Reconfiguration Algorithms for Robotically Manipulatable Structures

Daniel Lobo [1,3], David Alan Hjelle [1], Hod Lipson [1,2]

[1]Sibley School of Mechanical & Aerospace Engineering

[2]Computing and Information Science

Cornell University, Ithaca NY 14853, USA

dah283@cornell.edu

hod.lipson@cornell.edu

[3]Departamento de Lenguajes y Ciencias de la Computación

Universidad de Málaga, Málaga 29071, Spain

dlobo@geb.uma.es

Abstract— *This paper addresses the design and optimization of robotically-reconfigurable structures from an algorithmic point of view. First, we address the algorithmic challenge of searching for a sequence of structural modifications that can transform a given modular structure into a new target structure that serves a different function. The target structure is not explicitly specified, only its desired function, and so the planning algorithm needs to account for both design and the corresponding deconstruction and construction sequence simultaneously. A number of different internal representations are considered and compared. We then demonstrate the algorithm under uncertainty in resource availability, where components may become missing during the build. Finally, we physically implement a robotically manipulatable structure design and manually confirm the reconfiguration process. We suggest that a combination of reconfigurable structures, robust reconfiguration algorithms that function under resource uncertainty, and reconfiguration robots, could open the door to a metabolic process where structures are decomposed and recomposed autonomously to meet varying needs for a variety of applications from infrastructure recovery to space exploration.*

Index Terms—*reconfigurable mechanisms, reconfigurable robotics*

1. Introduction

Biological metabolism is the process by which an organism breaks down food into its constituent modular elements (catabolism) and then uses those raw materials to create new tissue (anabolism). Metabolic processes demonstrate interesting properties that are difficult to replicate in synthetic structures, such as continual reuse of modular elements in new organisms, autonomous disassembly and assembly processes, self repair, continuous adaptation to functional requirements, and robustness to resource fluctuations. Duplicating these properties in a robotic ecology and composing, decomposing, and then recomposing items out of such modular elements could have a wide range of applications, ranging from infrastructure recovery to space exploration. Figure 1 shows a concept image of a robotic ecology metabolizing truss structures in the proposed way.

Machine metabolism is a long-term goal, consisting of multiple challenges. In this paper, we address the first two key challenges: (a) a solution for the algorithmic problem of how to transform an initial structure into an unknown desired target structure that optimizes a known desired new function and (b) how to design a truss such that it is easily robotically manipulatable. Finally we verify both the component design and reconfiguration process by physically testing them in reality. The type of robot(s) that can perform this and its/their motion plans have not been addressed yet, but both the reconfiguration algorithm and the design of the truss have the goal to be performed by robot(s).

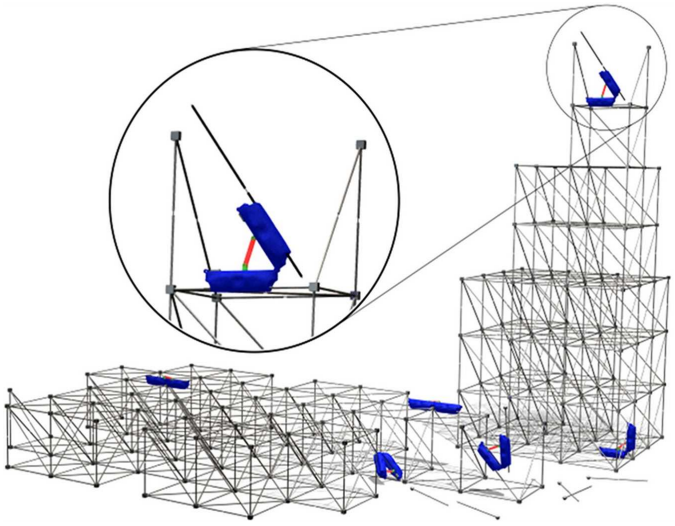


Figure 1. Structural metabolism concept. Artist rendition of several robots performing metabolism on a truss structure. Structural metabolism implements properties of biological metabolism, such as autonomous disassembly and assembly, automated design from encoded requirements, and robustness to resource fluctuations.

2. BACKGROUND AND RELATED WORK

Self-reconfigurable modular robotics traditionally considers systems with homogeneous self-mobile modules that change their own shape by rearranging the connectivity of their parts [1]. This mobility is important in a system intended to reconfigure repeatedly or frequently, such as locomotion over different terrains [2] or playback of dynamic three-dimensional systems [3]. However, in a system intended for the construction of static structures, self-mobility is unnecessary [4]. Additionally, static structures and mobile constructor robots can be applied to a broader range of problems. Although our algorithm resembles in part self-reconfiguration planning algorithms it resolves a problem fundamentally different. The self-reconfiguration planning problem consists in the determination of a sequence of motions that change an initial configuration into a specified goal configuration. Our algorithm solves a substantially different problem that consists in the determination of a sequence of motions that change an initial configuration into an unknown desired goal configuration that optimizes certain known desired new properties or functions. Whereas in self-reconfiguration planning algorithms both the initial and desired configuration are used as input parameters, in our algorithm the input parameters are the initial configuration and the target function. Thus, the design of the target structure can remain unknown in advance, and can be optimized for both the functional requirements and for the elements available in the original structure, and changed on the fly subject to resource fluctuations.

The separation into passive and active units has been considered previously. Werfel and Nagpal [4] present a decentralized algorithmic approach to automatically building user-specified three-dimensional structures from modular cubic units. Terada and Murata [5] present an automatic assembly system based in passive cubic building blocks and an assembler robot. Everist *et al.* [6] present a system—focusing on low-level control instructions for robots—for in-space assembly of 2D structures. Jones and Mataric [7] present a coordinated multi-robot construction system of cubic bricks in a planar structure focusing in the communication and coordination of the robots. Butler *et al.* [8] present a generic model for lattice-based self-reconfigurable robots including several generic locomotion algorithms. The results to be presented here will differ from these studies in that our robotic system decomposes a given structure into constituent building blocks and reassembles the same building blocks into a target structure, and that our approach uses truss structures rather than modular cubic units, allowing lighter structures and more flexibility in reconfiguration.

The problem of graph morphing, the process of transforming a given graph G_1 into another graph G_2, differs from this work as graph morphing deals with isomorphic graphs where the transformation is performed by moving the nodes of the graph smoothly in the space, preserving some constraints in the process [9]. In our problem, however, the edge lengths cannot vary and the graph connectivity is allowed to change during the process. In addition, in graph morphing, both initial and target configurations are given, but in our algorithm only the initial configuration is given along with some functional requirements that the target configuration should meet.

Our work is also unique in its truss optimization. Truss topology optimization is one of the most interesting and at the same time difficult problems in structural optimization. The structure approach was firstly proposed by Dorn [10] where duality was used to formulate the optimal topology problem (minimal weight subject to stress constraints) as a linear programming problem. However, very technical and complicated calculations had to be used. Genetic algorithms proposed by Holland [11] are useful and effective for treating discrete variables and have frequently been used for problems of truss topology optimization [12-14]. Our work is novel in that it optimizes a transformation process, able to be accomplished by robotic manipulation, rather than simply optimizing the final structural design.

3. PROBLEM STATEMENT

The problem addressed in this paper is to obtain a set of instructions to perform on a robotically reconfigurable structure to transform it from a given initial structure to an unspecified target structure satisfying given functional constraints. Rather than simply specifying the configuration of the final structure, the focus of our algorithm is in the process of disconnection and reconnection of edges along with the rotation of the substructures necessary for structure transformation.

The truss structures we utilize can be abstracted as a set of edges connected rigidly to joints in 3D space. The edges cannot vary in length (except in the process of disconnection or reconnection from the joints, as described later) and are indivisible units. The joints only allow edges to be connected to a certain number of fixed axes; we use joints with 18 sockets that accept edges that are aligned to 9 possible axes. A joint can have zero or one edge per socket. Joints are always placed in space in the same orientation or in a rotationally equivalent orientation. Edges can accept a joint on either end or at both ends. The joints that are placed in the floor are tightly connected to it; we ensure at least three joints on the floor to guarantee stability. Edges and joints cannot be superimposed at any point in the structure. These constraints must be met at each stage in the reconfiguration process.

The reconfiguration process is accomplished by disconnecting edges from one or both of their joints in order to divide the structure in two or more sections. Then, those sections are reconnected (potentially after a rotation of the section), forming a new configuration. (In order to keep the connection axes of the joints invariant, the allowed rotations should be integer multiples of 90°.) These steps of division and reconnection can be repeated as many times as necessary. The number of edges cannot vary at any point during the reconfiguration. However, we allow the number of joints to vary freely; thus, the resulting structure does not need to have the same number of joints as the given structure.

The reconfiguration process should contain as minimum number of steps as possible. In order to quantify the cost independently of the robots used to manipulate the structure, we assume the same cost to reconnect any type of section, independently of the number of trusses and joints it contains and the distance that the section should be moved. More realistic performance costs will be determined in future work.

The resulting structure should meet defined functional requirements or constraints. These constraints are open to any numerical measurement that can be performed on the structure. For example, constraints could require structures with a specific height, structures with a maximum height, structures that connect a set of points in the space, structures that support a specific load, etc.

Future work includes the design of the robots capable of traversing and assembling and disassembling the structures, as well as the algorithms necessary for robotic control, path planning, and coordination in order to accomplish the reconfiguration steps.

To the best of our knowledge this problem has not been formulated or solved previously. The number of possible configurations for a given structure of any number of edges *n* is exponential in *n*. As a result, for non-trivial functional requirements or constraints, we have to look for heuristics which can give near optimal solutions.

4. Reconfiguration Algorithm

Rather than using a direct structure representation (such as a graph with struts representing edges and nodes representing hubs), our algorithm uses a construction tree as an intermediate representation. This section of the paper will present this construction tree, explain the operations that can be performed on this tree, and demonstrate the use of the construction tree in the proposed genetic algorithm.

4.1. Construction Tree

Any structure can be deterministically represented by a construction tree specifying a series of assembly operations combining primitive building blocks. A structure can be regenerated deterministically from the construction tree, and conversely, a construction tree (often non-unique) can be generated from a given structure.

A construction tree is composed of primitive units called building blocks. A single building block consists of a certain number of connected edges arranged in a certain fixed configuration. Figure 2a-c shows different types of building blocks in an example cube-like structure. In each cube-like structure, three independent building blocks are highlighted with different colors. The building block consists of one edge in Figure 2a, two edges in Figure 2b, and three edges in Figure 2c. In Figure 2d, the entire structure has been divided using a combination of building blocks of three, two, and one edge. Different constructions trees are obtained from the same structure if different building blocks are used; hence, an ordered list of building blocks should be defined a priori. Additionally, an initial joint must be specified to be used as a

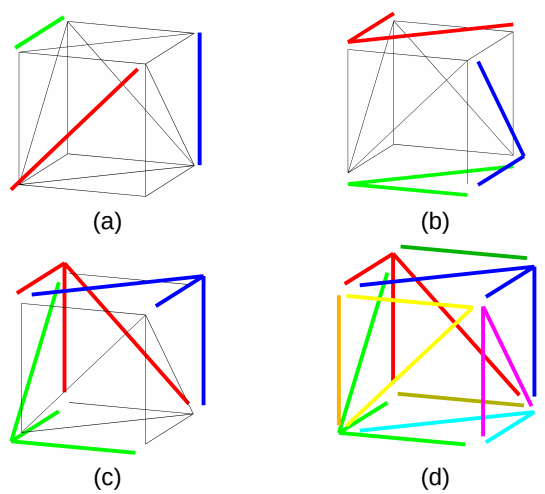


Figure 2. Different types of building blocks can be used to build a construction tree from a truss structure. They are highlighted in color. (a) Three building blocks of only one edge each. (b) Three building blocks of two edges each. (c) Three building blocks of three edges each. (d) The example structure completely composed of building blocks of one, two, and three edges.

starting point for building the construction tree. The built construction tree varies depending on the starting point chosen. Each node of a construction tree stores a building block, its orientation, and the joint that connects that building block to the building block of its parent node.

To obtain a construction tree from a structure, the joints of the structure are traversed with a greedy algorithm using the building blocks as the guides to drive the search. The next joints to be visited are stored in a queue. The first joint that is inserted in this queue is the one defined as the initial joint. The search proceeds extracting and processing the joints from the queue. To process a joint, the algorithm determines if a building block, with edges not yet included in previously building blocks, can be found that uses that joint. The available building blocks are stored in a queue which is searched linearly, starting with the most complex building blocks. The first building block that matches is stored in a new node in the construction tree and the remaining joints in the building block are added to the queue of joints to be visited. If the new node is the first in the construction tree, it is the root node of the tree; otherwise, the new node will be the child of the node whose building block added the joint to the queue. The process continues looking for other building blocks using the same joint. If no building blocks match, the joint is removed from the queue and the next one is extracted and processed. In this way, all the edges of the structure are visited and represented in one of the building blocks of the construction tree. If the queue of building blocks includes single-edged building blocks of each possible lengths (typically as its last elements), it is assured that any structure can be traversed completely.

A step by step example of the generation of a construction tree using a simplified 2D structure is shown in Figure 3. The given structure is shown in frame (a). It has a joint in every

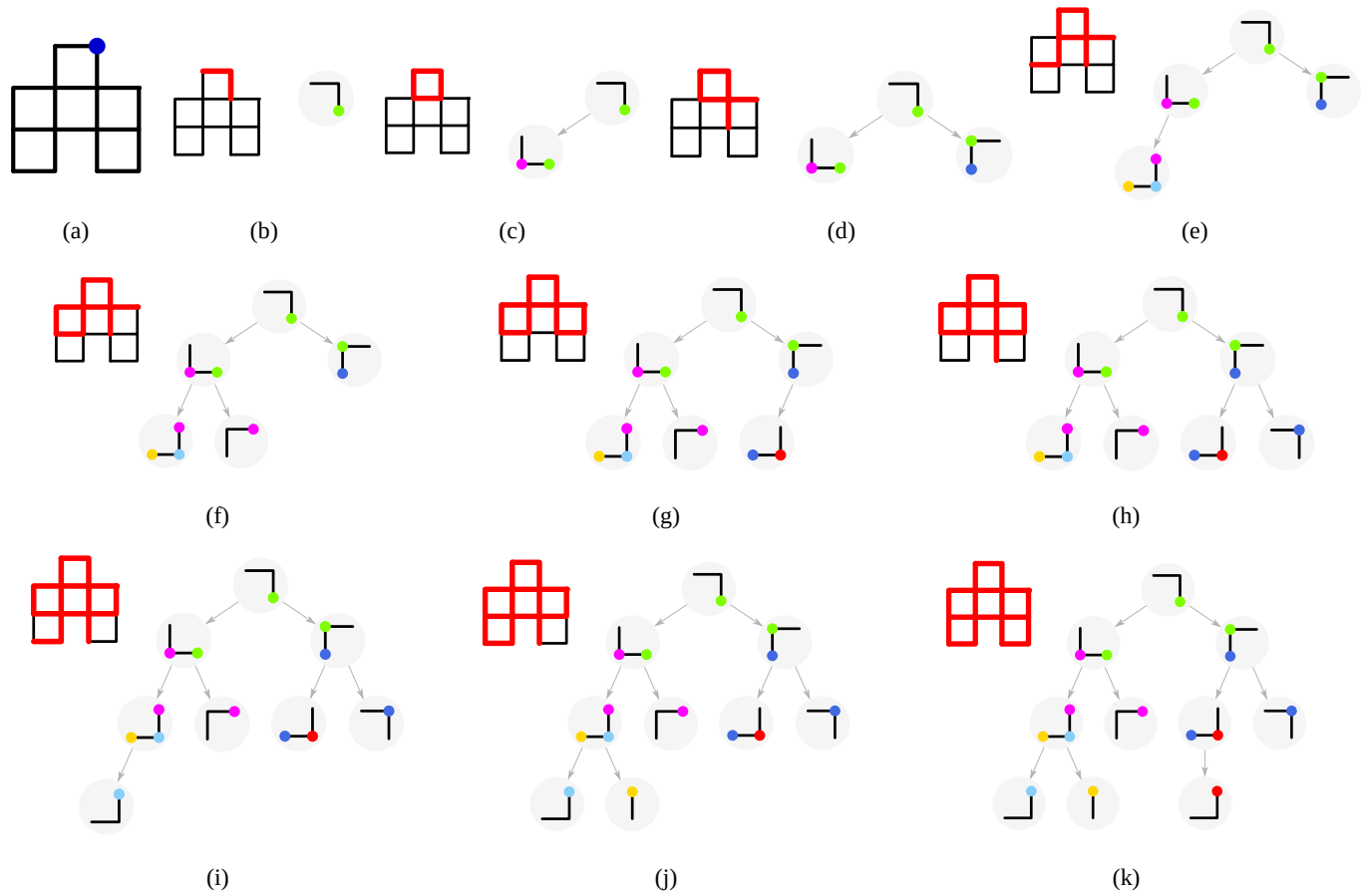


Figure 3. Construction tree representation of a structure. A step-by-step example of the generation of a construction tree simplified using a 2D structure. (a) The given structure with the initial joint highlighted in blue. (b-k) Steps of building the construction tree. In each frame, the given structure with the edges processed so far is on the left, and, on the right, the built construction tree.

intersection of edges. The initial joint is in the lower left, highlighted in blue. In this example, a list of two possible building blocks has been used: one formed by two edges connected in 90°, the second formed by a single edge. In frames (b) to (k), the given structure is shown at left with the already processed edges highlighted in red. The construction tree built so far is shown in the right side of the frame. Inside each node of the construction tree is a representation of the building block contained by that node; the colored dots in the building blocks represent the connecting joints between the building blocks of parent and child nodes. Joints of the same color indicate a connection. Frame (b) shows the first step. The first building block is found (highlighted in red) and the first node is added to the construction tree with the information about the building block and its orientation. The other two joints of that building block are added to the joint queue. There are no more building blocks that can be found from the first joint, so the next joint in the queue is used and a new building block is found (frame (c)). A new node is added to the tree as a child of the first node because the connecting joint came from the building block of the first node. The new node stores the building block used, its orientation, and the joints where this building block connects with the building block of its parent node. (Again, this is shown in the diagram by coloring connecting joints the same color in both nodes.) The same procedure is done in the rest of the structure (frames (c) to (k)). Finally, frame (k) shows the complete construction tree.

The reverse process, obtaining a structure from the construction tree, is performed traversing the construction tree in preorder and connecting each building block via the joints specified in each node.

The use of construction trees is motivated by the necessity of an intermediate structural representation conducive to algorithmic manipulation. There are other well studied intermediate representations that could have been used in this problem, such as grammars or L-systems. Those representations, however, do not fit well with the restriction that the number of struts cannot change during the reconfiguration process, as it is not trivial to design a grammar or an L-system that maintains the number of edges generated after modifying it. The construction tree representation allows basic operators (see section 1.2) to change the underlying structure without modifying the number of struts. Additionally, the problem requires obtaining the representation of a given structure, and that is easily achieved by the construction tree, but not trivial by using grammars or L-systems.

4.2. Structure Variation Operators

Alterations of the construction tree directly correspond to changes in the represented structure, facilitating the generation of a reconfiguration process. We can thus represent the desired reconfiguration process as a series of alterations on the construction tree. Such trees have the key property that if we only move the connection between the nodes in the tree, the number of edges of the generated structure does not change. This is a requirement of the problem.

Several operations that alter a construction tree (and thus the represented structure) have been designed:

- **Branch move**. A first node is selected from the tree and its parent is changed to be a second selected node of the tree. The new parent cannot be in the subtree of the first selected node in order to avoid cycles in the tree.
- **Partial branch move**. This operation is similar to a branch move but a third node is selected from the subtree of the first selected node. The parent of this third node is changed to be the parent of the first selected node. Then, the parent of the first selected node is changed to be the second selected node, as in the branch move.
- **Branch swap**. Two nodes whose subtrees do not share any nodes (again, to avoid cycles in the tree) are selected. The connections to their parent nodes are swapped between them.
- **Branch node rotation**. A node is selected from the tree and all the building blocks in its subtree are rotated at the same angle. (The angle must be compatible with the physical capabilities of the joints.) This causes a part of the structure to rotate around the joint that connects the selected node to its parent node.

In addition to specifying the nodes in a branch movement or swap, the operations also include parameters of the exact joints of the building block where the new parent-child will be connected. Also, in branch moving and swapping, a rotation is performed in the subtree that has been moved. This is necessary to connect the new subtree in an appropriate orientation. The rotation axis and degree (in integer multiples of 90°) are also parameters in these operations. The problem of possible superposition will be addressed later on.

The set of structure variation operators can transform any tree to any other tree with the same number of nodes. So, the algorithm can achieve any block connectivity and any block rotation. Therefore, using trees with blocks of 1 edge, the algorithm can transform any structure to any other structure with the same number of edges. The use of blocks of more than 1 edge brings faster computation in exchange for reconfiguration restrictions.

Figure 4 shows an example of the branch move operator using the same simple structure and construction tree used in Figure 3. Although the operators are applied only to construction trees, in order to clarify the process, the figure shows the structures that each intermediate construction tree codifies. Frame (a) shows the construction tree that was built in the example of Figure 3 and the structure that is codified by it. Frame (b) highlights the branch selected by the move branch operator in the construction tree and highlights the affected edges in the structure. In frame (c), the branch has moved according to the branch move operator, which changes the physical structure as shown. Finally, frame (d) shows the rotation operator applied to the tree in frame (c). In this case, the rotation is 90° counterclockwise and is applied to each node in the tree.

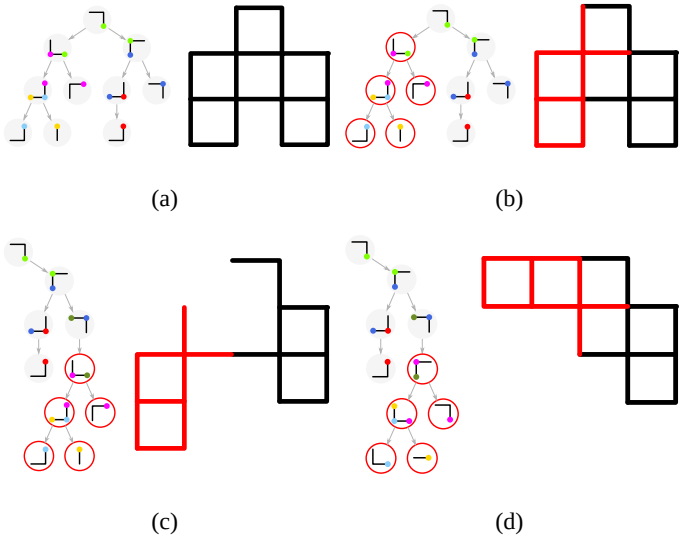


Figure 4. An example of the branch move and rotation operators applied to a construction tree. (a) An initial structure and its construction tree. (b) The branch selected by the move branch operator and the structure edges codified by that branch are highlighted. (c) The branch has moved according to the branch move operator by changing the parent of the root node of the branch. The resulting change in the structure is shown on the right. (d) A rotation operator is applied to the same subtree.

4.3. Morphology Search Process

A genetic algorithm was implemented to evolve the reconfiguration sequence applied to the structures. The morphogenetic function, the process of obtaining a phenotype from a genome, proceeds first by building a construction tree from the given structure. Then, the construction tree is modified using the operations shown in section 4.2. Finally, the resulting structure is generated from the modified construction tree and its fitness is evaluated. Fitness in the context of genetic algorithms quantifies the optimality of a solution. Specifically in our algorithm, fitness measures how optimal a solution is in terms of the cost of the transformation (number of steps) and how well the structure meets the functional requirements. The first step is performed at the commencement of the algorithm, and the other two steps are performed to each candidate individual of the evolving population.

The genome of each individual starts with a gene that encodes the structure's starting joint for the generation of the construction tree of the given structure. This gene is a natural number in the range of 1 to n where n is the number of joints of the given structure. (Again, this is necessary to make the relationship between construction trees and physical structures

deterministic.) This is the only gene in the genomes of the initial population, and is selected at random for each individual in the initial population.

Secondly, the genome contains the list of operations among those shown in section 1.2 that will be performed to the construction tree during the morphogenetic function. Each operation will be encoded as a different gene and will contain the necessary parameters to define it. The number of operation genes that are in the genomes of the individuals is variable, and hence the length of the genomes is variable.

The genetic operators used are one-point crossover and mutation. The mutations that can be applied to a genome are:

- **Mutate the first gene**. The first gene represents the initial joint of the structure where the building of the construction tree starts. This mutation will insert a random value in the range of 1 to *n*, where *n* is the number of joints of the given structure. Changing this gene will result in a different construction tree in the morphogenetic function.
- **Mutate an operator parameter**. One parameter of a randomly chosen operation genes is replaced to a random value within the limits of that parameter.
- **Delete a gene**. One operation gene is chosen randomly and is removed.
- **Add a gene**. A position is chosen randomly and a new operator gene is inserted. The operator and its parameters are also randomly chosen.

The selection method used in the genetic algorithm is deterministic crowding [15], where offspring replace their most similar parent if they have equal or better fitness and are discarded otherwise. Population size (50), mutation probability (0.5), and crossover probability (0.5) have been tuned empirically.

An optimal reconfiguration implies that the reconfiguration is done in the minimum possible number of steps. In order to contemplate this, the fitness is specified as a linear combination of the functional requirements specific to the problem and the number of genes in the genome. Also, overlapping edges are not allowed in any step of the reconfiguration. A very low fitness is given if this happens in an individual.

5. Physical Implementation

In order to demonstrate physical feasibility, we designed a set of truss elements capable of implementing the structures formed by our algorithm and suitable for robotic manipulation. The elements we designed consist of struts and nodes capable of assembling cubic trusses with face-centered diagonals. Figure 5a contains a close-up image of a strut and node and Figure 5b contains an example of a cube constructed with such parts. The nodes consist of 18 threaded sockets in each of the necessary axes, inserted into a 3D-printed shape. The struts consist of two identical carbon fiber rods internally threaded on both ends via threaded inserts fastened inside the rods with epoxy. See [16] for details.

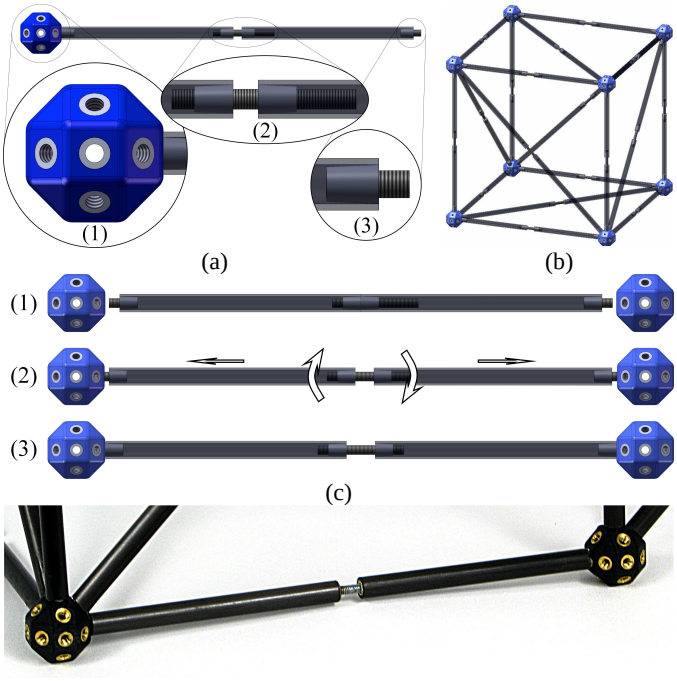


Figure 5. Truss elements allow order-independent assembly: (a) Elements consist of an 18-axis threaded node (details shown in a1) and a long strut. This strut is split in the center with a threading mechanism (a2) and uses a threading mechanism on either side to attach to the node (a3); (b) Elements can be used to construct a variety of structures such as this cube; (c) The truss elements can be accessed in a random-access manner by inserting the contracted element between nodes (c1), twisting the two element halves (c2) causing the center thread to lengthen and the end threads to enter the nodes (c3). Note that all threads are right-handed threads. (d) A physical implementation of node and strut.

6. Results

This section gives results on various examples of structure reconfiguration demonstrating our approach. The algorithm has been implemented in Matlab (MathWorks Inc.). The time to evolve the structures is about 10 hours running the code in Matlab in a conventional computer (Pentium 4, 3.2 GHz, 1 GB memory, Microsoft Windows XP). Figure 6a shows the given three-dimensional structure used in these examples, with a height of 2 units. Joints fixed in the floor are highlighted in the figure by a larger size.

In these examples, we are attempting to maximize the height of the structure although minimizing the number of reconfiguration steps. So, our algorithm has tried to find the best reconfiguration steps that transform the given structure into another structure that is as tall as possible. The fitness criterion consists in a linear combination of the height of the structure and the number of steps in the reconfiguration, i.e. the number of genes in the genome, because each gene represents a step in the reconfiguration. The coefficients of the linear combination are 1 and -0.01, respectively. The building blocks used in the construction trees of these examples are as follows: first, two connected edges with 90 degrees between them; next, two connected edges with 45 degrees; next, two

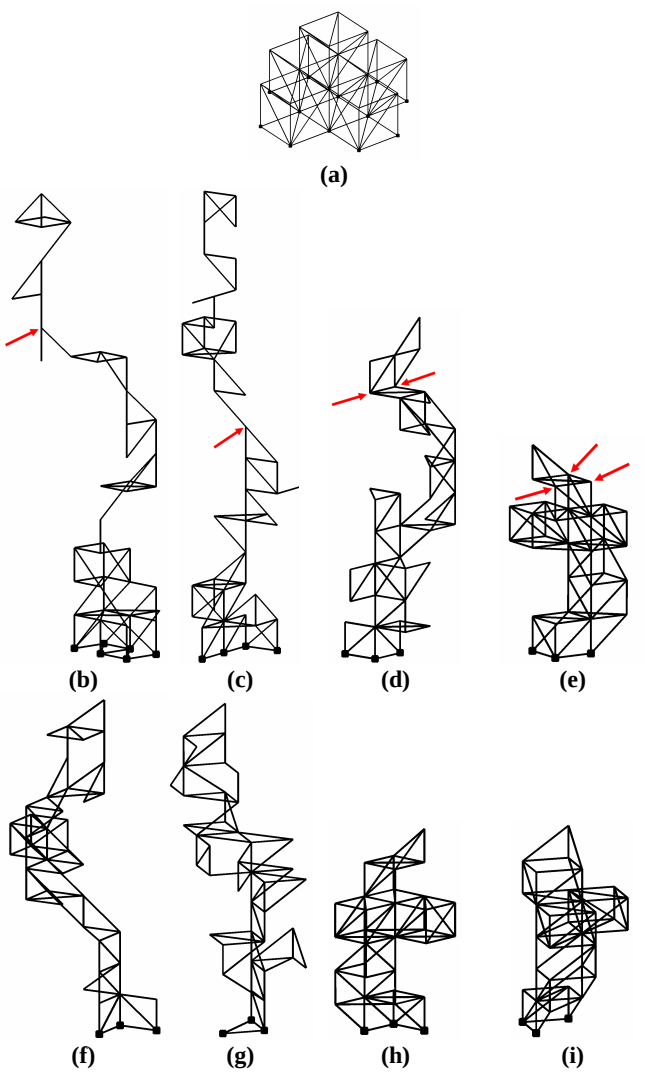


Figure 6. (a) Initial structure. Joints on the floor are highlighted. (b-i) Evolutionary reconfiguration results from the initial structure in Figure 6a with the criteria of maximizing the height while minimizing the number of reconfiguration steps. Red arrows in (b, c, d, e) show a set of minimum number of nodes that can be eliminated in order to separate the structure; hence, the number of arrows is the vertex connectivity of the graph. (b, c) Two results with a height of 14 units but a very low robustness (the vertex connectivity is 1) from two different evolutions. (d) Resultant structure with vertex connectivity 2. (e) Result structure with vertex connectivity 3. (f, g) Additional results structures with vertex connectivity 2 and height 8 units. (h, i) Additional results structures with vertex connectivity 3 and height 6 units.

connected edges with any angle; and if none of the above has been found, a single edge. Figure 6b and Figure 6c show two preliminary results after two different evolutions, each of 5000 generations. Both structures have a height of 14 units, improving the given structure by 12 units in a reasonably number of 8 and 9 steps respectively.

However, these structures lack robustness: it is easy to appreciate that there are parts in the structure where a single joint undergoes a large bending moment. In order to resolve this problem, a new element is introduced in the fitness: the vertex connectivity of a graph. The vertex connectivity of a graph is the minimum number of vertices that are necessary to remove in order to disconnect the graph. Since, the resulting structure can be treated as a graph, it is possible to calculate its vertex connectivity. The vertex connectivity of the structures of Figure 6b and Figure 6c is 1, because removing, for example, the vertex marked with the arrow causes the graph to be disconnected. Notice the structures with vertex connectivity 2 in Figure 6d and vertex connectivity 3 in Figure 6e. The red arrows indicate a minimum set of vertices that, if removed, disconnect the graph. In the fitness function, individuals whose vertex connectivity is below a desired value are penalized with a very low fitness.

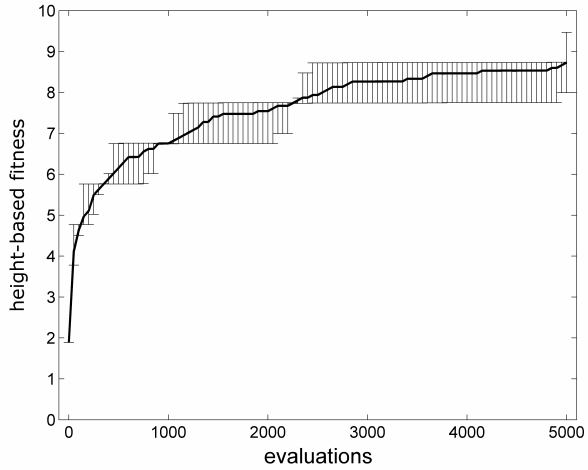
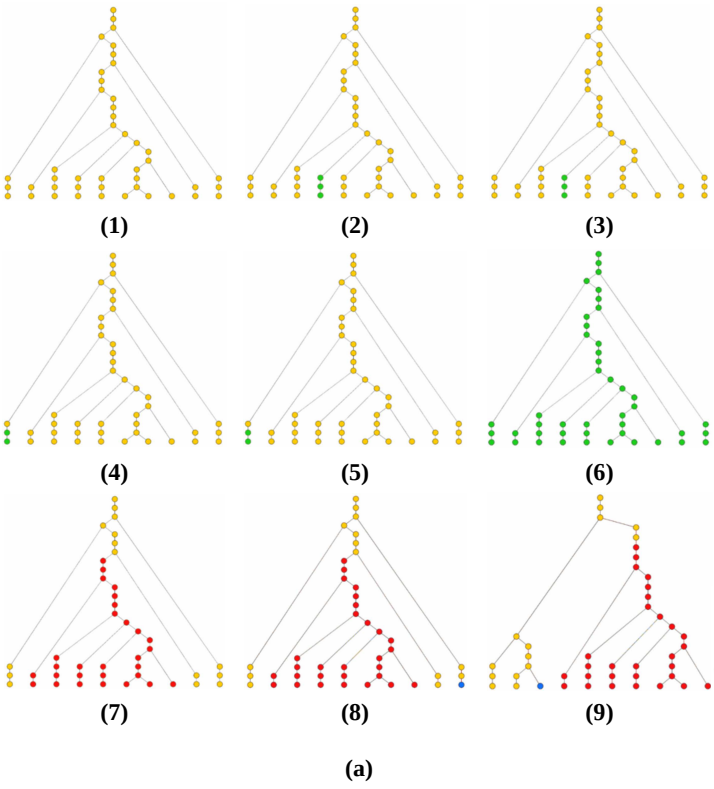


Figure 7. Average maximum population fitness values over 15 evolutionary runs, with error bars indicating the first and third quartiles.

Results with vertex connectivity 2 are shown in Figure 6f and Figure 6g with two structures, both with height 10 units. (Again, 5000 generations were used in the evolutionary algorithm.) Similarly, results for structures with vertex connectivity 3 are shown in Figure 6h and Figure 6i. These two structures have a height of 6 units. Figure 7 shows the average maximum population fitness values over 15 evolutionary runs. Figure 8 and Figure 9 show a reconfiguration result in detail. Figure 8a shows the reconfiguration trees in each step of the reconfiguration process. The genome contains the following operations: a branch movement (the branch that moves is shown in green) from the tree in (a3) to (a4); a rotation of the full structure in (a6); and a branch swap in the trees (a8) and (a9) (between the branches shown in red and blue).

The total number of operations in this solution is 3. Figure 8b shows the simulated structure reconfiguration. The building blocks corresponding to the nodes highlighted in (a) are also highlighted in (b). Figure 9 shows the same structure reconfiguration using a physical implementation suitable for robotic manipulation (presented in section 5) as demonstration of physical feasibility.



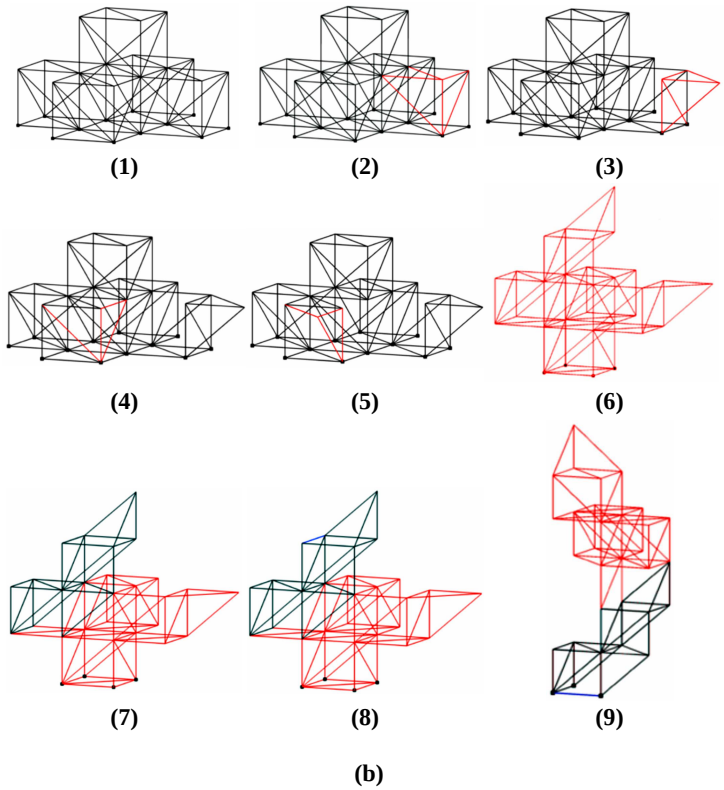


Figure 8. Detailed reconfiguration steps from an evolution result. (a) Reconfiguration trees in each step of the reconfiguration process. (b) Structure reconfiguration in the simulation. The building blocks corresponding to the nodes highlighted in (a) are also highlighted in (b).

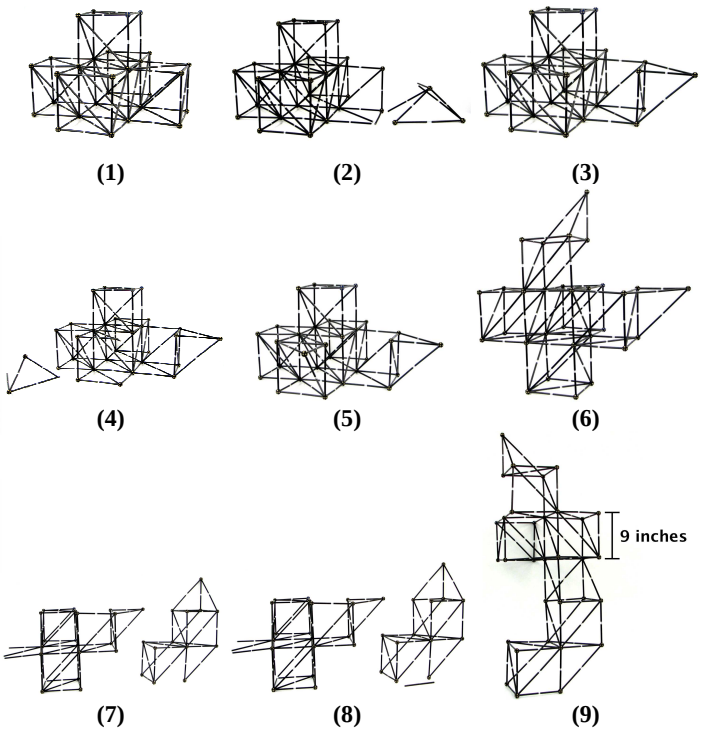


Figure 9. Physical implementation of structure reconfiguration. The same reconfiguration steps from the evolution result shown in Figure 8 using our truss elements as a demonstration of physical feasibility.

Other sizes of building blocks have also been tried. Besides the building blocks of 2 edges used in the examples shown thus far, we have tried also building blocks of just 1 edge (Figure 10a) and of 3 edges (Figure 10b). Figure 11a shows the maximum fitness of 15 evolutions for each combination of vertex connectivity required (2 or 3) and the building blocks used (building blocks of 1, 2 or 3 edges). Figure 11b is equivalent but displays the mean fitness and its standard deviation among the same 15 evolutions per vertex connectivity and building block combinations.

6.1. Resource Fluctuation and Complex Fitness

In order to demonstrate the robustness of the algorithm we have performed experiments, using the same building blocks and vertex connectivity of 2, by randomly deleting 10% of the edges of the given structure (Figure 10c). This is an analogous to a lack of proper nutrient resources in biological metabolism. The average results obtained after 20 runs do not degrade with this change, in terms of height. An example is shown in Figure 10d. The height of this structure is 10 units, the same that the structures obtained from the original given structure. This robustness demonstration is very important for real problems with the robotic system where some truss can be lost or broken.

Also more complex functional requirements have been tested. A physics simulator based on springs has been implemented to simulate realistically the resulting structures [17]. Figure 12 shows an evolutionary result using the physics

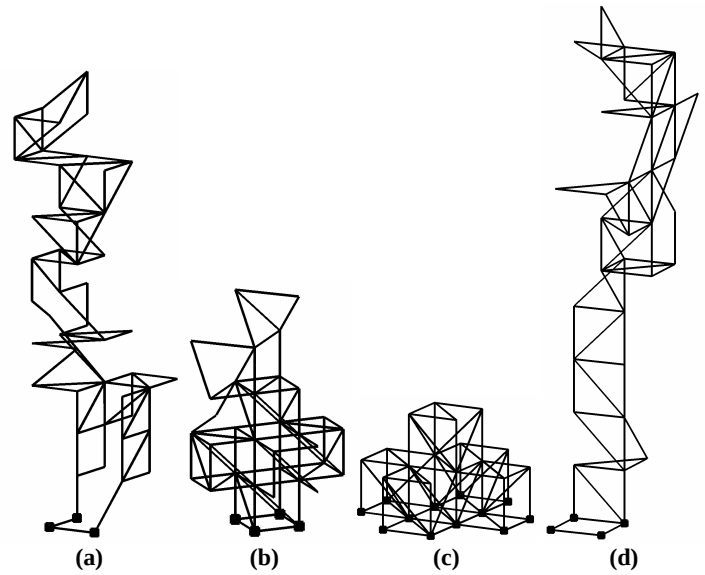


Figure 10. Evolutionary results using different building block configurations (a-b) and demonstrating robustness of the algorithm (c-d). The building blocks consist of (a) just 1 edge and (b) 3 edges. Both results have a vertex connectivity requirement of 2. (c) Given structure with 10% of its edges deleted. (d) Reconfigured structure found by the algorithm from the given structure in (c). The resulting structure does not degrade (in terms of height or vertex connectivity) with the loss of edges of the given structure.

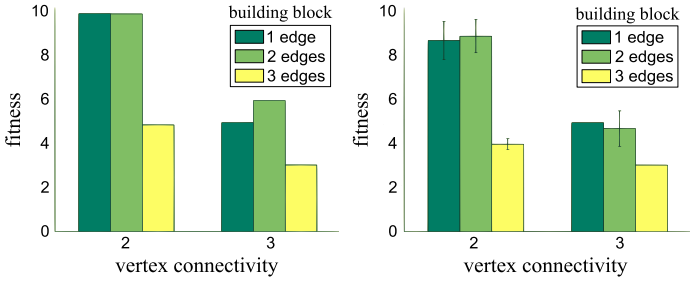


Figure 11. Comparison of different building-block types. Performance comparison using 15 evolutions for each combination of vertex connectivity required (2 or 3) and building block type used (building blocks of 1, 2, and 3 edges). (a) Maximum fitness. (b) Mean fitness and its standard deviation.

simulator in the fitness function. The functional requirements used in this example were modeled with the aim to find a bridge, maximizing the length of the structure but, at the same time, minimizing the tension of the edges and the deflection of the structure under its own weight. The fitness criterion is a linear combination of the length, tension, deflection of the structure and the number of steps in the reconfiguration. The coefficients in such linear combination are 1, -1, -1 and -0.01 respectively. The structure tension is calculated as the average sum of the tensions of each truss.

The structure deflection is calculated as the distance between the lowest node of the structure and the lowest fixed node of the structure. Fixed nodes are those which x

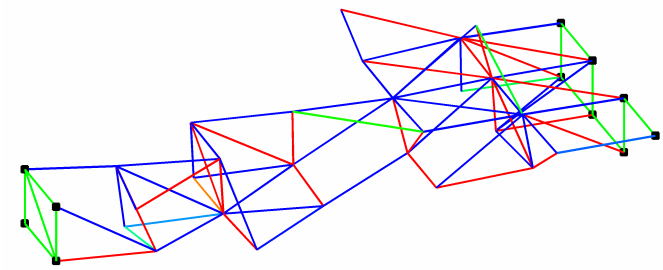


Figure 12. Evolutionary result using a physics simulator based on springs as part of the fitness function. The functional requirements used in this example were modeled with the aim to find a bridge, starting with the structure in Figure 6a. They consist in maximize the length of the structure but minimizing at the same time the tension of the edges and the deflection of the structure under its weight. The colors of the edges indicate degree of stress: blue indicates tension, red compression, and green indicates a relaxed state. The nodes highlighted in black indicate fixed nodes.

coordinate is maximum or minimum, i.e. the most right and most left nodes of the structure. The colors of the edges indicate degree of stress: blue is tension, red is compression, and green is relaxed. The black nodes are fixed to ground on either side of the bridge. In the fitness function, the right-most and left-most nodes of the resulting structure remain fixed in the physics simulator. The given structure in this example is also the structure shown in Figure 6a.

Figure 13 shows additional evolutionary results. Starting with the two short towers shown in Figure 13a, a bridge using the physics simulator (Figure 13b) and a tower that maximizes its height while minimizing the number of reconfiguration steps (Figure 13c) have been evolved separately. Figure 13d shows a human-designed bridge from which a shelter that covers maximum area above the ground (Figure 13e) and a tower that maximizes its height (Figure 13f) are evolved separately.

7. Conclusion

This paper has demonstrated an algorithmic process that solves the problem of how to transform one truss structure into another that achieves a specific functional goal. We have demonstrated the transformation process generated by this algorithm on a physical truss.

Our algorithm successfully creates transformation steps useful in robotic reconfiguration. It uses the concept of a construction tree as a formal representation of the structures in order to allow for tree transformation operations equivalent to transformation steps in the structure. The functional requirements are specified in the evolutionary algorithm as part of the fitness function. Finally, the quantity of struts is conserved during the algorithmic manipulation.

The algorithm has been proven with several experiments using 3D structures. First, we presented experiments that reconfigured the given structure to new structures maximizing their height while minimizing the number of reconfiguration steps. We investigated different parameters in the algorithm to

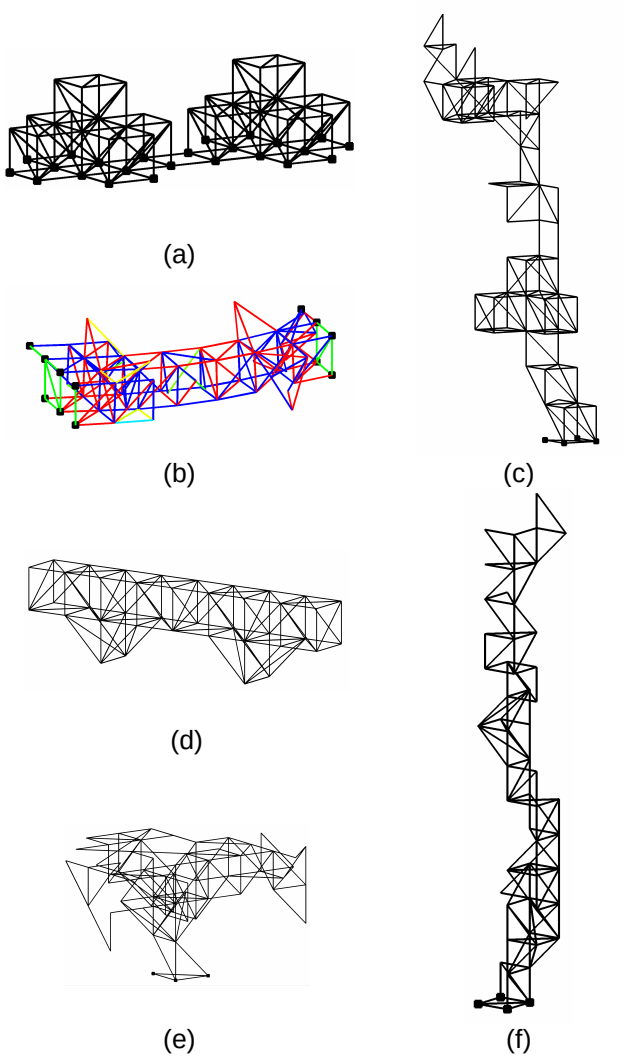


Figure 13. Additional evolutionary results. From the two tower structure of (a) a bridge using the physics simulator (b) and a tower that maximizes its height (c) are evolved separately. In (d) is shown a hand-designed bridge used to evolve both (e) a shelter structure, covering the maximum area above the ground, and (f) a tower that maximizes its height.

test its performance. We also presented experiments to test the robustness of the system by deleting 10% of the edges of the given structure and showing still successful reconfigurations. A more complex experiment was presented using a physics simulator in the fitness function to test several physical parameters. For example, we showed a result that consisted in a bridge structure that maximized the length of the bridge but minimizing the tension of the edges and the deflection of the structure.

Finally, we demonstrated physical feasibility by showing physical truss elements in a step by step reconfiguration process designed completely by the genetic algorithm.

In future work, we will design the robots capable of traversing and assembling and disassembling the structures and the algorithms necessary for robotic control, path planning, and coordination in order to accomplish the reconfiguration steps autonomously.

ACKNOWLEDGMENTS

Thanks are due to Francisco J. Vico for valuable suggestions and support, and to the groups of Daniela Rus, Eric Klavins, and Mark Yim for perceptive discussions about machine metabolism. This work was supported in part by the U.S. National Science Foundation's Office of Emerging Frontiers in Research and Innovation, grant #0735953, and by the Sixth European Union Framework Program for Research and Technological Development, contract #028892.

REFERENCES

[1] M. Yim, W. M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]". *Robotics & Autom Mag, IEEE*, 14(1): 43-52, 2007.

[2] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata, "Distributed adaptive locomotion by a modular robotic system", in *Proc. IROS'04*, 2004.

[3] M. De Rosa, S. Goldstein, P. Lee, J. Campbell, and P. Pillai, "Scalable shape sculpting via hole motion: motion planning in lattice-constrained modular robots", in *Proc. ICRA'06*, 2006.

[4] J. Werfel, and R. Nagpal, "Three-Dimensional Construction with Mobile Robots and Modular Blocks", *Int J Robotics Res*, 27(3-4): 463-479, 2008.

[5] Y. Terada, and S. Murata, "Automatic assembly system for a large-scale modular structure - hardware design of module and assembler robot", in *Proc. IROS'04*, 2004.

[6] J. Everist, K. Mogharei, H. Suri, N. Ranasinghe, B. Khoshnevis, P. Will, and W. M. Shen, "A system for in-space assembly", in *Proc. IROS'04*, 2004.

[7] C. Jones, and M. J. Mataric, "Automatic synthesis of communication-based coordinated multi-robot systems", in *Proc. IROS'04*, 2004.

[8] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic Decentralized Control for Lattice-Based Self-Reconfigurable Robots", *Int J Robotics Res*, 23(9): 919-937, 2004.

[9] S. G. Kobourov, and M. Landis, "Morphing planar graphs in spherical space". *J Graph Algorithms Applications*, 12(1):113-127. (2008)

[10] W. S. Dorn, R. E. Gomory, and H. J. Greenberg, "Automatic design of optimal structures", *J Mecanique*, 3: 25-52, 1964

[11] J. Holland, "Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence", Univ. of Michigan Press, 1975.

[12] C. A. Coello, and A. D. Christiansen, "Multiobjective Optimization of Trusses using Genetic Algorithms", *Computers Structures*, 75(6): 647-660, 2000.

[13] W. Lingyun, Z. Mei, W. Guangming, and M. Guang, "Truss optimization on shape and sizing with frequency constraints based on genetic algorithm", *Computational Mechanics*, 35(5): 361-368, 2005.

[14] S. Rajeev, and C. S. Krishnamoorthy, "Genetic algorithm-based methdologies for design optimization of trusses", *J Struct Engineering*, 123(3): 350-358, 1997.

[15] S. W. Mahfoud, "Crowding and preselection revisited", in *Proc. PPSN'92*, 1992.

[16] D. A. Hjelle, H. Lipson, "A robotically reconfigurable truss", in *Proc. ReMAR'09*, 2009.

[17] D. Lobo, "MatSprings: Matlab 3D spring truss simulator", 2008. [Online]. Available: http://www.geb.uma.es/matsprings.html