# On Embeddability of Modular Robot Designs

Yannis Mantzouratos*†, Tarik Tosun*‡, Sanjeev Khanna†, Mark Yim‡

***Abstract*—We address the problem of detecting embeddability of modular robots: namely, to decide automatically whether a given modular robot design can simulate the functionality of a seemingly different design. To that end, we introduce a novel graph representation for modular robots and formalize the notion of embedding through topological and kinematic conditions. Based on that, we develop an algorithm that decides embeddability when the two involved designs have tree topologies. Our algorithm performs two passes and involves dynamic programming and maximum cardinality matching. We demonstrate our approach on real modular robots and show that we can detect embeddability of complex designs efficiently.**

## I. Introduction

Modular reconfigurable robot systems have been studied for several decades [22], [17]. These systems have demonstrated the ability to achieve a wide range of tasks like walking, rolling, climbing over obstacles, grasping and carrying items for various applications, including search and rescue [21] and waiting restaurant tables [1].

One of the most interesting aspects of modular reconfigurable robots is the ability to transform into different shapes to adapt to needed tasks. Techniques to automatically determine which shapes and configurations can accomplish a task would make these systems more powerful. Most tasks are compositions of many subtasks: for example, an assembly task could be composed of many pick-and-place operations.

We refer to the automated, generative design of a modular robot from a task specification as *design synthesis*. In a generative system, it would be useful to build systems hierarchically, building subgroups of modules that achieve subtasks. As a first step towards design synthesis, we consider the following problem: can we efficiently determine if a subgroup of modules configured for a kinematic task can be realized in a larger group of modules configured for another kinematic task? For example, given a subgroup of modules that can function as a planar arm, is there a set of modules in a larger configuration that could serve the same function? We call this problem the *design embedding* problem.

In this paper, we provide a formal definition for design embedding via topological and kinematic conditions, and a poly-time algorithm using dynamic programming and matching to efficiently detect when one design can be embedded in another design. The algorithm is intended to be run offline on a central computer. Information about embeddability can then be used to make decisions about the designs. For example, Section VII discusses how kinematic behaviors can be translated from one design to another design that embeds it.

## II. Related Work

Related approaches with modular robots in the past have included narrower optimization of specific kinematic linkages for manipulation problems [3], [19] as well as a selection approach, choosing the most appropriate configuration for a task from a given set of configurations [13]. In these cases, the robot would sense the environment for features and select the most appropriate configuration from among a small set to reach a goal in a locomotion task. Behaviors have also been automatically generated by identifying functional substructures (e.g. knees) in modular robot designs [2].

A more general approach would look at configurations on a finer-grain scale. Since the system is already modular, analyzing the capabilities of assemblages by varying modules is natural. However, the number of possible arrangements of modules grows exponentially with the number of available modules, which makes the selection approach intractable. For very simple tasks such as locomotion in a line, fine-grain generative approaches have been achieved using evolutionary approaches [10].

Design synthesis has been studied in the context of automated machine design [7], [18]. However, in the most general sense this requires an understanding of the space of all tasks, and the relationships between module composition and their interaction with the environment, which is very broad.

In this work, we use graph representations of modular robots. Existing work in graph representations of modular robots includes recognizing if two full configurations are the same [14], identifying graph automorphisms [12], and recognizing identical substructures for efficient

* Y. Mantzouratos and T. Tosun contributed equally to this work.
† Dept. of Computer and Information Science
‡ GRASP Lab and Dept. of Mech. Eng. and Appl. Mechanics
School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia PA, USA. Contact: mantzouratos@gmail.com, tarikt@grasp.upenn.edu, sanjeev@cis.upenn.edu, yim@grasp.upenn.edu.

reconfiguration [11]. Our work distinguishes itself by including task implications on configurations, defining conditions to replicate the capabilities of a design by replicating its structure. To our knowledge, it is the first representation that captures the full kinematic structure of a modular robot design; in fact it can represent any acyclic kinematic structure composed of revolute joints.

## III. Preliminaries

This section provides the basic graph-theoretical concepts and definitions that are used throughout the paper; a more elaborate exposition can be found at [6].

Let $G\,(V, E)$ denote an undirected graph, where $V$ is a set of nodes, and $E \subseteq V \times V$ is a set of undirected edges. Given a subset $V' \subseteq V$ of the vertices, the subgraph induced by $V'$ is given by $(V', \{(u, v) \in E \mid u, v \in V'\})$.

A *simple path* $v_1 \rightsquigarrow v_k = (v_1, v_2, \ldots, v_k)$ in $G$ is a sequence of distinct nodes in $V$ such that for consecutive nodes $v_i, v_j$ in the path, $(v_i, v_j) \in E$. The *length* of a path is just the number of edges it contains and the *distance* between two nodes $u, v \in V$ in $G$, denoted by $\delta_G(u, v)$, is the minimum length of a path from $u$ to $v$. By convention, $\delta_G(u, u) = 0$ and $\delta_G(u, v) = \infty$ when such a path does not exist.

$G$ is called a *tree* if each pair of nodes is connected by exactly one simple path. When a tree is rooted at a node $\tau \in V$, ancestors and descendants of $u \in V$ are defined as follows. Any $v \in V$ such that $\delta_G(\tau, v) < \delta_G(\tau, u)$ and path $v \rightsquigarrow u$ does not involve $\tau$ is called an *ancestor* of $u$ with respect to $\tau$. Similarly, any $v \in V$ such that $\delta_G(\tau, v) > \delta_G(\tau, u)$ and path $v \rightsquigarrow u$ does not involve $\tau$ is called a *descendant* of $u$ with respect to $\tau$. We denote the set of descendants of a node $u$ with respect to $\tau$ as $\operatorname{desc}(u, \tau) \subseteq V$. We will write $G[u \downarrow \tau]$ to denote the subtree of $G$ that is rooted at $u$ and contains exactly $\operatorname{desc}(u, \tau)$, i.e., $G[\{u\} \cup \operatorname{desc}(u, \tau)]$.

We denote immediate descendants by $\mathcal{N}(u, \tau) = \{v \in \operatorname{desc}(u, \tau) \mid \delta_G(u, v) = 1\}$ and call them the *children* of $u$ with respect to $\tau$; a node can have multiple children. An immediate ancestor is denoted by $\mathcal{P}(u, \tau) = \{v \in V \setminus \operatorname{desc}(u, \tau) \mid \delta_G(u, v) = 1\}$ and is called the *parent* of $u$ with respect to $\tau$. Each node has one parent, except $\tau$ which has none. We can naturally extend the above to *child* and *parent edges*, which will be denoted by $\mathcal{N}^e(u, \tau)$ and $\mathcal{P}^e(u, \tau)$ respectively.

Finally, a *rigid body* is a set of points capable of rotation and translation in Euclidean space. Each rigid body is defined by a *body frame* and *origin*. Reference frames are denoted with an uppercase calligraphic letter (e.g. $\mathcal{W}$), and vectors in boldface (e.g. $\mathbf{v}$). The position of point $p$ relative point $o$ in frame $\mathcal{W}$ will be written ${}^{\mathcal{W}}\mathbf{r}_{p/o} \in \mathbb{R}^3$. The orientation of frame $\mathcal{B}$ relative frame $\mathcal{W}$ will be written ${}^{\mathcal{W}}R^{\mathcal{B}} \in SO(3)$.

## IV. Topological Embedding

### A. Definitions and Statement of Main Result

We now formally introduce the graph representation of modular robotic designs that we will use throughout our discussion of topology, and present the notion of topological design embedding.

**Definition 1.** (Unit Block). *A unit block $B = \langle \phi \rangle$ is an elementary rigid body capable of implementing a prespecified set of built-in functionalities $\phi \in \mathcal{F}$.*

Built-in functionality is independent of topology; e.g., consider a block equipped with sensors, a processor unit or a battery. We define a partial order on unit blocks on a functional basis: $B_1 \preceq B_2$ if and only if $\phi_1 \subseteq \phi_2$.

**Definition 2.** (Modular Robot Design). *Given a set of unit blocks $\mathbb{B}$, a robot design $D = \langle G(V, E), \beta \rangle$ defined on $\mathbb{B}$ is a labelled, undirected graph $G$, where nodes of $G$ correspond to unit blocks through $\beta : V \mapsto \mathbb{B}$, and edges between two nodes $u$ and $v$ represent a revolute joint connecting $\beta(u)$ to $\beta(v)$.*

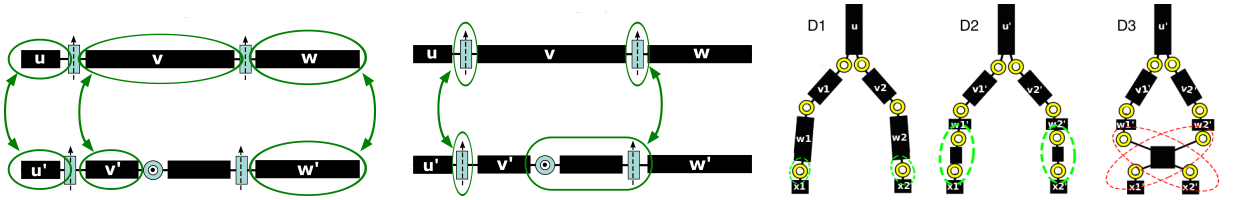
In Section V-B, we will extend the definition of unit blocks to represent rigid bodies, and map edges to revolute joints that provide movement. For now, we postpone discussion of kinematics until the topological algorithm is explained completely.

**Definition 3.** (Design Embedding). *Given two designs $D_1 = \langle G_1(V_1, E_1), \beta_1 \rangle$ and $D_2 = \langle G_2(V_2, E_2), \beta_2 \rangle$ defined on a set of unit blocks $\mathbb{B}$, and an injective mapping $f : V_1 \mapsto V_2$, we say that $D_1$ embeds in $D_2$ with respect to $f$, and write $D_1 \sqsubseteq_f D_2$, if and only if:*

1) *Functionality subsumption: $\forall u \in V_1$, we have $\beta_1(u) \preceq \beta_2(f(u))$.*
2) *Connectivity preservation: $\forall (u, v) \in E_1$, there exists a simple path $\pi_{uv} = f(u) \rightsquigarrow f(v)$ in $G_2$.*
3) *Path disjointness: for any pair of edges with distinct endpoints $(u_1, v_1), (u_2, v_2) \in E_1$, the corresponding paths $\pi_{u_1 v_1}$ and $\pi_{u_2 v_2}$ in $G_2$ are vertex-disjoint. In addition, for any $(u, v_1), (u, v_2) \in E_1$, $\pi_{uv_1}$ and $\pi_{uv_2}$ share only $f(u)$.*

In general, we refer to $D_1$ as the *subdesign* and $D_2$ as the *superdesign*. Where there is no chance of confusion, we omit $f$ and write $D_1 \sqsubseteq D_2$.

Fig. 1 offers the intuition behind the definition. Condition (1) requires every vertex in the subdesign to map to a vertex of equal or superior functionality in the superdesign. Condition (2) preserves the connectivity of the subdesign once it is embedded: nodes which




(a) Condition 1: every node at the top maps to a node of superior functionality at the bottom.

(b) Condition 2: all edges at the top map to distinct simple paths at the bottom.

(c) Condition 3: $D_1 \sqsubseteq D_2$, but $D_1 \not\sqsubseteq D_3$, since the path disjointness condition is violated (red).

Fig. 1: Topological conditions for embedding.

were able to interact through the joints can still do it, albeit maybe through longer paths. Finally, condition (3) ensures that degrees of freedom which are independent in the subdesign remain independent in the superdesign.

From a topological perspective, embeddability is equivalent to whether the subdesign is a topological minor of the superdesign; see [8] and references therein.

We are now ready to state our main result.

**Theorem 1.** *Given two designs $D_1 = \langle G_1(V_1, E_1), \beta_1 \rangle$ and $D_2 = \langle G_2(V_2, E_2), \beta_2 \rangle$ defined over a set of unit blocks $\mathbb{B}$, where $G_1$ and $G_2$ are trees of maximum degree $d$, there exists a deterministic algorithm that decides whether $D_1 \sqsubseteq D_2$ in time $O(|V_1| \cdot |V_2| \cdot d^{2.5})$.*

Note that $d$ is the maximum number of edges incident on any node. For most real robot applications, $d \leq 5$.

### B. Outline of Algorithm

We now present a dynamic programming algorithm that decides whether $D_1 \sqsubseteq D_2$ and if so, produces a mapping $f : V_1 \mapsto V$. We focus on obtaining a yes or no answer rather than the mapping itself; it can be reconstructed the regular dynamic programming way, by keeping track of the option we selected at each step in a separate array and backtracking; see [4] for details.

We will maintain a $|V_1| \times |V_2|$ truth table $T$, where $T[v_1, v_2]$ is true under a specified rooting $\tau_1 \in V_1$, $\tau_2 \in V_2$ if and only if $D_1[v_1 \downarrow \tau_1] \sqsubseteq D_2[v_2 \downarrow \tau_2]$. At the end of the algorithm, $T[\tau_1, \tau_2]$ answers whether $D_1 \sqsubseteq D_2$ under $\tau_1$ and $\tau_2$; if the answer is negative, we repeat the process for a new rooting until we either get a positive answer or we exhaust all possible rootings, in which case we conclude that $D_1 \not\sqsubseteq D_2$.

Initially, all entries are false. We subsequently proceed bottom-up, starting from the leaves of $D_1$ and moving gradually towards $\tau_1$. As the base case, we consider a leaf $v_1 \in V_1$ and check whether it embeds in the leaves of $D_2$, setting the appropriate entries of $T$ accordingly:

$$T[v_1, v_2] = \text{true} \iff \beta_1(v_1) \preceq \beta_2(v_2),$$

for all leaves $v_2 \in V_2$. Intuitively, this corresponds to checking whether a unary design consisting of $v_1$ is embedded in another unary design that contains only $v_2$, in which case the only relevant embedding condition to check is functionality subsumption.

Subsequently, we move towards $\tau_2$ by examining the parents of the leaves of $D_2$; for such a parent $p_2 \in V_2$, we set $T[v_1, p_2]$ to true if $\beta_1(v_1) \preceq \beta_2(p_2)$, or

$$\exists v_2 \in \mathcal{N}(p_2, \tau_2) \text{ such that } T[v_1, v_2] = \text{true}.$$

The intuition is that either $v_1$ is subsumed by $p_2$, or it is subsumed by one of $p_2$'s children. We continue likewise until we complete the $v_1$-th row of $T$ and then repeat for another leaf of $V_1$. At the end, we know exactly which nodes of $D_2$ can host the leaves of $D_1$.

We are now ready to move up one level in $D_1$ as well. Let $p_1 \in V_1$ denote a parent of a leaf from the previous step. Starting from nodes that are at the same height in $D_2$ as $p_1$ is in $D_1$, we set $T[p_1, p_2]$ to true in the following two cases.

The first one, which corresponds to $p_1$ embedding directly in $p_2$, is triggered if $\beta_1(p_1) \preceq \beta_2(p_2)$ and in addition, there exists an assignment $M \subseteq \mathcal{N}(p_1, \tau_1) \times \mathcal{N}(p_2, \tau_2)$ of children of $p_1$ to children of $p_2$ such that

$$\begin{aligned} |M| &= |\mathcal{N}(p_1, \tau_1)|, \text{and} \\ T[v_1, v_2] &= \text{true} \ \ \forall (v_1, v_2) \in M. \end{aligned} \tag{1}$$

Essentially, these conditions make sure that every child of $p_1$ embeds in a unique child of $p_2$.

The second case captures that while $p_1$ might not directly embed in $p_2$ in the way described above, it may still embed in one of $p_2$'s children, i.e.,

$$\exists v_2 \in \mathcal{N}(p_2, \tau_2) \text{ such that } T[p_1, v_2] = \text{true}. \tag{2}$$

Notice that since we are proceeding bottom-up, $T[p_1, v_2]$ is filled before $T[p_1, p_2]$, and therefore our computations are always well defined.

After the $p_1$-th row of $T$ is calculated, we repeat for all nodes at the same height, and then move upwards exactly the same way until we hit $\tau_1$. Essentially, we fill our table by performing a reverse pre-order traversal on $G_1$, where at each step of the traversal we process the nodes of $G_2$ in reverse pre-order as well. Notice that it is not necessary to process all the leaves first, which we did

in our exposition for simplicity, as long as we process a node only after having dealt with all of its descendants.

It only remains to show how to compute $M \subseteq \mathcal{N}(p_1, \tau_1) \times \mathcal{N}(p_2, \tau_2)$ from equation (1): we construct a bipartite graph with $\mathcal{N}(p_1, \tau_1)$ on the left side, $\mathcal{N}(p_2, \tau_2)$ on the right, and an edge $(v_1, v_2)$ if and only if

$$v_1 \in \mathcal{N}(p_1, \tau_1), v_2 \in \mathcal{N}(p_2, \tau_2) \text{ and } T[v_1, v_2] = \text{true}.$$

Intuitively, $v_1$ is connected to $v_2$ if and only if subdesign $D_1[v_1 \downarrow \tau_1]$ embeds in $D_2[v_2 \downarrow \tau_2]$. We subsequently compute a maximum cardinality matching [9]. At this point it is also possible to incorporate arbitrary user input that explicitly disallows embedding of particular nodes.

### C. Formal Analysis

**Lemma 1.** (Algorithm Soundness). *Given designs $D_1$ and $D_2$, if the algorithm accepts then $D_1 \sqsubseteq D_2$.*

*Proof.* Suppose that the algorithm accepts and let $\tau_1$ and $\tau_2$ reflect the roots of $G_1$ and $G_2$ at the time the positive answer was computed, and $f : V_1 \mapsto V_2$ be the suggested mapping. Relabel nodes of both graphs to reflect reverse pre-order, and let $V_1 = \{u_1, u_2, \ldots, \tau_1\}$ and $V_2 = \{v_1, v_2, \ldots, \tau_2\}$. We will use strong induction on $u_i$, $i = 1, \ldots, |V_1|$.

For the base case, let $T[u_1, v_j]$ be true. Since $u_1$ is a leaf, $D_1[u_1 \downarrow \tau_1]$ contains only $u_1$ and the only relevant criterion from definition 3 is functionality subsumption. By construction, $T[u_1, v_j]$ is true if and only if the functionality of $\beta_1(u_1)$ is subsumed by either $\beta_2(v_j)$ or one of its children, and therefore, $D_1[u_1 \downarrow \tau_1] \sqsubseteq D_2[v_j \downarrow \tau_2]$, for any $j = 1, \ldots, |V_2|$.

Now assume that for $i \leq k$, if $T[u_i, v_j]$ is true then $D_1[u_i \downarrow \tau_1] \sqsubseteq D_2[v_j \downarrow \tau_2]$, for all $j$. We are going to show that the same holds for $u_{k+1}$ too. Indeed, fix an arbitrary $v_j$ and suppose that $T[u_{k+1}, v_j] = \text{true}$. If $u_{k+1}$ is a leaf, we are done by the argumentation above, so assume otherwise. Recall that there are two cases which could have forced the entry to be true.

If the first case triggered the truth assignment, then it must be that $\beta_1(u_{k+1}) \preceq \beta_2(v_j)$ and there exists a matching $M$ between $\mathcal{N}(u_{k+1}, \tau_1)$ and $\mathcal{N}(v_j, \tau_2)$ such that (1) holds. Obviously the functionality subsumption criterion of embedding holds. By hypothesis and the fact that $M$ covers all the children of $u_{k+1}$, connectivity is maintained - the subdesigns rooted at the children of $u_{k+1}$ embed in children of $v_j$, and since $f(u_{k+1}) = v_j$, every edge $(u_{k+1}, u') \in E_1$ corresponds to a path utilizing $(v_j, v') \in E_2$, where $(u', v') \in M$. Finally, path disjointness is also maintained. By hypothesis, we only need to care about paths of the form $\pi_{u_{k+1}, u'} = f(u_{k+1}) \rightsquigarrow f(u')$, where $u' \in \mathcal{N}(u_{k+1}, \tau_1)$. Since $f(u')$ maps to a distinct child of $v_j$ by definition of matching (say $f(u') = v'$), and $f(u_{k+1}) = v_j$, it follows that each $\pi_{u_{k+1}, u'}$ starts with a distinct edge of the form $(f(u_{k+1}), f(u')) = (v_j, v') \in E_2$. Paths cannot share any vertex later, since that would imply a cycle in $G_2$, contradicting its tree structure. Consequently, all criteria of embedding are satisfied, and $D_1[u_{k+1} \downarrow \tau_1] \sqsubseteq D_2[v_j \downarrow \tau_2]$.

The second case follows by the same argumentation and a similar (nested) inductive argument on $v_j$. □

**Lemma 2.** (Algorithm Completeness). *Given designs $D_1$ and $D_2$, if $D_1 \sqsubseteq D_2$ then the algorithm accepts.*

*Proof.* Suppose that $D_1 \sqsubseteq_f D_2$ for $f : V_1 \mapsto V_2$. Let $\tau_1$ be an arbitrary node in $V_1$, $\tau_2 = f(\tau_1) \in V_2$ and consider running the algorithm with $\tau_1$ and $\tau_2$ as the roots. As before, relabel the nodes of both graphs to reflect reverse pre-order; we sketch an inductive argument on $V_1$.

As a base case, notice that $D_1[u_1 \downarrow \tau_1]$, which consists only of $u_1$, embeds into some $D_2[v_j \downarrow \tau_2]$ as a direct result of the functionality subsumption criterion, where $f(u_1) = v_j$. Therefore, $T[u_1, v_j]$ is trivially true.

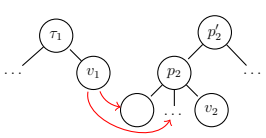
For the inductive step, consider $u_{k+1}$ and let $f(u_{k+1}) = v_j$. By the functionality subsumption criterion, $\beta_1(u_{k+1}) \preceq \beta_2(v_j)$. We only need to show that there exists a matching $M$ satisfying (1) and we are done – equation (2) takes care of propagating the result upwards. By the connectivity preservance and path disjointness criteria, it must be the case that for every $u' \in \mathcal{N}(u_{k+1}, \tau_1)$, there exists a unique, vertex-disjoint path $\pi_{u_{k+1}, u'}$ in $G$. While $f(u')$ might not be in $\mathcal{N}(v_j, \tau_2)$, vertex disjointness implies that each of $\pi_{u_{k+1}, u'}$ passes through a unique, distinct child of $v_j$. By hypothesis then, the respective entries of $T$ are true, and the suggested matching will satisfy (1). □

**Lemma 3.** (Algorithm Runtime). *The algorithm presented above runs in time $O(|V_1| \cdot |V_2|^2 \cdot d^{2.5})$.*

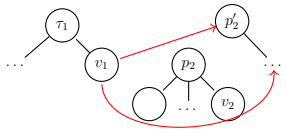
*Proof.* For each rooting $\tau_1 \in V_1$ and $\tau_2 \in V_2$, we fill a table of size $|V_1| \times |V_2|$. Each entry might require solving a matching instance on a bipartite graph of $O(d)$ nodes, which takes time $O(d^{2.5})$ [9]. As a result, the algorithm requires time $O(|V_1| \cdot |V_2| \cdot d^{2.5})$ to check a fixed rooting. Now note that based on the proof of lemma 2, we can fix $\tau_1$ and iterate over $V_2$ for the choice of $\tau_2$ instead of trying all rootings. The claim follows. □

### D. 2-pass Approach

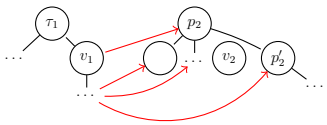
We now improve the runtime by a factor of $|V_2|$ to obtain the claim of Theorem 1. As is, the algorithm needs to try $O(|V_2|)$ rootings to decide embeddability. Since the root $\tau_1 \in V_1$ is fixed, let $T_v$ be the table computed when $D_2$ is rooted at $v \in V_2$. We show here




(a) $v_1$ embeds in a child of $p_2$ other than $v_2$.




(b) $v_1$ embeds in $p_2'$ even if $D_2[p_2 \downarrow p_2']$ is removed.




(c) $v_1$ is subsumed by $p_2$ and every child of $v_1$ embeds either in a child of $p_2$ other than $v_2$, or in $p_2'$ without reusing any parts of the subtree $D_2[p_2 \downarrow p_2']$. When this happens, it is equivalent to re-rooting the tree at $p_2$.

Fig. 2: The three cases of message construction.

that only two passes suffice to compute a table $T^*$ such that $T^*[\tau_1, v] = \text{true}$ iff $T_v[\tau_1, v] = \text{true}, \forall v \in V_2$. It is then not hard to see that $D_1 \sqsubseteq D_2$ iff at least one entry of the $\tau_1$-th row of $T^*$ is true.

During the first pass, we root $D_2$ arbitrarily at $\tau_2 \in V_2$ and compute $T_{\tau_2}$ as before. The second pass involves top-down message passing. In particular, we iterate through $V_2$ in pre-order, starting from $\tau_2$, and each node $p_2 \in V_2$ forwards to every child $v_2 \in \mathcal{N}(p_2, \tau_2)$ a message $\mu(p_2, v_2)$ that is equal to

$$\{v_1 \in V_1 \mid D_1[v_1 \downarrow \tau_1] \sqsubseteq (D_2[p_2 \downarrow p_2] \setminus D_2[v_2 \downarrow p_2])\}.$$

Intuitively, $\mu(p_2, v_2)$ contains all nodes of $V_1$ that can successfully embed in $p_2$ without using the subtree hanging from $v_2$. Given $T_{\tau_2}$ and the respective message from the parent of $p_2$, say $p_2' \in V_2$, we compute $\mu(p_2, v_2)$ by iterating over $V_1$ in arbitrary order and including $v_1 \in V_1$ if any of the following holds (fig. 2):

1) $T_{\tau_2}[v_1, v_2'] = \text{true}$ for a node $v_2' \in \mathcal{N}(p_2, \tau_2) \setminus \{v_2\}$, i.e., $v_1$ embeds in a child of $p_2$ other than $v_2$.
2) $v_1 \in \mu(p_2', p_2)$, i.e., $v_1$ embeds in $p_2'$ even if none of its children are allowed to embed in $D_2[p_2 \downarrow p_2']$.
3) $\beta(v_1) \preceq \beta(p_2)$ and there exists an assignment $M \subseteq \mathcal{N}(v_1, \tau_1) \times [(\mathcal{N}(p_2, \tau_2) \setminus \{v_2\}) \cup \{p_2'\}]$ that satisfies equation (1) with $T_{\tau_2}$ as the truth table, and in addition, if $(v_1', p_2') \in M$, then $v_1' \in \mu(p_2', p_2)$. This means that $v_1$ is subsumed by $p_2$ and every child of $v_1$ embeds either in a child of $p_2$ other than $v_2$, or in the parent of $p_2$ without reusing any parts of the subtree $D_2[p_2 \downarrow p_2']$.

**Lemma 4.** (Message Correctness). *The process described above computes $\mu(p_2, v_2)$ correctly.*

*Proof.* Let $\mu(p_2, v_2)$ denote the message as computed by the algorithm and define $\mu^*(p_2, v_2)$ to be $\{v_1 \in V_1 \mid D_1[v_1 \downarrow \tau_1] \sqsubseteq (D_2[p_2 \downarrow p_2] \setminus D_2[v_2 \downarrow p_2])\}$.

To show that $\mu(p_2, v_2) \subseteq \mu^*(p_2, v_2)$, suppose that $v_1 \in \mu(p_2, v_2)$ and consider what forced this decision:

1) There exists a child $v_2' \neq v_2$ such that $T_{\tau_2}[v_1, v_2'] = \text{true}$. Then, by correctness of the original algorithm, $D_1[v_1 \downarrow \tau_1] \sqsubseteq D_2[v_2' \downarrow \tau_2]$. However, $D_2[v_2' \downarrow \tau_2]$ is a subset of $D_2[p_2 \downarrow p_2]$, since $v_2'$ is still in $\mathcal{N}(p_2, p_2)$, and it does not have any nodes in common with $D_2[v_2 \downarrow p_2]$, otherwise we would have a cycle. It then follows that $v_1 \in \mu^*(p_2, v_2)$.
2) If $v_1 \in \mu(p_2', p_2)$, then by a simple inductive argument we obtain that $v_1 \in \mu^*(p_2', p_2)$, and since $D_2[p_2' \downarrow p_2'] \setminus D_2[p_2 \downarrow p_2']$ is just one branch of $D_2[p_2 \downarrow p_2]$ that does not have anything to do with any other children of $p_2$, the claim follows.
3) This condition is straightforward, since $v_2$, and therefore the subdesign hanging from it, are completely excluded from the matching process.

Opposite direction is similar and we only sketch it. Assume $v_1 \in \mu^*(p_2, v_2)$ and consider where $v_1$ embeds:

1) In a child $v_2'$ of $p_2$ other than $v_2$; but then $D_2[v_2' \downarrow p_2] = D_2[v_2' \downarrow \tau_2]$.
2) In the parent of $p_2$, say $p_2'$, without involving $p_2$. Then, it must be that $v_1 \in \mu^*(p_2', p_2)$ and inductively we get $\mu^*(p_2', p_2) = \mu(p_2', p_2)$.
3) In $p_2$ and $v_2$ is not included in the relevant matching.

All our conditions then follow. □

Similarly, once $\mu(p_2, v_2)$ is passed to $v_2$, we set $T^*[v_1, v_2]$ to true for each $v_1 \in V_1$ that satisfies any of the following conditions:

1) $T_{\tau_2}[v_1, v_2] = \text{true}$.
2) $v_1 \in \mu(p_2, v_2)$.
3) $\beta(v_1) \preceq \beta(v_2)$ and there exists an assignment $M \subseteq \mathcal{N}(v_1, \tau_1) \times (\mathcal{N}(v_2, \tau_2) \cup \{p_2\})$ that satisfies equation (1) with $T_{\tau_2}$ as the truth table, and in addition, if $(v_1', p_2) \in M$, then $v_1' \in \mu(p_2, v_2)$.

A case analysis identical to that of lemma 4 should persuade us that $T^*$ is built as claimed, and this concludes our algorithm and the proof of theorem 1.

## V. Kinematic Admissibility

We now extend our notion of embedding to capture kinematic functionality. As mentioned previously, nodes will represent rigid bodies, and edges will represent revolute joints.

### A. Extending Definitions

We extend the definition of a unit block to represent a rigid body. A unit block $B = \langle o_B, \mathcal{B}, \alpha, \phi \rangle$ is a rigid body with origin $o_B$, reference frame $\mathcal{B}$, attachment points $\alpha$, and built-in functionalities $\phi$. We refer to $o_B$ simply as

$B$ when the meaning is not ambiguous, and adopt the convention that nodes are named with lowercase letters, whereas the rigid bodies they represent are named with the same uppercase letter (*e.g.* node $b$ will represent unit block $B$, with body frame $\mathcal{B}$).

The *attachment points* $\alpha = \{{}^{\mathcal{B}}\mathbf{r}_{\alpha_1/B}, {}^{\mathcal{B}}\mathbf{r}_{\alpha_2/B}, \ldots\}$ are the set of points where $B$ is connected to other unit blocks by revolute joints. As each attachment point is attached to a single revolute joint, we sometimes refer to attachment points by the corresponding joint, *e.g.* if joints $J$ and $K$ attach to $B$ at $\alpha_1$ and $\alpha_2$, we write $\alpha = \{{}^{\mathcal{B}}\mathbf{r}_{J/B}, {}^{\mathcal{B}}\mathbf{r}_{K/B}, \ldots\}$.

Next, we associate with each edge a revolute joint through $\gamma : E \mapsto \mathbb{J}$, similar to $\beta$ for nodes. A *revolute joint* $J = \langle {}^{\mathcal{A}}\boldsymbol{r}_{J/A}, {}^{\mathcal{B}}\boldsymbol{r}_{J/B}, {}^{\mathcal{A}}\hat{\boldsymbol{u}}, {}^{\mathcal{B}}\hat{\boldsymbol{u}}, \theta, {}^{\mathcal{A}}R^{\mathcal{B}}_0 \rangle$ connects a pair of unit blocks ($A$ and $B$) and permits rotation about an axis. $J$ has attachment points ${}^{\mathcal{A}}\boldsymbol{r}_{J/A}$ and ${}^{\mathcal{B}}\boldsymbol{r}_{J/B}$, rotation axis specified by unit vectors ${}^{\mathcal{A}}\hat{\boldsymbol{u}}$ and ${}^{\mathcal{B}}\hat{\boldsymbol{u}}$, joint angle $\theta$, and reference orientation ${}^{\mathcal{A}}R^{\mathcal{B}}_0$. The *joint-angle* $\theta$ of $J$ specifies the amount of rotation about ${}^{\mathcal{A}}\hat{\mathbf{u}}$, relative to the *reference orientation* ${}^{\mathcal{A}}R^{\mathcal{B}}_0 = {}^{\mathcal{A}}R^{\mathcal{B}}(\theta = 0)$. Given a fixed reference orientation, we can find ${}^{\mathcal{A}}R^{\mathcal{B}}$ as a function of $\theta$: ${}^{\mathcal{A}}R^{\mathcal{B}}(\theta) = \exp(\theta[{}^{\mathcal{A}}\hat{\mathbf{u}}]_\times){}^{\mathcal{A}}R^{\mathcal{B}}_0$. Here, $[{}^{\mathcal{A}}\hat{\mathbf{u}}]_\times$ is the cross-product matrix of ${}^{\mathcal{A}}\hat{\mathbf{u}}$.

### B. Kinematic Admissibility

When we say the embedding $D_1 \sqsubseteq_f D_2$ is *kinematically admissible*, we mean that $D_1$ exactly replicates every kinematic DoF present in $D_2$. Intuitively, to verify kinematic admissibility, we must find a configuration (set of joint angles) for $D_2$ such that for each joint in $D_1$, the corresponding joint in $D_2$ exactly matches its position and axis. If we lock the position of all joints $\gamma(E_2 \setminus f^e(E_1))$ (those which do not correspond to joints in $D_1$) while in this configuration, we get a kinematic structure identical to $D_1$. We present a local form of this global requirement by augmenting two of the conditions of Definition 3:

**Definition 4** (Kinematic Admissibility)**.** *Given $f$ such that $D_1 \sqsubseteq_f D_2$, $f$ is kinematically admissible if it satisfies the following two conditions.*

*(1) (Nodes) Let $b' \in V_2$ embed $b \in V_1$. There must exist a position $[{}^{\mathcal{B}}\mathbf{r}_{B'/B}]^*$ and orientation $[{}^{\mathcal{B}}R^{\mathcal{B}'}]^*$ such that for every $J_i \in \gamma(\mathcal{N}^e(b))$ there is a unique $J'_i \in \gamma(\mathcal{N}^e(b'))$ such that:*

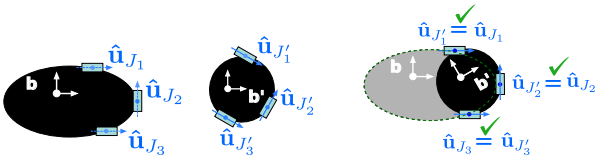
$$ {}^{\mathcal{B}}\mathbf{r}_{J_i/B} = \left[{}^{\mathcal{B}}R^{\mathcal{B}'}\right]^* {}^{\mathcal{B}'}\mathbf{r}_{J'_i/B'} + \left[{}^{\mathcal{B}}\mathbf{r}_{B'/B}\right]^* \quad (3) $$

$$ {}^{\mathcal{B}}\hat{\mathbf{u}}_{J_i} = \left[{}^{\mathcal{B}}R^{\mathcal{B}'}\right]^* {}^{\mathcal{B}'}\hat{\mathbf{u}}_{J'_i} \quad (4) $$

*(2) (Paths) Let path $\pi_{ab} = a' \rightsquigarrow b'$ in $G_2$ embed edge $(a, b) \in E_1$. Let $(a', c') \in \mathcal{N}^e(a')$ be the first edge of $\pi_{ab}$. Let $K' = \gamma((a', c'))$ and $K = \gamma((a, b))$ be aligned; that is, let ${}^{\mathcal{B}}\mathbf{r}_{K'/B} = {}^{\mathcal{B}}\mathbf{r}_{K/B}$ and ${}^{\mathcal{B}}\hat{\mathbf{u}}_{K'} = {}^{\mathcal{B}}\hat{\mathbf{u}}_K$. Let $\Theta(\pi_{ab})$ be the set of angles of all joints on $\pi_{ab}$. There must exist joint angles $[\Theta(\pi_{ab})]^*$ such that:*

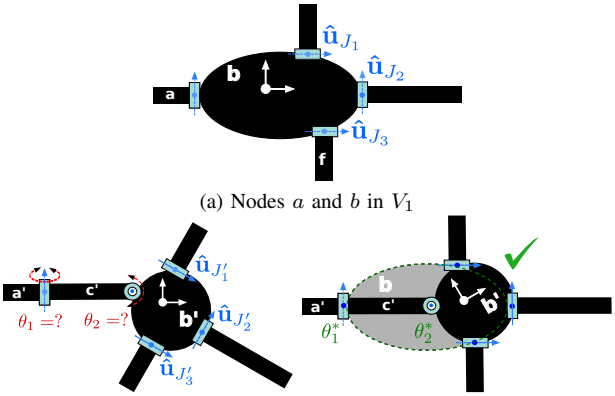
$$ {}^{\mathcal{B}}\mathbf{r}_{B'/B}\left([\Theta(\pi_{ab})]^*\right) = \left[{}^{\mathcal{B}}\mathbf{r}_{B'/B}\right]^* \quad (5) $$

$$ {}^{\mathcal{B}}R^{\mathcal{B}'}\left([\Theta(\pi_{ab})]^*\right) = \left[{}^{\mathcal{B}}R^{\mathcal{B}'}\right]^* \quad (6) $$




(a) Node $b \in V_1$ (b) Node $b' \in V_2$ (c) $[{}^{\mathcal{B}}\mathbf{r}_{B'/B}]^*$ and $[{}^{\mathcal{B}}R^{\mathcal{B}'}]^*$

Fig. 3: Def. 4, 1 (nodes). In fig. (c), a position and orientation have been found in which each child joint of $b$ is aligned with a corresponding child joint of $b'$.




(a) Nodes $a$ and $b$ in $V_1$

(b) Nodes $a'$, $c'$, and $b'$ in $V_2$ (c) Configuration $\Theta^*(\pi_{ab})$ under which condition 2 is satisfied.

Fig. 4: Def. 4, 2 (paths). For path $\pi_{ab} = (a', c', b')$ to embed edge $(a, b)$, there must be angles $[\Theta(\pi_{ab})]^*$ for which ${}^{\mathcal{B}}\mathbf{r}_{B'/B} = [{}^{\mathcal{B}}\mathbf{r}_{B'/B}]^*$ and ${}^{\mathcal{B}}R^{\mathcal{B}'} = [{}^{\mathcal{B}}R^{\mathcal{B}'}]^*$

See Fig. 3 and 4 for an illustration. The condition on nodes ensures that whenever some $b'$ embeds $b$, there is a special position $[{}^{\mathcal{B}}\mathbf{r}_{B'/B}]^*$ and orientation $[{}^{\mathcal{B}}R^{\mathcal{B}'}]^*$ for $b'$ in which some of its child edges match with all child edges of $b$. The condition on paths ensures that there is a configuration for the path connecting $f(a)$ to $f(b)$ which actually allows $b'$ to assume this special position and orientation.

### C. Checking Kinematic Admissibility

To check (1) for a modular robot system with known links and joints, we pre-compute solutions $\left[{}^{\mathcal{B}}\mathbf{r}_{B'/B}\right]^*$ and $[{}^{\mathcal{B}}R^{\mathcal{B}'}]^*$ for each pair of nodes in the system by brute-force. When running the algorithm, they can be

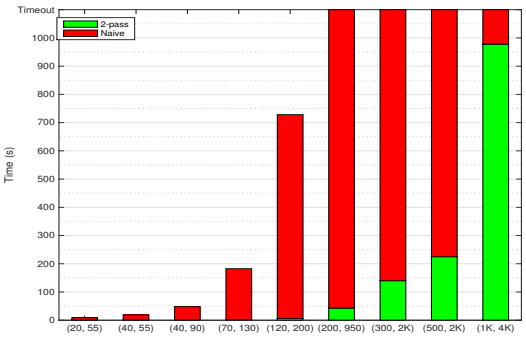



Fig. 5: 2-pass against naive embedding on random trees. $x$-axis is benchmark size as a function of the nodes in the subdesign and the superdesign. Timeout is 2 hours.

quickly found by table-lookup. (2) is checked through inverse kinematics (IK); when an edge maps to a path, we impose the additional restriction that an IK solution for this path must be found which allows its terminating node to reach $[{}^{\mathcal{B}}\mathbf{r}_{B'/B}]^*$ and $[{}^{\mathcal{B}'}R^{\mathcal{B}}]^*$. IK takes time exponential in the number of joints on the path $\pi_{ab}$, so in practice it is the costliest operation in our algorithm.

## VI. Experiments

The algorithms were implemented in Python on top of graph-tool [15] and are available to the community at `modlabupenn.org/embedding`. All experiments were run on a single core of an Intel i7 at 2.4GHz; reported times are the average over ten repetitions.

The first experiment is a topology benchmark: designs are random trees of max degree 5, and each node is assigned one of two functionalities uniformly at random. Since the naive approach is much simpler to implement, is it worthy to adopt 2-pass from a practical viewpoint? Fig. 5 shows that naive quickly becomes infeasible, with 2-pass outperforming it and scaling really well with input size. Even large instances, where designs contain thousands of nodes, are solved in almost 15 minutes.

In another experiment, not plotted due to space limits, subdesigns are complete binary trees where each node is assigned one of two functionalities uniformly at random. Superdesigns are complete binary trees of twice the depth, with nodes at even depth assigned one functionality and nodes at odd depth assigned the other one. There, 2-pass mapped a complex 127 node subdesign to a 16K node superdesign in less than 27 minutes.

Profiling the algorithm, almost 67% of the time is spent in the first pass, and 54% is consumed in generating and solving small matchings. Kinematics are definitely the bottleneck; almost ten minutes are required for designs of 200 and 500 nodes, as compared to less than 30 seconds for topology. For smaller designs with 25 and 80 nodes, kinematic embedding is detected in $< 5$ seconds.

(a) Grasper (b) Walker

Fig. 6: A grasper and a walker built out of SMORES [5].

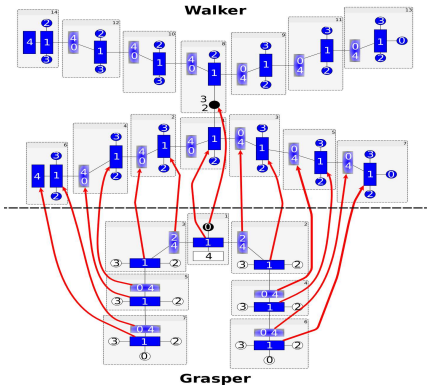


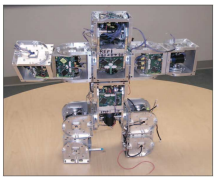

Fig. 7: Walker design on top and grasper design on bottom. Red arrows show the discovered embedding.

## VII. Applications

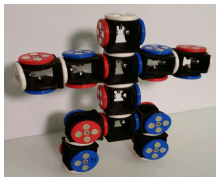
We turn to practical applications of embedding, and discuss how we can perform automatic control translation from the subdesign to the superdesign once a kinematically admissible embedding has been computed.

As a first example, consider the grasper and walker robots pictured in figure 6. Both designs are built out of SMORES modules [5]. Each SMORES module has four DoF: three continuously rotating faces called *turntables* and one central hinge with a $180^o$ range of motion. When two SMORES modules connect, the connected faces become rigidly attached; rather than representing such a connection with an edge, we fuse the faces into a single node which is then considered a member of both modules. Figure 7 shows the underlying designs and the embedding found by our algorithm in under one second.

We are now able to map behaviors from the grasper to the walker. Kinematic behaviors for a modular robot design can be specified by gait tables containing a time-series of joint-angles [20]. Given a gait table for the grasper that produces a desired behavior (like wrapping the arms around an object to immobilize it), we can use the mapping from our algorithm to translate the gait table and achieve the same desired behavior with the walker.



(a) SuperBot subdesign [16].



(b) SMORES superdesign.

Fig. 8: SuperBot Design embeds in SMORES design.

More importantly, the same idea can be extended across different modular robot systems. As an illustration, our algorithm detects that the SuperBot design [16] of fig. 8a embeds in the SMORES design of fig. 8b, and in general, it can be shown that a SuperBot module always embeds in a design of two SMORES modules. Detecting such embeddings automatically and using them to translate behaviors between platforms could save time for researchers who would otherwise try to re-create behaviors manually, especially when working on complex designs where embeddability is not as straightforward.

## VIII. Conclusion and Future Work

We developed and implemented a poly-time algorithm to decide if a given modular robot design can be embedded into another design. The algorithm processes real-life designs in a matter of seconds and scales well with input size. We also formalized the notion of embedding, based on graph representations of modular robots, and highlighted automatic control translation as an application.

In the near future, we will look into handling designs with a small number of cycles and decreasing the runtime of kinematic checking. In the longer term, we will move from detecting embeddability to design synthesis. We believe that our embedding approach is a useful starting point for this line of research, and have obtained promising preliminary results.

## Acknowledgements

The authors would like to acknowledge the support of NSF grant CCF-1138847.

## References

[1] S. Balakirsky, S. Chitta, G. Dimitoglou, J. Gorman, K. Kim, and M. Yim. Robot challenge [competitions]. *Robot. Automat. Mag.*, 19:9–11, 2012.

[2] S. Bonardi, M. Vespignani, R. Moeckel, J. Kieboom, S. Pouya, A. Sproewitz, and A. Ijspeert. Automatic generation of reduced CPG control networks for locomotion of arbitrary modular robot structures. In *Proc. RSS*, 2014.

[3] I.M. Chen and J.W. Burdick. Determining task optimal modular robot assembly configurations. In *Proc. ICRA*, 1995.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Intro. to Algorithms*, chapter 15, pages 394–395. MIT Press, 3rd edition, 2009.

[5] J. Davey, N. Kwok, and M. Yim. Emulating self-reconfigurable robots: Design of the smores system. In *Proc. IROS Conf.*, 2012.

[6] R. Diestel. *Graph Theory*, chapter 1. Springer, 4th edition, 2010.

[7] S. Finger and J. Rinderle. A transf. appr. to mechanical design using bond graph grammer. 1990.

[8] M. Grohe, K. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proc. 43rd STOC*, 2011.

[9] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[10] G. Hornby, H. Lipson, and J.B. Pollack. Evolution of generative design systems for modular physical robots. In *Proc. ICRA*, 2001.

[11] F. Hou and W. Shen. Graph-based optimal reconfiguration planning for self-reconfigurable robots. *Robotics and Autonomous Systems*, 2013.

[12] B. McKay. Nauty user's guide (v2.4). *Computer Science Dept., Australian Nat. Univ.*, 2007.

[13] R. O'Grady, R. Gross, F. Mondada, M. Bonani, and M. Dorigo. Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. In *Proc. 8th ECAL*, 2005.

[14] M. Park, S. Chitta, A. Teichman, and M. Yim. Automatic configuration recognition methods in modular robots. *Int. J. Robot. Research*, 2008.

[15] T. Peixoto. The graph-tool python library. `http://graph-tool.skewed.de/`, 2014.

[16] B. Salemi, M. Moll, and W. Shen. SUPERBOT: Deployable, multifunctional, and modular self-reconfigurable system. In *Proc. IROS.*, 2006.

[17] K. Stoy, D. Brandt, and D. Christensen. *Self-reconfigurable robots: An Intro.* MIT Press, 2010.

[18] K. Ulrich and W. Seering. Synthesis of schematic descriptions in mechanical design. *Res. in Eng. Design*, pages 3–18, 1989.

[19] G. Yang and I.M. Chen. Task-based optimization of modular robot configurations: minimized DoF approach. *Mechanism and machine theory*, 2000.

[20] M. Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford, 1994.

[21] M. Yim, D. Duff, and K. Roufas. Modular reconfigurable robots: an approach to urban S&R. 2000.

[22] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian. Modular self-reconfigurable robot systems. *Robot. Automat. Mag.*, 2007.