], we were able to implement the planner in under 15 lines of code. For *general*^{1}, we included a naive, randomized resource management policy (Fig. **??**). We tested the performance of the algorithm for shape change from a block with aspect ratio 6x3x2 to a 3x3x4 block. We scaled the size of the ensemble from 36 to 2304 metamodules, maintaining the aspect ratio. This allowed us to directly compare the relative work done by the creation and deletion sub phases of the algorithm.

As shown in Fig. **??**, the total number of actions required by *intermediate* is linear in the size of the ensemble. The *intermediate* implementation also provides a lower limit on the number of actions required by any implementable system, as *intermediate* ignores the locality of resources, and is able to move them instantaneously within the ensemble. This is observable in the behavior of *general*^{1}, which performs the same number of creations and deletions as *intermediate*, but performs from one to three orders of magnitude more actions, in the form of random resource transfers. These random resource transfers are the primary source of inefficiency in *general*^{1}.

By examining the time to completion of the creation and deletion portions of the algorithm in Fig. **??**, we can see that the lag in deletion is the primary factor in *general*^{1}s significantly longer completion times. We attribute this to the

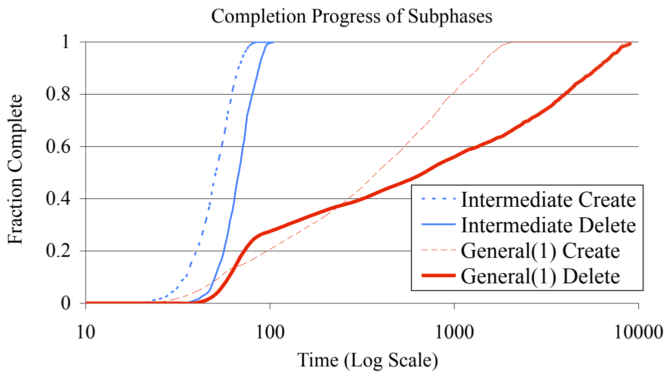


Fig. 8. **Completion of create and delete sub phases in *intermediate* and *general*^{1} over time:** Relative completion percentages of creation and deletion in a 2304 metamodule ensemble over time. Note that time is log-scale.

need to have two adjacent destroyers in order to proceed with deletion, whereas creation requires only a single creator and an open location. As the deletion depletes the local population of destroyers, and random *transfers* diffuse new destroyers very slowly, the deletion process is a significant bottleneck. This points to the need for further optimization of the deletion phase in order to improve the overall efficiency of *general*^{1}.

VI. Conclusions

The difficult problem of shape planning in a modular robotic system with nonholonomic motion constraints has in the past been dealt with in system-specific ways. We have devised a general approach to efficient shape planning by introducing a metamodule abstraction, the *pixel* system, that eliminates nonholonomic constraints. We extend this ideal system to the *general*^{n} abstraction that can be physically realized, yet continues to ensure ease of planning.

The effectiveness of our metamodule abstraction has been shown with a very simple, yet effective planner, that maintains global connectivity and is provably complete. A great benefit of our approach is that the planner is not tied to a particular modular robot, and can be used on any hardware on which a *general*^{n} metamodule abstraction can be devised. The rule set we have presented for *general*^{n} can serve as a guideline for metamodule construction on any modular robotic system.

A further contribution of our approach is the clean separation of concerns between the shape planning and resource allocation. A planner need not concern itself with the mechanics of transporting resources around the ensemble, but only indicate (implicitly) where they are needed by attempting to create or destroy metamodules. Likewise, the resource manger need not be aware of the planning sequence. The simple planner we have described works acceptably with a naive, randomized resource manager. Future enhancements to performance may be achieved through independent efforts in optimizing the planning algorithm and by implementing a more effective resource manager.

Acknowledgements

This work was partially funded by the National Science Foundation (NSF) under grant no. CNS-0428738, Intel Research Pittsburgh, and Carnegie Mellon University. The authors would like to thank the members of the Claytronics research group.