2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Planning the Reconfiguration of Grounded Truss Structures with Truss Climbing Robots that Carry Truss Elements

Seung-kook Yun[1], David Alan Hjelle[2], Eric Schweikardt[2], Hod Lipson[2,3], Daniela Rus[1]

[1]*Computer Science and Artificial Intelligence Laboratory*
Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

[2]*Sibley School of Mechanical and Aerospace Engineering*
[3]*Computing and Information Science*
Cornell University, Ithaca, New York, USA

yunsk@mit.edu, dah283@cornell.edu, ees68@cornell.edu, hod.lipson@cornell.edu, rus@csail.mit.edu

***Abstract*— In this paper we describe an optimal reconfiguration planning algorithm that morphs a grounded truss structure of known geometry into a new geometry. The plan consists of a sequence of paths to move truss elements to their new locations that generate the new truss geometry. The trusses are grounded and remain connected at all time. Intuitively, the algorithm grows gradually the new truss structure from the old one. The truss elements are rigid bars joined with 18-way connectors. The paper also introduces the design of a truss-climbing robot that can execute the plan.**

I. Introduction

Our long-term goal is to apply a reconfiguration paradigm to construction via self-assembly. We wish to create self-assembling robot systems consisting of passive structural modules, possibly manufactured on-demand and/or composed from elements present in the environment, combined with active robotic modules. The structural passive elements are rigid passive bars and general connectors capable of supporting multiple bars. The active elements are robotic modules that may travel on the structural components, pick up or disassemble a passive element from a known location in the structure, carry the element to a desired location on the structure and connect the passive element at the destination. For the case when the passive elements are rigid bars, the structures that can be created with this paradigm are self-assembling and self-reconfiguring trusses. If the robot elements are an integral part of the truss, the truss is a dynamic and controllable structure. The resulting structures form a large class of truss and linkage geometries. We introduced some of our initial ideas for designing robots and passive parts, as well as self-assembly and control algorithms for these types of trusses in [1], [2], [3].

In this paper we describe an algorithm for reconfiguring truss structures. We assume that the truss structure consists of passive elements. Truss-climbing robots are capable of (1) disconnecting a truss element, (2) carrying it along a path on the truss, an (3), re-attaching it to new locations on the truss. We develop an algorithm that takes as input a grounded target truss structure and a grounded goal truss structure. The algorithm computes an optimal set of truss element moves that reconfigure the target truss into the goal truss while ensuring that all truss elements stay connected at all times. The algorithm considers trusses that consist of rigid bars of a fixed number of lengths and connectors of one type. Figure 1 shows an example for changing the geometry of a truss structure.

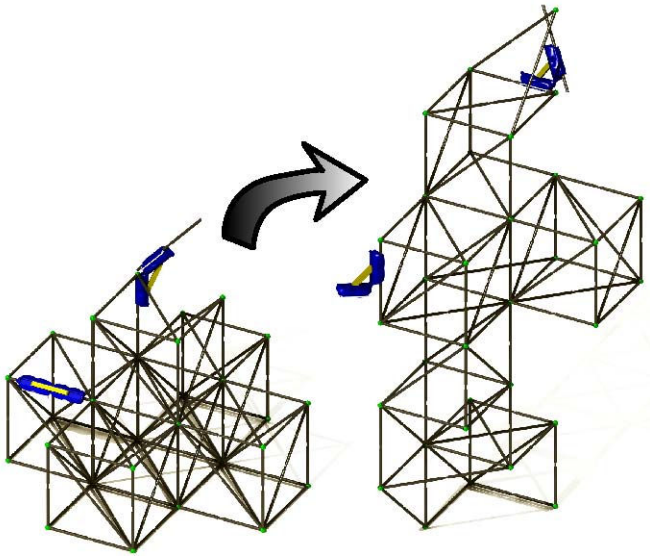

Fig. 1. Artist rendition of several hinge robots decomposing and recomposing truss structures. Structural metabolism replicates properties of biological metabolism such as autonomous disassembly and assembly, continuous reuse of modular elements, automated design from functional requirements, and resilience to raw material variation.

The main contribution of this paper is algorithmic. We also propose a system design for truss elements (connectors and edges of two types), and robots capable of executing the reconfiguration algorithms proposed. This truss robot system is partially completed and will be the subject of a different paper.

A. Inspiration

In developing our problem formulation and solution, we were inspired by biological metabolism. An organism breaks down food into constituent building blocks (catabolism) and then uses those building blocks for its own growth (anabolism). This process has several properties of interest to our problem: extensive reuse and recycling of modular building blocks, autonomous deconstruction and reconstruction, autonomous design given a set of functional requirements, resilience to raw material variation, and self-repair. These properties have not generally been replicated in synthetic structures. Duplicating such properties in a robotic ecology could yield a number of practical benefits, from more robust manufacturing processes to improved recycling ability to space exploration.

The general replication of these properties is a distant computational goal. Many challenges exist. How many and what kind of building blocks could be used for effective metabolic processes? (Doyle *et. al.* [4] suggest that a relatively small set of building blocks can yield a large number of source and target structures in a robust manner.) How can functional definitions be transformed into physical designs in an autonomous way? What kind of manipulation is required? How is the transformation process to be accomplished?

If one simplifies the problem to solely deal with single-diagonal cubic truss structures as shown in Figure 1, the problem becomes more tractable, but significant questions still remain.

B. Related Work

Many robots have been developed as climbers and/or manipulators of various structures, including general truss structures. An inchworm robot, using electromagnetic force to attach itself to ferrous surfaces, was developed in [5], but structural manipulation abilities are not included. Ripin *et al.* [6] and Tavakoli *et al.* [7] developed pole-climbing robots, and the latter has some capacity for manipulation. Amano, Osuka, and Tarn [8] developed a robot for climbing high-rise buildings. None of these three robot is capable of general truss traversal, however. TREPA [9], a parallel robot; ROMA [10], a caterpillar-like robot; and Shady3D[11], a modular robot utilizing a passive member, are capable of traversing a wide variety of structures, but do not have the ability to effect structural assembly. Nechyba and Xu [12] developed a truss-walking inspection robot, SM^2, for space station trusses. Skyworker was developed for orbital assembly tasks [13], and was demonstrated performing truss-like assembly tasks [14]. However, the truss structure was not specifically designed for robotic manipulation and required an independent vision robot to perform the assembly. Our proposed robot design (discussed briefly in section II and in more detail in [15]) is fashioned for traversal of general cubic trusses with face-centered diagonals and for physical manipulation and reconfiguration of such trusses. It utilizes a custom truss design discussed briefly in Figure 3 and in detail in [16].

C. Outline

This paper is organized as follows. Section II introduces the design for a truss climbing robot and for the truss elements (edges and nodes) it operates with. Section III-B describes and analyzes an optimal reconfiguration planning solution that is centralized and efficient but does not maintain truss connectivity. Section III-C describes and analyzes a reconfiguration planning solution that is guaranteed to maintain connectivity and compute the correct plan. Section IV presents simulation results.

II. Designing A Robot Truss Carrier

The robot design we have chosen for this work is one we refer to as a hinge robot, as shown in Figure 2. Two halves of the robot are connected together via a hinge. The angle between the halves can be adjusted to any angle between 35° and 180°. This is done with a pair of linear actuators and a rigid connecting link. Moving one or both of the linear actuators changes the angle between the halves. Each half, then, is capable of gripping a truss element, rotating itself and the entire robot around that truss element, and translating along that truss element using internal wheeled grippers.

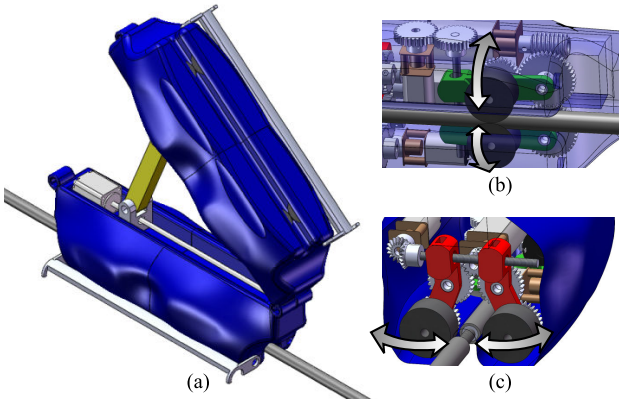

Fig. 2. The hinge robot (a) is capable of 35° to 180° actuation between the halves via two linear actuators, one on each half. The translational mechanism [shown from the bottom in (a)] and rotational mechanism [shown in a cutaway view in (b)] can be actuated to grip or release the truss element, as well as move the robot along the element. The rotational actuation doubles as the ability to fasten and unfasten the truss elements via twisting.

If a single half of the robot is currently gripping a truss element, the robot can then move about a truss by translating, rotating, and adjusting the angle between halves in order to align the free half with another truss element adjacent to it. (This element may be at an angle of 45°, 90°, 135°, or 180° from the originating element.) The robot then grips the destination element, releases the originating element, and the process begins again.

If a strut needs to be carried from one location to another on the truss, a carrying pod (shown on the side of each half of the robot in Figure 2) can be actuated to retrieve a strut from the structure, hold it, and return it as needed.

The robot is also capable of manipulating the truss, given amenable truss element designs. The truss elements in Figure

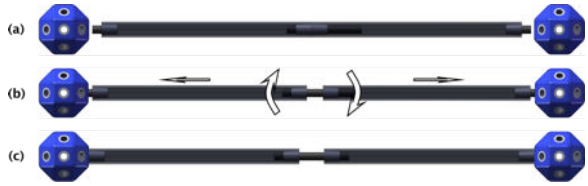

Fig. 3. Truss elements designed for robotic manipulation. The hubs contain threaded holes in each principle axis as well as in the diagonal axes. The struts contain threaded studs on either end, and also have a threaded stud and mating insert in the middle. This allows the sides to be inserted or removed from the hub independently via rotation without requiring the movement of the hub.

3 are designed for ease of robotic manipulation via a twisting action in complex truss structures [16]. Since the robot is able to rotate itself about a truss element, it is also able to remove struts via rotation. Thus, the robot is capable of completely manipulating such truss structures.

III. Planning the Optimal Reconfiguration of Trusses

In this section we formulate the specific truss reconfiguration problem we solve in this work and describe an efficient algorithm.

The truss structures we consider are general compositions and arrangements of rigid bars of different lengths connected at truss nodes using the connector design described in Section II. Our truss uses a meta-module with the geometry of a cube. Thus, without loss of generality, we focus on two types of truss elements: (1) *sides*, which are short bars for the cube sides, and (2) *diagonals*, bars that make up the diagonals in the basic cube meta-structure. All trusses are grounded.

Each truss structure is represented as a weighted graph $G = (V, E)$. Vertices V correspond to the nodes of the truss. Edges E show the connectivity of the truss nodes by truss elements. The weight of each edge indicates the type of edge (e.g. side or diagonal for the proposed design.)

Planning for the reconfiguration of the truss represented by graph G_1 into the truss represented by graph G_2 can be formulated as optimal matching between G_1 and G_2. we wish to keep the truss connected at all times as well as to guarantee the globally optimal matching solution. The intuition is as follows. First, we compare G_1 and G_2 to identify their overlap. The overlap corresponds to truss elements that do not have to move in the process of reconfiguring one object into another. Next, truss elements in G_1 that are not part of this overlap are assigned new locations to assemble G_2. This involves computing a truss trajectory for moving each element to the new location (by robots that can carry truss elements), and the order in which the moves have to be done. We wish to minimize the total number of steps required to complete the truss reassembly task.

A. Problem Formulation and Assumptions

The goals for the truss reassembly planning algorithm are as follows:

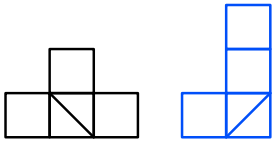

Fig. 4. Source and target structure: G_1 (left) and G_2 (right)

- find the graph $G_m = G_1 \cap G_2$ that yields the optimal matching between $G_1 - G_2$ and $G_2 - G_1$
- compute the trajectories for moving the edges in $G_1 - G_2$ to assemble G_2 subject to the constraint of maintaining connectivity of the entire structure

The cost function is the total traveling distance of the truss elements (e.g. for the edges in $G_1 - G_2$). The cost function can be extended with other criteria such as maintaining the integrity of the structure in the presence of gravity, etc.

The goals for the general reassembly planning problem of arbitrary trusses are challenging. The maximal common subgraph (MCS) isomorphism algorithm is NP-hard [17] in the general case. Though the bipartite matching in a graph can be solved, executing the matching may disconnect the graphs in our case because the matching physically moves a truss that includes an edge and a couple of nodes.

In order to find an efficient solution for truss reassembly planning, we make the following assumptions:

- G_1 and G_2 are restricted to two types of truss elements: *sides* and *diagonals*. Each edge is labeled by its type (side or diagonal). This assumption is relaxed straightforwardly to trusses with a finite number of types for their truss elements.
- The orientation of the side truss elements is orthogonal along one of the x, y, or z axes.
- A truss node (vertex) has the ability to hold and store multiple truss elements.
- Trusses are grounded.

The first, second, and fourth assumptions restrict the geometric structure of the truss. This class of trusses admits polynomial-time algorithms for reconfiguration planning. The third assumption is needed to ensure that during the process of reconfiguration the structure remains connected. The prototype connector (see Figure 3) was designed with this goal in mind. Thus, a robot can connect a truss element to a node temporarily.

We do not impose any restriction on the size of G_1 and G_2. Specifically the size of the initial truss does not have to be identical to the size of G_2.

B. Finding the optimal matching by scanning

The truss elements have fixed lengths and orientations. Therefore, we may consider the truss structure as a set of the square cubes whose edges are the side truss elements and whose diagonals are diagonal truss elements. The optimal matching can be obtained by scanning G_2 over the G_1 as described in Algorithm 1.

Figure 4 illustrates Algorithm 1 in the context of a 2D example for ease of explanation. Algorithm 1 works with

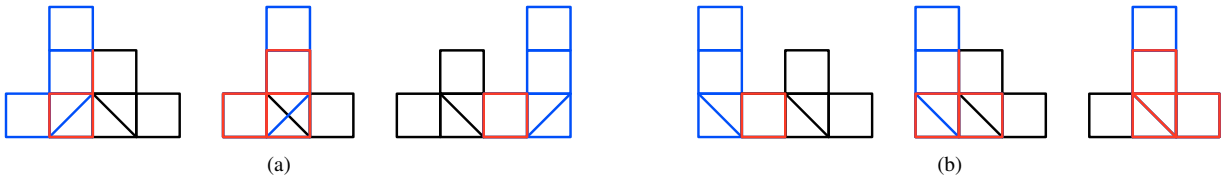

Fig. 5. Scanning the target structure over the source structure. black edges are trusses that belong to only G_1 and Blue edges are only for G_2. The overlapped edges (common subgraph) are highlighted by the red lines. (a) G_2 is scanned from left to right (b) Flipped G_2 is scanned over G_1

Algorithm 1 Scanning algorithm for the optimal matching. The algorithm returns the optimally merged graph G_m and the optimal matching M_m in G_m.

1: Make G_1 and G_2 cornered at the origin
2: **for** orientation $O_{G_2} \in \left[0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right]$ **do**
3: Rotate G_2 by O_{G_2} w.r.t z-axis
4: $X_1 = \max\{x(G_1)\}$, $Y_1 = \max\{y(G_1)\}$
5: $X_2 = \max\{x(G_2)\}$, $Y_2 = \max\{y(G_2)\}$
6: **for** $x_t \in [-X_2 \dots X_1 + X_2]$, $y_t \in [-Y_2 \dots Y_1 + Y_2]$ **do**
7: Transform G_2 by (x_t, y_t)
8: $n =$ number of the overlapped elements in $G_1 \cap G_2$
9: $G_3 = G_1 \cup G_2$
10: M = OptimalMatching$_{side}$ ($G_1 - G_2$, $G_2 - G_1$) + OptimalMatching$_{diagonal}$ ($G_1 - G_2$, $G_2 - G_1$)
11: **end for**
12: **end for**
13: $M_m = \text{argmin}_M(cost(M))$
14: $G_m = \text{argmin}_{G_3}(cost(M))$

general 3D trusses that meet our assumptions. The principle of operation in 3D is similar to 2D. First, G_2 is overlapped on G_1 from the left to the right as in Figure 5. Next, G_2 is flipped about z-axis and scanned again. Here the optimally merged graph is the right of Figure 5(b). In a 3D structure, the scan is over the x-y plane, and G_2 is rotated to account for four types of orientation. Matching is done separately for each edge type. The optimal matching is the sum of the optimal matchings for the side truss elements and the optimal matching for the diagonal truss elements using the optimally merged graph G_m.

Note that Algorithm 1 can be extended to trusses with a finite number of edge types. Optimal matching is done for each edge type and all the results are merged (line 10). Algorithm 1 can be applied even for a cases in which the initial truss represented by G_1 and the goal truss represented by G_2 are not identical in the type and number. When m_1 is greater than m_2, the algorithm chooses the optimal edges of G_1 to build G_2. If $m_2 > m_1$, the algorithm builds only parts of G_2 with the minimal cost from G_1.

Theorem 1: The running time of Algorithm 1 is $O(nm^4)$, where n is number of the nodes and m is number of the truss elements.

Proof: The computation required for comparing the graphs is $\mathrm{O(m)}$. The number of the scans is bounded by the xy-region of the graphs which can not be more than n_1+n_2. The optimal matching is computed by the Hungarian algorithm [18], which has $O(m^3)$ runtime. Note that computing the cost matrix for the Hungarian algorithm requires the execution of Dijkstra's algorithm $\mathrm{O(m)}$ times; however, the running time of the Hungarian algorithm dominates. ■

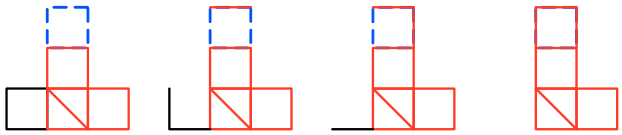

Fig. 6. Broken connectivity by performing the matching. The black trusses are $G1 - G2$ and The blue ones are $G2 - G1$. The black trusses move to the locations of the blue ones.

C. Maintaining Connectivity

Algorithm 1 yields the optimal matching for transforming G_1 into G_2 but does not provide a correct sequence of moves to ensure that the structure (e.g. all the truss elements) will stay connected at all time. Figure 6 illustrates some snapshots from performing the reconfiguration from G_1 to G_2 for the example of Figure 4. Note that one of the black truss elements has to be moved to the uppermost edge. However, this location is not reachable before the other truss elements move to their matched locations. Additionally, the next black truss loses its shortest path because the first black truss is gone.

To maintain connectivity and the shortest paths, an additional computational step is needed. The order of the obtained optimal matching needs to be analyzed and processed so that all truss elements stay connected at all times. If a target location is not reachable by any means, that truss element is moved along its trajectory to the farthest available intermediate truss vertex and temporarily stored there. That is, the paths for the truss elements are divided so that the elements may move to an intermediate point along the path until full connectivity to the target location is available. Practically, elements can be buffered by temporarily connecting them to a joint with a free connector.

Algorithm 2 describes the analysis and computation required in order to generate trajectories with intermediate storage locations that connectivity at all time for all truss elements. Let S be the set of the source truss elements of $G_1 - G_2$ and T be the set of the target edges for $G_2 - G_1$. Initially T is empty.

Algorithm 2 uses a dynamic graph G. G has all the edges of $G_1 \cup G_2$, however only the edges of G_1 are activated at the

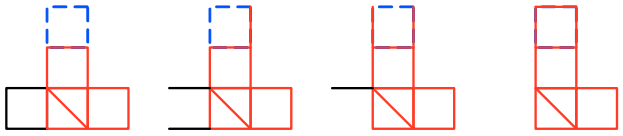

Fig. 7. The adjusted order of the matching in Figure 6.

start. As the computation proceeds, the set of active edges changes. The algorithm ends if all the $s_i \in S$ reach their target locations t_i. The algorithm chooses a truss element s_i whose target is connected to the current structure $S-T$. If the element does not belong to any paths of the other truss elements, it advances to its target. Otherwise, the algorithm picks another element the path of which includes the current element. If the two elements are the same trusses (diagonal or side), they exchange their target and adjust the paths according to the new targets. The exchange is reasonable since the two trusses are physically same and it does not hurt the optimality. If there is no same element of the same type among the path-overlapping elements, the algorithm searches for the deepest predecessor of the element and let it advance to its target. After the exchange, we repeat the process until $S = T$.

Algorithm 2 guarantees no queue when the structures are made of only a single type of the trusses, since it can always fill the picked edge with the deepest predecessor as long as no cycle in the paths. The queue is necessary only if the only predecessor has the different type. The concept of exchanging is also useful when the work is extended to a distributed system where many robots collaborate [1], [3].

Theorem 2: An edge s_i that is not in p_j $(j \neq i)$ can always be founded.

Proof: Suppose every s_i is in p_j $(j \neq i)$. Then at least a pair of paths p_i crosses each other, which means there is a loop in P. This is a contradiction because it means we can have a better matching by exchanging t_i with the intersection of the paths p_i or by cutting the loop. ■

Theorem 3: Algorithm 2 terminates.

Proof: By Lemma 2, the algorithm adds a part of P to the trajectory in every loop. Since P has a finite number of the paths, it will be completely traversed by S. ■

Figure 7 shows the adjusted paths of Figure 6. The black edges move to the blue edge locations without breaking connectivity. In this example, no source truss edge has been paused and added to the queue of elements with unreachable destination.

Algorithm 2 Exchange algorithm for trajectories to maintain the connectivity and the shortest paths

```
S = truss(G_1 − G_2)
T = edge(G_2 − G_1)
P = path(S → T) in G_m
deactivate T in G_m
Q = ∅
while S ≠ T do
  pick s_i such that t_i ∈ T − S is connected to S − T and p_i ∈ S − T
  trussSelected = false
  while not trussSelected do
    if s_i ∉ p_j (j ≠ i, j ∈ S − T) or s_i ∈ Q then
      move s_i along p_i
      delete p_i
      activate the edges that are connected to t_i
      pull out s_i from Q (if s_i ∈ Q)
      trussSelected = true
    else
      choose s_j such that s_i ∈ p_j
      if ∃s_j such that s_i ∈ p_j and type(s_i)=type(s_j) then
        exchange t_i and t_j
        p_i ← p_j(s_i → t_j)
        p_j ← p_j(s_j → s_i) + p_i(s_i → t_i)
        i ← j
      else
        pick s_k, the deepest predecessor of s_j
        move s_k along p_k
        if s_k ≠ t_k then
          s_k → Q
        end if
        trussSelected = true
      end if
    end if
  end while
end while
```

IV. Results

We have implemented the algorithms described in Section III and we evaluated them on six 3D canonical structures. In this section we describe the results.

A. Solution examples

Figure 8 shows an example set of truss structures[16]. The left structure consists of 6 cubes connected as a compact structure. The others are like a tower. G_1 to G_5 has a total of 83 truss elements, 31 of which are diagonal and the rest side. G_6 has only 75 truss elements. Figure 9 and 11 show how the planning algorithms in this paper morph G_1 into G_3 and G_6 for this example. Figure 10 shows transformation G_1 to G_5 with a programmatically generated animation. Between individual reconfiguration steps, the robot uses Dijkstra's algorithm to plan the shortest path to the start point of its next reconfiguration step. The exchange algorithm is used to generate the paths. Fortunately, the structure does not require a queue. Note that G_1 can transform to G_6 in the optimal way even though the numbers of the trusses are different.

B. Performance analysis

Table I summarizes the performance of the algorithm for all combinations of the given truss geometries. The source structure is successfully transformed into the target structure as minimizing the total displacements of the truss elements.

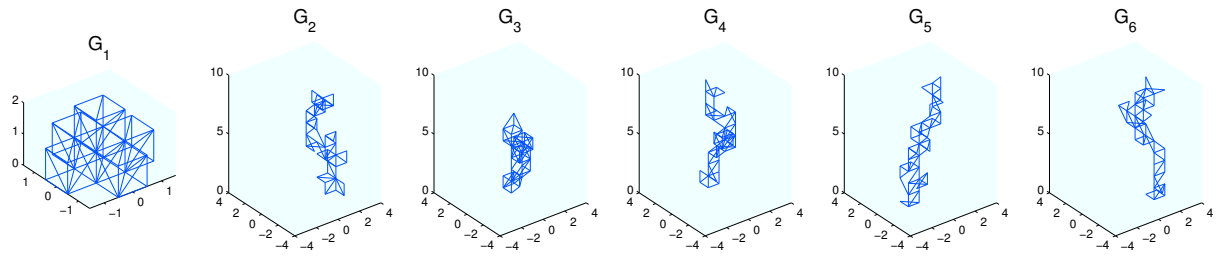

Fig. 8. 6 truss structures. Each structure has 83 trusses except for G_6 that has only 75 trusses[16].

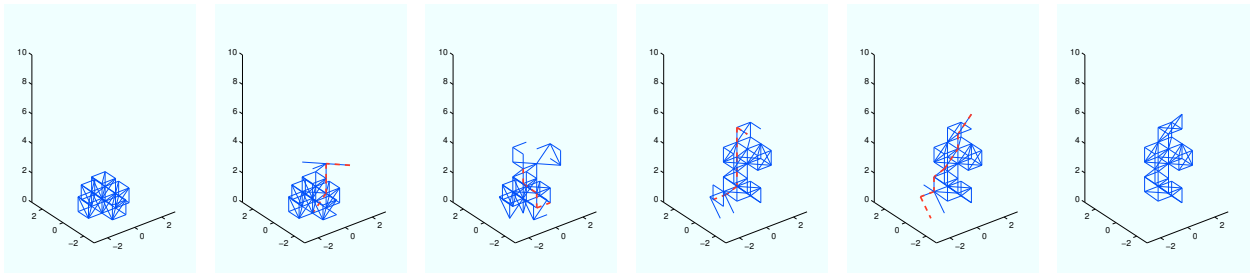

Fig. 9. Morph from G_1 to G_3. Thickness of a truss element denotes number of the queued truss elements at that location. The red dotted line is a path of the currently moving truss elements. 62 truss elements move to their matched edge in 257 steps.

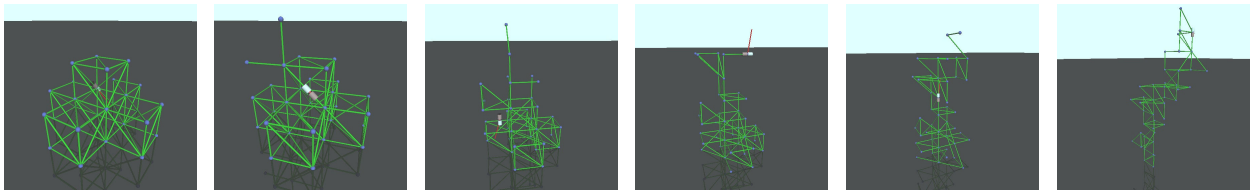

Fig. 10. Morph from G_1 to G_5. The figures are programmatically generated animation depicting a structural reconfiguration planned by the algorithm.

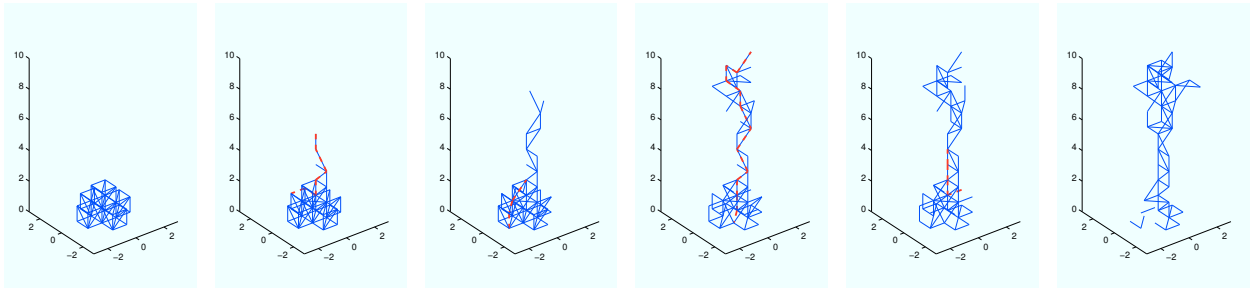

Fig. 11. Morph from G_1 to G_6. 63 trusses move to the matched edge with 427 steps. G_1 has more trusses and there are 8 remaining trusses.

We do not need any queue for the given structures. In the future, the required conditions for no queue will be considered.

On the other hand, the cost can be considered as time required for a robot to finish the transformations. In future, an algorithm for multi-robot processing may reduce the times by parallelism.

V. Conclusions

This paper presented and demonstrated an algorithm that solves the problem of how to optimally transform a given truss structure into another structure, piece by piece, subject to maintaining a variety of constraints. The algorithm is realizable using a future robot capable of traversing a truss and executing element removal and insertion operations at desired locations. We have also presented initial designs for such a robot, as well as physical elements that can be removed and inserted at any valid truss location.

Because of the regular nature of the truss lattice, the algorithms presented remain tractable in the size of the source and target graphs. Similarly, capabilities of the robot, such as the type of joints it can traverse and the number

TABLE I

RESULT OF THE SIMULATIONS

Matching	Moving trusses	Cost	Total Q
$G_1 \rightarrow G_2$	68	418	0
$G_1 \rightarrow G_3$	62	257	0
$G_1 \rightarrow G_4$	70	462	0
$G_1 \rightarrow G_5$	72	449	0
$G_1 \rightarrow G_6$	63	427	0
$G_2 \rightarrow G_3$	62	220	0
$G_2 \rightarrow G_4$	68	158	0
$G_2 \rightarrow G_5$	63	128	0
$G_2 \rightarrow G_6$	49	122	0
$G_3 \rightarrow G_4$	60	249	0
$G_3 \rightarrow G_5$	58	240	0
$G_3 \rightarrow G_6$	55	248	0
$G_4 \rightarrow G_5$	69	139	0
$G_4 \rightarrow G_6$	56	145	0
$G_5 \rightarrow G_6$	56	135	0

elements it can carry in one pass greatly affect the type and performance of the algorithm that can be achieved. Various aspects of the truss elements design also impact the algorithm performance, such as the ability to remove and insert elements at random access, and structural the ability of cantilevered elements to sustain the load of a working robot.

The coupled algorithmic and hardware centered nature of this problem open the door to many future challenges and opportunities. In particular, we are interested in performance of this system when multiple robots operate on the structure simultaneously, and when knowledge about the current global state of the structure, resource availability and inter-robot communications are imperfect. In addition, practical considerations related to structural integrity, stress, and vibration, as well as assembly failures and non-regular lattices are also of interest. An additional interesting challenge is the inverse problem of designing useful target structures that a given structure can be easily transformed into [16].

VI. ACKNOWLEDGEMENTS

This work was done as a collaboration between the DRL group at MIT and the Cornell Computational Synthesis Lab as part of the U.S. National Science Foundation, Emerging Frontiers in Research and Innovation (EFRI) grant #0735953. Seung-kook Yun is supported in part by Samsung Fellowship. We are grateful for this support. We are also grateful to the groups of Eric Klavins and Mark Yim for insightful discussions on reconfiguration. Support for this work was also provided in part by Intel. Thanks to Daniel Lobo for providing the source and target truss data.

REFERENCES

[1] C. Detweiler, M. Vona, Y. Yoon, S. kook Yun, and D. Rus, "Self-assembling mobile linkages," *IEEE Robotics and Automation Magazine*, vol. 14(4), pp. 45–55, 2007.

[2] S. kook Yun and D. Rus, "Self assembly of modular manipulators with active and passive modules," in *Proc. of IEEE/RSJ IEEE International Conference on Robotics and Automation*, May 2008, pp. 1477–1482.

[3] S. kook Yun and D. Rus, "Optimal distributed planning for self assembly of modular manipulators," in *Proc. of IEEE/RSJ IEEE International Conference on Intelligent Robots and Systems*, Nice, France, Sep 2008, pp. 1346–1352.

[4] M. Csete and J. Doyle, "Bow ties, metabolism, and disease," *Trends in Biotechnology*, vol. 22, pp. 446–450, 2004.

[5] K. D. Kotay and D. L. Rus, "Navigating 3d steel web structures with an inchworm robot," *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, vol. 1, pp. 1178–1185 vol.2, 1996.

[6] Z. M. Ripin, T. Soon, A. B. Abdullah, and Z. Samad, "Development of a low-cost modular pole climbing robot," in *TENCON 2000*, vol. 1, 2000, pp. 196–200 vol.1.

[7] M. Tavakoli, M. Zakerzadeh, G. Vossoughi, and S. Bagheri, "A hybrid pole climbing and manipulating robot with minimum DOFs for construction and service applications," *Industrial Robot: An International Journal*, vol. 32, no. 2, pp. 171–178, 2005.

[8] H. Amano, K. Osuka, and T. J. Tarn, "Development of vertically moving robot with gripping handrails for fire fighting," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 2001, pp. 661–667 vol.2.

[9] R. Aracil, R. Salateru, and O. Reinoso, "A climbing parallel robot," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 1, pp. 16–22, 2006.

[10] C. Balaguer, A. Gimnez, J. Pastor, V. Padrn, and M. Abderrahim, "A climbing autonomous robot for inspection applications in 3D complex environments," *Robotica*, vol. 18, no. 03, pp. 287–297, 2000.

[11] Y. Yoon and D. Rus, "Shady3d: A robot that climbs 3d trusses," *Robotics and Automation, 2007 IEEE International Conference on*, pp. 4071–4076, April 10-14 2007.

[12] M. Nechyba and Y. Xu, "Human-robot cooperation in space: SM^2 for new spacestation structure," *Robotics & Automation Magazine, IEEE*, vol. 2, no. 4, pp. 4–11, 1995.

[13] P. J. Staritz, S. Skaff, C. Urmson, and W. Whittaker, "Skyworker: a robot for assembly, inspection and maintenance of large scale orbital facilities," in *IEEE International Conference on Robotics and Automation (ICRA 2001)*, vol. 4, 2001, pp. 4180–4185 vol.4.

[14] S. Skaff, P. Staritz, and W. Whittaker, "Skyworker: Robotics for space assembly, inspection and maintenance," *Space Studies Institute Conference*, 2001.

[15] D. Hjelle and H. Lipson, "A robotically reconfigurable truss," in *ASME/IEEE International Conference on Reconfigurable Mechanisms and Robots (ReMAR 2009)*, 2009.

[16] D. Lobo, D. Hjelle, and H. Lipson, "Reconfiguration algorithms for robotically manipulatable structures," in *ASME/IEEE International Conference on Reconfigurable Mechanisms and Robots (ReMAR 2009)*, 2009.

[17] *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.

[18] H. Kuhn, "The hungarian method for the assignment problem," *Naval Res. Logist. Quart.*, vol. 2, pp. 83–97, 1955.