This accepted article to IROS 2019 is made available by the authors in compliance with IEEE policy.
Please find the final, published version in IEEE Xplore, DOI: 10.1109/IROS40897.2019.8967640.

Reconfiguration Motion Planning for Variable Topology Truss

Chao Liu, and Mark Yim

***Abstract*— This paper presents an algorithm to do motion planning for a new class of self-reconfigurable modular robot: the variable topology truss (VTT). Modular robots consist of many modules that can be configured into various structures, and motion planning problem for modular robots with many degrees of freedom and many motion constraints is a significant challenge. In this paper, we propose a novel motion planning algorithm for modular robots to handle this problem with huge state space inspired by DNA replication process — the topology of DNA can be changed by cutting and resealing strands as tanglements form. In a variable topology truss, a single node with enough edge modules can split into a pair of nodes and two separate nodes can be merged to become an individual one. This self-reconfiguration ability results in more potential applications for this type of robots in unstructured environment, such as space and underseas but also leads to more challenges for reconfiguration planning. A novel way to model the robot in a nonuniform grid space is presented and a simple local planner is also developed to check the validation of possible actions. This approach significantly simplifies the problem and some experiment results show that the complicated problem can be solved in a reasonable time.**

I. INTRODUCTION

Self-reconfigurable modular robots are composed of many repeated building blocks (modules) from a small set of types. Often all modules have uniform docking interfaces that allow transfer of mechanical forces and moments, electrical power, and communication throughout the robot [1]. One type of modular robotic system is a reconfigurable truss structure. One class of truss robots is commonly called a variable geometry truss (VGT) [2], in which the truss members have variable length, such as TETROBOT [3]. A variable *topology* truss (VTT) is similar to VGT with additional capability to self-reconfigure the attachment of members at the nodes, changing its topology by merging or splitting nodes [4]. Two separate nodes in the truss can dock to form one node which connects all of the involved members. Similarly, a single node with a sufficient amount of members can undock into a pair of nodes. Hence, the variable topology truss system has both the efficiency benefits of VGTs and the flexibility of self-reconfigurable robots [5].

An example application using VTT structures aims to build a robot system that can be deployed into a disaster scenario shown in Fig. 1 [4]. The robot is mobile (forming truss legs, or tumbling [6]) and can move into buildings and reconfigure into large support structures to reinforce the building, shoring to prevent further collapse as first responders search for victims. This application exploits the inherent large strength to weight ratio of truss structures.

The authors are with GRASP Lab and Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104, USA chaoliu@seas.upenn.edu, yim@seas.upenn.edu This work was sponsored by AFOSR grant FA2386-17-1-4656

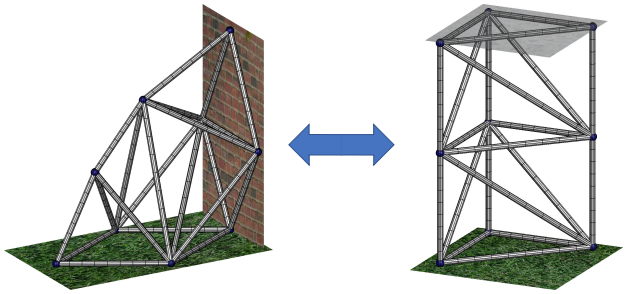

Fig. 1. A variable topology truss can reconfigure itself into different configurations with respect to different tasks.

We define a *configuration* to be the set of truss member link lengths and their node assignments. The assignments of links to nodes defines the topology and link lengths define the shape. The output of a *reconfiguration motion planner* is the sequence of actions to transform the robot from a start configuration to a goal configuration, e.g. Fig. 1.

VTT systems are composed of edge modules which include both the linear actuator and two ends that have the ability to dock onto other ends [4]. The end positions fully determine the state of the system. Each configuration has constraints on the arrangement of edge modules to guarantee controllability. This includes making sure the topology of the modules nominally form a statically determinate truss. This means every node will have at least 3 members attached (each node is of degree 3 or higher). Because these structures have multiple loop constraints, we cannot map actuator DoF to system DoF. Instead it becomes more convenient to map 3 DoF to node translation positions and back out the required truss member lengths to achieve that position. Another side effect of these constraints are that reconfigurable VTT systems have at least 18 members [4] (18 actuated DoF), thus motion planning for these systems can have very high dimension.

Since topological reconfigurations also happen at nodes, we consider all actions to occur at nodes. For example, if a set of edge modules E_1 intersect at node v_1, and another set of edge modules E_2 intersect at node v_2, then the states of these edge modules can be changed by manipulating node v_1 and v_2, including changing the locations of them, merging v_1 and v_2 or splitting if v_1 and v_2 are already merged. For some simple cases, translating the nodes small distances in the space can happen without collision. However, in many instances translating nodes will often lead to the truss members colliding with other truss members. In many

of those cases, there may not be a path for that node to reach a goal location without collision given the topology of the configuration. Traditional motion planning methods would require analysis of the motion, geometric reasoning of the topological spaces in which collision can occur and reconfiguration planning to change the topology if required.

This paper presents a novel method of motion planning combined with limited reconfiguration to enable motions of nodes without regarding to internal collision via reconfiguration. This behavior is similar to DNA replication in which topoisomerase can change the topology of DNA by cutting and resealing strands as tanglements form [7]. This paper presents an algorithm inspired by DNA motion to solve this fundamental reconfiguration problem.

The rest of the paper is organized as follows. Sec. II reviews relevant and previous works. Sec. III introduces the variable topology truss configuration and formalize the problem. Sec. IV discusses the algorithm, including how the space is discretized, the transition model and actions, collision check method and graph search algorithm. Some experiment results are shown in Sec. V. Finally, Sec. VI talks about the conclusions and future work.

II. RELATED WORK

The variable topology truss was first introduced in [8]. The hardware concepts and challenges associated with the planning, control and implementation were discussed. The reconfiguration ability of the system was analyzed in [4]. Then some useful techniques for reconfiguration planning of variable topology system was discussed in [5] and a modified retraction-based RRT algorithm was designed for VTT reconfiguration planning. However, this still needs to handle a huge and complicated configuration space with high dimensions when moving a node around and intermediate configurations with different topology have to be specified manually when topology reconfigurations involved.

A divide-and-conquer approach for reconfiguartion planning of modular robots was published by Casal and Yim [9]. Two algorithms are presented which are based on graph representation of configurations and a substructure set resulting in a hierarchy construction of initial and goal configurations. There are also other graph-based algorithms developed for different chain-type modular robots in [10] [11] [12] and [13]. However, these algorithms are not applicable to VTT's because they do not account for the common case above where node motions require reconfiguration to avoid self-collision. [14] presented a distributed reconfiguration planning algorithm based on some concepts and the bottom-up algorithm to compute maximum common subconfiguration (MCS) in [15]. This algorithm relies on the motion capability of each individual module which is not applicable to VTT.

A shape morphing algorithm for linear actuator robots (LARs) is presented in [16]. This system is made up of linear actuators connected at universal joints into a network. [17] presented a shape morphing idea for VTT based on a new approach to compute the collision-free configuration space for nodes. Our work differs from these work in that the motion involved is not just changing the geometry shape of the structure but also the robot graph topology.

This paper presents a new motion planning framework for variable topology truss, with which the robot can change its shape autonomously and quickly, including *geometry reconfiguration action* which only moves the node positions and *topology reconfiguration action* which has to split or merge nodes. A new method to discretize the space nonuniformly is proposed so that discrete actions can be applied for a motion task and the space can be explored efficiently. This method is the combination of exact cell decomposition method and approximation method. The cells are determined based on truss geometry and then are subdivided into octants. Different from common octree decomposition [18], cells are not necessarily to be cubic. Collision-free actions can be computed efficiently and a sequence of optimal reconfiguration actions are generated by simple graph search algorithm. We demonstrate this algorithm on some motion tasks and necessary analysis is provided.

III. ROBOT CONFIGURATION

A variable topology truss can be viewed as a special type of parallel robot — constructed by linear actuators connected at special node joints. Hence, the structure of a variable topology truss can be modeled as an undirected graph $G = (V, E)$ where V is the set of vertices of G and E is the set of edges of G: each member can be regarded as an undirected labeled edge $e \in E$ of the graph and every intersection among members can be treated as a vertex $v \in V$ of the graph. Every $v \in V$ has two properties: `ID` and `Pos` where `ID` is used to label the vertex and `Pos` is used to define the Cartesian coordinates of the vertex namely $v[\texttt{Pos}] = [v_x, v_y, v_z]^\intercal \in \mathbb{R}^3$. In this way, the state of a member is fully defined by `Pos` properties of its two vertices written as $e = (v_1, v_2)$ where v_1 and v_2 are two vertices of edge e.

Since two separate nodes in VTT can dock to form one single node which connects all of the involved members and a single node can undock into a pair of nodes [4] (as long as there are no less than six members) shown in Fig. 2, the number of nodes can change. However, the number of members in a system are physical elements which cannot merge or disappear so that the number must remain constant.

In this paper, the fundamental reconfiguration problem can be stated. For a variable topology truss $G = (V, E)$, the goal is to change the state of edge module $e \in \hat{E}$ from its

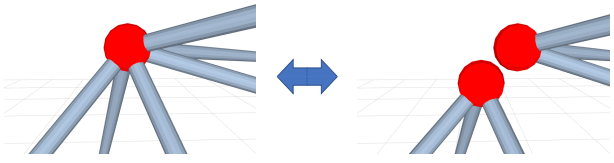

Fig. 2. A single node with six members can be splitted into a pair of nodes and two separate nodes can also merge into a single node.

initial state to its goal state in which $\hat{E} \subset E$. However, as the discussion above, edge modules have to be controlled in form of a group because their states can only be modified by manipulating the involved nodes. For any $e = (v_1, v_2) \in E$ in a variable topology truss configuration $G = (V, E)$, let $v_1\,[\texttt{Pos}] = e\,[v_1]$ and $v_2\,[\texttt{Pos}] = e\,[v_2]$, then the current state of this edge module e is fully defined by $e\,[v_1]$ and $e\,[v_2]$. Assume $\forall e \in \hat{E}$ intersect at node $\hat{v}$, then the motion task is, $\forall e \in \hat{E}$, change $e\,[\hat{v}]$ from $[x_i, y_i, z_i]^\intercal$ to $[x_g, y_g, z_g]^\intercal$.

IV. MOTION PLANNING ALGORITHM

A. Grid Space Model

Planning in discrete space makes it possible to efficiently plan a sequence of discrete actions for a complicated motion task. However, when discretizing space, discretization resolution is an important issue. There is a trade-off for different discretization resolution: too fine resolution results in a large search space — too coarse resolution may bring about no solution for a motion task even if there exists a valid path. This problem is especially significant for modular robots, such as a variable topology truss, which may have high dimensional spaces. Here we present a novel way to represent the system in a discretized workspace for a given variable topology truss which we call the *grid space*.

The automatically generated grid space has a stepsize that adapts depending on the density of the space. A small exploration stepsize is used for high occupancy subspace and large exploration stepsize for low occupancy subspace. The density is a function of the node positions of both the initial variable topology truss configuration and a goal variable topology truss configuration. Let V be the set containing all nodes with unique locations and its size is N. Firstly all of these node locations are extracted in x-axis, y-axis and z-axis respectively:

$$X = \{v_x | v \in V\}$$
$$Y = \{v_y | v \in V\}$$
$$Z = \{v_z | v \in V\}$$

Sort X, Y and Z in nondecreasing order to generate three sequences $\hat{X}$, $\hat{Y}$ and $\hat{Z}$ respectively. For each sequence, if two adjacent elements are the same or very close, keep only one element. Use the parameter δ to set the threshold for closeness. Note that δ will often be some fraction of the size of the truss. Then, for every two adjacent elements (e.g. x_i and x_j), insert a midpoint between them (e.g. $(x_i + x_j)/2$) into the list as an intermediate position. This will subdivide each cell in space into octants like octree decomposition. With the final three sequences generated as follows:

$$\hat{X} = \{x_0, x_1, \cdots, x_{N_x}\}$$
$$\hat{Y} = \{y_0, y_1, \cdots, y_{N_y}\}$$
$$\hat{Z} = \{z_0, z_1, \cdots, z_{N_z}\}$$

the following conversions from grid space to Cartesian space in x-axis, y-axis and z-axis can be obtained respectively:

$$f_x(i_x) = x_{i_x} \quad i_x = 0, 1, \cdots, N_x \tag{2a}$$

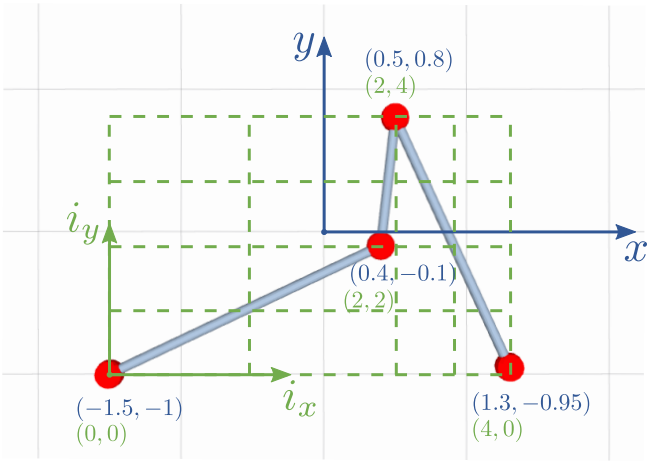

Fig. 3. Two dimensional example with four nodes and three members with $\delta = 0.1$. The generated grids are shown with "- -" and coordinates in Cartesian space are in dark blue color and coordinates in grid space are in light green color.

$$f_y(i_y) = y_{i_y} \quad i_y = 0, 1, \cdots, N_y \tag{2b}$$
$$f_z(i_z) = z_{i_z} \quad i_z = 0, 1, \cdots, N_z \tag{2c}$$

the inverse conversions (from Cartesian space to grid space) are:

$$f_x^{-1}(x) = \underset{i_x \in \{0,1,\cdots,N_x\}}{\arg\min} |x - x_{i_x}| \tag{3a}$$

$$f_y^{-1}(y) = \underset{i_y \in \{0,1,\cdots,N_y\}}{\arg\min} |y - y_{i_y}| \tag{3b}$$

$$f_z^{-1}(z) = \underset{i_z \in \{0,1,\cdots,N_z\}}{\arg\min} |z - z_{i_z}| \tag{3c}$$

Then the conversion from grid space to Cartesian space can be defined as

$$f([i_x, i_y, i_z]^\intercal) = \left[x_{i_x}, y_{i_y}, z_{i_z}\right]^\intercal \tag{4}$$

and the conversion from Cartesian space to grid space can be defined as

$$f^{-1}([x, y, z]^\intercal) = \left[f_x^{-1}(x), f_y^{-1}(y), f_z^{-1}(z)\right]^\intercal \tag{5}$$

Note that this is a nonuniform grid space. Given a variable topology truss configuration $G = (V, E)$ in Cartesian space, there is a corresponding variable topology configuration $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in grid space. A two dimensional example is shown in Fig. 3. Both Cartesian space and grid space coordinates are shown. It is apparent that, given a conversion between Cartesian space and grid space, different VTT configurations in Cartesian space may results in the same VTT configuration in grid space.

A truss example is shown in Fig. 4a and the equivalent truss in grid space is shown in Fig. 4b. In Cartesian space, the truss is a slightly deformed cube. The mapping in grid space is a regular cube with eight cells generated inside the cube.

Note that the non-uniform resolution is different than octree decompositions where a subdivision results in cells that are equally divided into cubes. Here the x-axis y-axis

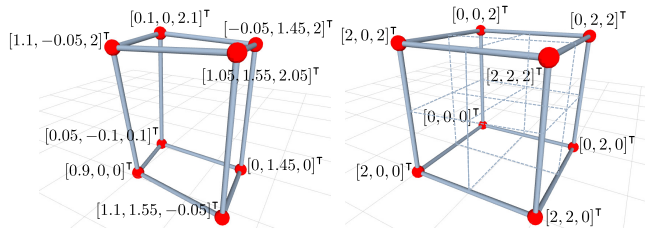

Fig. 4. (a) A Truss in Cartesian Space and (b) The Equivalent Cubic Truss in Grid Space with $\delta = 0.2$

and z-axis divisions may not be uniform. In addition, since the divisions are derived directly on the location of nodes, there are fewer subdivisions than those may occur in octree where the number of subdivisions can become large if the nodes are slightly off from an even division.

Once the conversion between Cartesian space and a grid space is computed, a VTT configuration can be simplified — if two nodes are very close and are third degree neighbors or higher, those nodes can be merged into one. This is determined by the parameter δ. There are some constraints for δ: Its value cannot be too large since the edge modules have a non-zero minimum length.

B. Node Motion Model and Reconfiguration Actions

Even though actuators are located in the truss members, it is not convenient to do motion planning in the joint space of those actuators. Instead it is more convenient to do motion planning from the viewpoint of node motions, doing the inverse kinematics to determine what link lengths will yield those node positions. This is similar to parallel robot motion planning where inverse kinematics is much easier than the forward kinematics.

After the nonuniform grid space is computed, we can model the motion of a node as discrete actions. The discrete action can be a geometric reconfiguration action or a topology reconfiguration action.

In grid space, we define 27 different possible discrete geometric motion actions for a free node shown in Fig. 5. The cube centered on the free node can be defined by its

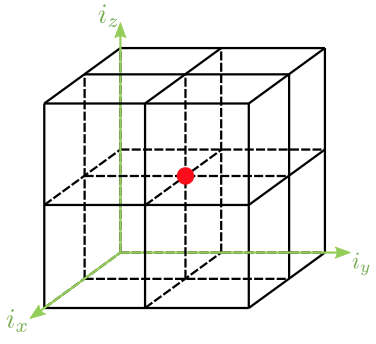

Fig. 5. In grid space, a node ● can move to 27 different locations with one discrete action, defined to be the 27 points of intersection between lines in the figure including the center of the cube which is the no motion action.

two corners: $[0, 0, 0]^\intercal$ and $[2, 2, 2]^\intercal$. Note that the cube in this grid space is not necessarily a cubic shape in Cartesian space.

For a node in a variable topology truss, there are some constraints on the motion that the node can execute. Every node has to be attached to at least three members in order to maintain controllability. If a node is controlled by six or more members, this node can be split into a pair of controllable nodes with multiple ways to split the involved members. For example, if a node is controlled by six members, there are 10 different ways to split six members into two groups of three. However, if a node is controlled by five members, then there is no way to split this single node as both nodes need to have at least three members.

We can exploit the reconfigurability of nodes to ease the collision free motion planning problem. If a node moves to a location in grid space that is already occupied by a node, then the corresponding moving action will end up with a merge action. In this way, we do not need to worry about collisions of nodes. For the VTT system, there are three possible atomic actions for a specific node: `Move`, `Merge` and `Split`. Since only one node may exist in a discrete location in grid space, there are four different combinations of atomic motion for a node:

$$a = \begin{cases} \texttt{Move} \\ \texttt{Split} + \texttt{Move} \\ \texttt{Move} + \texttt{Merge} \\ \texttt{Split} + \texttt{Move} + \texttt{Merge} \end{cases} \tag{6}$$

in which we define a to be the *discrete reconfiguration action*. Note that not all actions are executable because some actions may violate constraints resulting in a not valid motion.

As the reconfiguration process proceeds, some nodes may disappear and some nodes may appear. Let $\hat{V} \subset V$ contain the current nodes intersected by edge modules in $\hat{E}$. Initially, $\hat{V} = \{\hat{v}\}$ because all edge modules in $\hat{E}$ intersect at node $\hat{v}$. When the discrete reconfiguration action occurs, $\hat{V}$ may change accordingly. For example, if $\hat{v}$ is split into $\hat{v}'$ and $\hat{v}''$, then $\hat{V} = \{\hat{v}', \hat{v}''\}$, or if $\hat{v}$ is merged with another node $\bar{v}$, then $\hat{V} = \{\bar{v}\}$. For each node in $\hat{V}$, the same reconfigurability analysis can be applied to generate all possible actions and the states of involved edge modules in set $\hat{E}$ will be changed accordingly. To minimize the search space for all possible actions, in this work, only one node is active, which means if there are multiple nodes in $\hat{V}$, apply action a for only one node.

C. Collision

Each node is controlled by multiple members and, in fact, this can be modeled as a parallel robot. When controlling a node to execute a motion action a, we are actually controlling a parallel robot to move in a complex environment occupied by many other edge modules. It is not straightforward to compute the collision-free space for this parallel robot directly but we can exhaustively check collision for every pair of members during the motion. For example, for a VTT configuration $G = (V, E)$, node $v \in V$ is going to execute

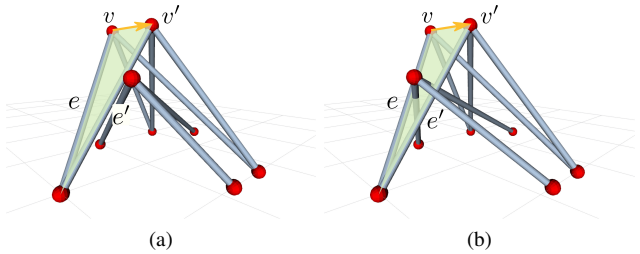

Fig. 6. Light green triangle (△) is sweeped by member e when moving node v to new location v' along the yellow (⟶) trajectory and the new position of member e is e'. e doesn't collide with any other members in (a) but does collide with two members in (b) during the motion.

an action a, and the involved member set is E^v. We need to check whether any $e \in E^v$ will collide with any other edge module in set $E \backslash E^v$ during the motion action process. Every edge module can be modeled as a line segment in space, thus, for every $e \in E^v$, naïvely we can just check the intersection between a moving line segment and a still line segment. This is actually not easy. With our nonuniform grid space and node motion model, the node is actually moving along a line segment. Hence, we present an easier and more efficient method to do collision check.

For a VTT configuration $G = (V, E)$ and a node $v \in V$, once a reconfiguration action a is executed, every moving member in E^v sweeps a triangle area (Fig. 6a), and if this member $e \in E^v$ collides with another member $\bar{e} \in E \backslash E^v$, then $\bar{e}$ must intersect with the triangle $\triangle$ generated by e (Fig. 6b). There are two cases: $\bar{e}$ is not parallel to $\triangle$ and $\bar{e}$ is parallel to $\triangle$. For the first case, there are already many efficient algorithm to test the intersection between a line segment and a triangle in 3D space, such as Möller-Trumbore ray-triangle intersection algorithm [19]. For the second case, it is just a simple 2D geometry math problem.

D. Transition Model

A transition model is required to describe the effect of a reconfiguration action on a VTT configuration. For a VTT configuration $G = (V, E)$, given a discrete reconfiguration action a, a new VTT configuration $G' = (V', E')$ is able to be computed by transition model $\mathcal{F}(G, a)$ if action a is executable.

The transition model process is shown in Fig. 7. Given a VTT configuration $G = (V, E)$ and a reconfiguration task, the grid space can be computed as described in Sec. IV-A. A corresponding VTT configuration $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in this grid space can be computed, where for each $v \in V$, its corresponding position in grid space $\mathrm{v}[\texttt{Pos}]$ can be computed by Eq. (5). Then we can apply the discrete action a on the corresponding node of the VTT configuration $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in grid space to obtain the new states of all involved edge modules. With Eq. (4), the new position of the corresponding node (the intersection of these involved edge modules) can be calculated. Constraints need to be satisfied in Cartesian space. Only collision-free criteria is considered here but it is straightforward to add more constraints. An action satisfying all constraints is an *executable reconfiguration action*. There are four different types of reconfiguration actions shown in Eq. (6). For geometry reconfiguration actions, just apply the model in Sec. IV-C. For actions containing topology reconfiguration actions, apply collision check for only `Move` action because `Split` and `Merge` never cause any collision. Once the action is verified to be executable, the new VTT configuration in Cartesian space is obtained.

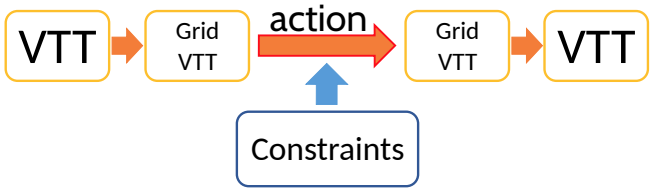

Fig. 7. Transition Model Diagram

E. Graph Search Algorithm

With the transition model, it is straightforward to do motion planning by graph search algorithms. Two VTT configurations can be connected by an edge if and only if there is an executable reconfiguration action taking the first VTT configuration to the second VTT configuration. We can start from the initial VTT configuration and stop until the goal configuration is visited. A graph search algorithm designed based on A^* framework is shown in Algorithm 1.

Line 1 — 5: Compute the nonuniform grid space and the equivalent initial VTT configuration and goal VTT configuration in this grid space. Also make two sets $\mathcal{Q}$ and $\bar{\mathcal{Q}}$ where $\mathcal{Q}$ contains all newly computed VTT configurations or non-visited VTT configurations in grid space and $\bar{\mathcal{Q}}$ contains all visited VTT configurations in grid space. The size of these two sets will change as the algorithm explores the configuration space. It is not feasible to compute all possible VTT configurations as the number grows exponentially in the number of members. At the beginning, only initial VTT configuration and goal VTT configuration are in set $\mathcal{Q}$ and the algorithm starts with the initial configuration. The value $g(\mathcal{G})$ is the cost of the path from $\mathcal{G}_i$ to a VTT configuration $\mathcal{G}$, so $g(\mathcal{G}_i) = 0$ and $g(\mathcal{G}_g) = \infty$ at the beginning.

Line 7 — 10: Every iteration always starts with the VTT configuration with the smallest cost $r(\mathcal{G})$ in set $\mathcal{Q}$ and $r(\mathcal{G}) = g(\mathcal{G}) + h(\mathcal{G})$ where $h(\mathcal{G})$ is a heuristic function of configuration $\mathcal{G}$ that estimates the cost of the cheapest path from $\mathcal{G}$ to $\mathcal{G}_g$. The heuristic value can be related to the distance from the current location to the goal location of every moving node. In the beginning, the initial VTT configuration has the smallest cost. Then update set $\mathcal{Q}$ and $\bar{\mathcal{Q}}$. The connection information among VTT configurations are not fully known, thus we need to try all possible actions for these moving members to create connections. Recall that only one node is active for each reconfiguration step, so the total number of all executable reconfiguration actions is not large and the worst case is $C^3_{|\hat{\mathcal{E}}|}$ (the total number of all combinations of $|\hat{\mathcal{E}}|$ elements in $\hat{\mathcal{E}}$ taken three at a time) for edge set $\hat{\mathcal{E}}$.

Algorithm 1: Graph Search Algorithm

Input: Initial VTT configuration $G_i = (V_i, E_i)$, edge set $\hat{E}$, the intersection node $\hat{v}$ and goal state of $e\,[\hat{v}]$ in which $\forall e \in \hat{E}$

Output: Tree of VTT configuration graphs

Compute nonuniform grid space;
Compute corresponding VTT configurations $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ and $\mathcal{G}_g = (\mathcal{V}_g, \mathcal{E}_g)$ in grid space and the corresponding edge set $\hat{\mathcal{E}}$ with $\forall e \in \hat{\mathcal{E}}$ intersecting at node $\hat{v}$;
$\mathcal{Q} = \{\mathcal{G}_i, \mathcal{G}_g\}$;
$\bar{\mathcal{Q}} = \emptyset$;
$g(\mathcal{G}_i) \leftarrow 0$, $g(\mathcal{G}_g) \leftarrow \infty$;
while $\mathcal{G}_g = (\mathcal{V}_g, \mathcal{E}_g) \in \mathcal{Q}$ **do**
  $\bar{\mathcal{G}} \leftarrow \arg\min_{\mathcal{G} \in Q} r(\mathcal{G}) = \arg\min_{\mathcal{G} \in Q}(g(\mathcal{G}) + h(\mathcal{G}))$;
  $\mathcal{Q} \leftarrow \mathcal{Q} \backslash \bar{\mathcal{G}}$;
  $\bar{\mathcal{Q}} \leftarrow \bar{\mathcal{Q}} + \{\bar{\mathcal{G}}\}$;
  Compute action set A containing all possible executable reconfiguration actions for set $\hat{\mathcal{E}}$ in grid space;
  for $\forall a \in A$ **do**
    $\mathcal{G} = \mathcal{F}(\bar{\mathcal{G}}, a)$;
    if $\mathcal{G}(\mathcal{V}, \mathcal{E}) \notin \bar{\mathcal{Q}}$ **then**
      if $\mathcal{G}(\mathcal{V}, \mathcal{E}) \in \mathcal{Q}$ **then**
        if $g(\bar{\mathcal{G}}) + c(a) < g(\mathcal{G})$ **then**
          $g(\mathcal{G}) \leftarrow g(\bar{\mathcal{G}}) + c(a)$;
          $p(\mathcal{G}) \leftarrow \bar{\mathcal{G}}$
        end
      end
      else
        $\mathcal{Q} \leftarrow \mathcal{Q} + \{\mathcal{G}\}$;
        $g(\mathcal{G}) \leftarrow g(\bar{\mathcal{G}}) + c(a)$;
        $p(\mathcal{G}) \leftarrow \bar{\mathcal{G}}$
      end
    end
  end
end
return p

Line 11 — 26: For every executable action $a \in A$, we can obtain a new connection with another VTT configuration by transition model. If this VTT configuration has been visited, then this connection is not a new connection. If not, then there are two cases: this configuration is not a newly found configuration which means there is already a connection between this configuration and another VTT configuration, or this configuration is a new one which has no connection before. For the first case, we need to check whether its value needs to be updated. $c(a)$ is the cost of the current action a. We should set the topology reconfiguration action cost a relative higher value because, for modular robots, docking and undocking are usually difficult. If its value is updated, then its parent $p(\mathcal{G})$ is also updated accordingly. For the second case, initialize the value and parent of this new VTT configuration, and update set $\mathcal{Q}$.

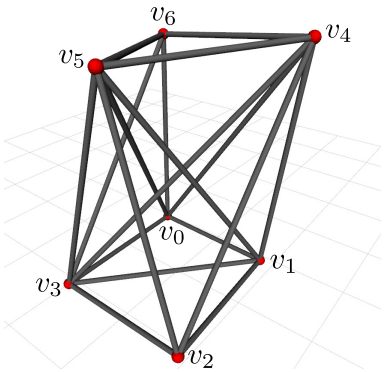

Fig. 8. Initial VTT Configuration

Once $\mathcal{G}_g = (\mathcal{V}_g, \mathcal{E}_g)$ is visited, then the algorithm ends. With p, a tree with visited VTT configurations as vertices, it is straightforward to find the path connecting the initial VTT configuration and goal VTT configuration as well as the optimal reconfiguration actions sequence. With this algorithm, there is no need computing all possible VTT configurations which is usually very time-consuming and the tree is built as we explore VTT configurations.

V. EXPERIMENT

The algorithm is implemented and tested with some tasks. Two reconfiguration examples are presented. The initial VTT configuration $G = (V, E)$ is shown in Fig. 8. The location of each node is in the following

$$v_0\,[\texttt{Pos}] = [0.05, 0, 0]^\intercal \quad v_1\,[\texttt{Pos}] = [0.1, 1.8, 0]^\intercal$$
$$v_2\,[\texttt{Pos}] = [2.1, 1.9, 0]^\intercal \quad v_3\,[\texttt{Pos}] = [2.1, 0, 0]^\intercal$$
$$v_4\,[\texttt{Pos}] = [0, 2.1, 3.1]^\intercal \quad v_5\,[\texttt{Pos}] = [1.95, 0.9, 3]^\intercal$$
$$v_6\,[\texttt{Pos}] = [0, 0, 2.9]^\intercal$$

In order to explore and search the graph efficiently, a suitable model of action cost and a good heuristic function are necessary. There are three different fundamental actions: `Move`, `Split` and `Merge`. The action cost model in our experiment is that the cost of `Move` action is the moving distance and the cost of `Split` or `Merge` is 1. The heuristic function is

$$h(\mathcal{G}) = \begin{cases} \sum_{v \in \hat{V}} \|v^g\,[\texttt{Pos}] - v\,[\texttt{Pos}]\| + 1, & |\hat{V}| > 1 \\ \|v^g\,[\texttt{Pos}] - v\,[\texttt{Pos}]\|, & |\hat{V}| = 1 \end{cases}$$

in which $v^g\,[\texttt{Pos}]$ and $v\,[\texttt{Pos}]$ are the goal location and current location of node v respectively, and $|\hat{V}|$ is the size of set $\hat{V}$. Recall that $\hat{V} \subset V$ contains all the current nodes intersected by edge modules in $\hat{E}$ and if there are more than one node in set $\hat{V}$, then there must be at least one `Merge` action afterwards, so the heuristic value is the sum of Euclidean distance for every node from its current location to its goal location plus one. Due to the collision avoidance constraints, this heuristic function $h(\mathcal{G})$ must be less than or equal to the cost of moving from $\mathcal{G}$ to $\mathcal{G}_g$, so the algorithm is guaranteed to find the optimal action sequence (or shortest path).

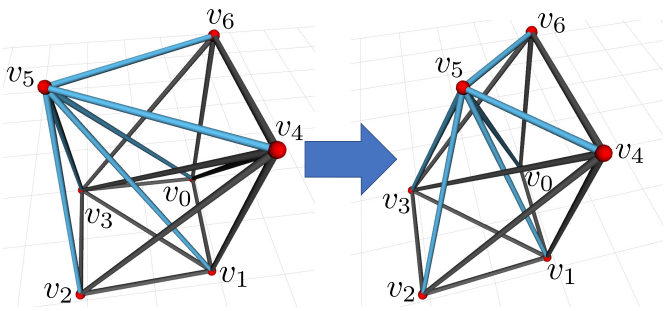

Fig. 9. $\forall e \in \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$, move $e\,[v_5]$ from initial location to goal location. With only geometry reconfiguration action, edge module (v_1, v_5) will collide with edge module (v_3, v_4).

1) Scenario 1: The first reconfiguration task is shown in Fig. 9. The task is to move edge module set $\hat{E} = \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$ and they all intersect at node v_5. $\forall e \in \hat{E}$, move $e\,[v_5]$ from its current position $[1.95, 0.9, 3]^\intercal$ to $[1, 0.9, 3]^\intercal$ which is very close to its original position. However, this motion cannot be executed by moving node v_5 to the goal position directly because edge module (v_1, v_5) will collide with edge module (v_3, v_4). Topology reconfiguration action is needed for this task.

The conversion between grid space and Cartesian space is computed first with $\delta = 0.2$ and the corresponding node locations in grid space are

$$\begin{aligned} \mathrm{v}_0\,[\mathtt{Pos}] &= [0,0,0]^\intercal & \mathrm{v}_1\,[\mathtt{Pos}] &= [0,4,0]^\intercal \\ \mathrm{v}_2\,[\mathtt{Pos}] &= [4,4,0]^\intercal & \mathrm{v}_3\,[\mathtt{Pos}] &= [4,0,0]^\intercal \\ \mathrm{v}_4\,[\mathtt{Pos}] &= [0,4,2]^\intercal & \mathrm{v}_5\,[\mathtt{Pos}] &= [4,2,2]^\intercal \\ \mathrm{v}_6\,[\mathtt{Pos}] &= [0,0,2]^\intercal \end{aligned}$$

and $\forall e \in E$, the goal location of $e\,[v_5]$ in grid space is $[2,2,2]^\intercal$. The lower bound of the grid space is $[0,0,0]^\intercal$ and the upper bound of the grid space is $[4,4,2]^\intercal$. In total, there are 75 grid locations. The size of $\hat{E}$ is six and the number of all possible arrangement for them in grid space is $75 \times 75 \times (C_6^3 C_3^3/2) = 56250$ and the size of action space is $56250 \times 52 = 2925000$. However, the algorithm only explores 3771 VTT configurations and the sequence of optimal actions can be found efficiently.

The solution by our motion planner is illustrated in Fig. 10. We first move node v_5 along $-x$-axis in grid space to a closer location ($[1.475, 0.9, 3.0]^\intercal$) in Cartesian space, then split the node into a pair of nodes so that two groups of edge modules can be controlled separately and move one of them to the location $[1.0, 1.35, 1.5]^\intercal$ which is below the obstacle edge module. This action moves the node with a longer distance than the previous one because the space occupancy is sparser along z-axis than that along x-axis. Next move two newly generated nodes along $-x$-axis to $[1.0, 0.9, 3.0]^\intercal$ and $[0.5, 1.35, 1.5]^\intercal$ respectively so that they can be merged in the goal location in the last step.

2) Scenario 2: The second reconfiguration task is shown in Fig. 11. The task is also to move edge module set $\hat{E} = \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$

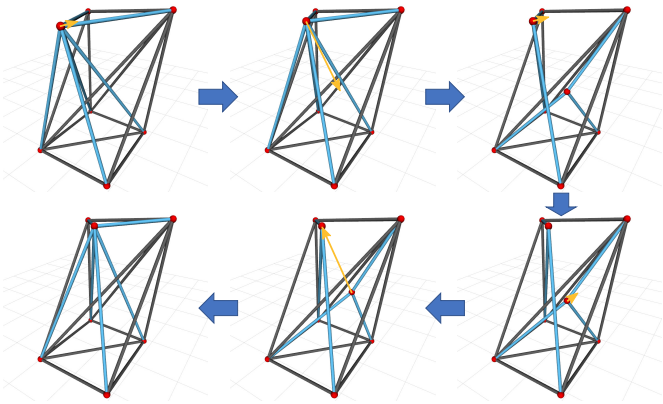

Fig. 10. The sequence to change the states of involved edge modules is shown and the motion directions are denoted as $\rightarrow$. First move the intersection node in a small step, and then split it into two separate nodes and move one of the node downward in a large step. Move both nodes closer and finally merge them in the goal location.

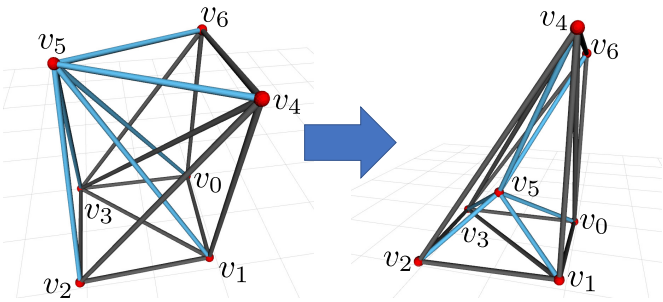

Fig. 11. $\forall e \in \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$, move $e\,[v_5]$ from initial location to goal location. With only geometry reconfiguration action, there is no way for edge module (v_0, v_5) and (v_1, v_5) to traverse edge module (v_3, v_4).

intersecting at node v_5 and $\forall e \in \hat{E}$, change $e\,[v_5]$ from its current position $[1.95, 0.9, 3]^\intercal$ to $[1, 1.2, 0.9]^\intercal$. The initial position is almost on the boundary of the truss but the goal position is almost in the center of the structure. Still with only geometry reconfiguration actions, it is impossible to finish this motion because edge module (v_3, v_4) is between two edge modules (v_0, v_5) and (v_1, v_5).

Set $\delta = 0.2$ to compute the conversion between grid space and Cartesian space and the corresponding node locations in grid space are

$$\begin{aligned} \mathrm{v}_0\,[\mathtt{Pos}] &= [0,0,0]^\intercal & \mathrm{v}_1\,[\mathtt{Pos}] &= [0,6,0]^\intercal \\ \mathrm{v}_2\,[\mathtt{Pos}] &= [4,6,0]^\intercal & \mathrm{v}_3\,[\mathtt{Pos}] &= [4,0,0]^\intercal \\ \mathrm{v}_4\,[\mathtt{Pos}] &= [0,6,4]^\intercal & \mathrm{v}_5\,[\mathtt{Pos}] &= [4,2,4]^\intercal \\ \mathrm{v}_6\,[\mathtt{Pos}] &= [0,0,4]^\intercal \end{aligned}$$

and $\forall e \in E$, the goal location of $e\,[v_5]$ in grid space is $[2,4,2]^\intercal$. The lower bound of the grid space is still $[0,0,0]^\intercal$ but the upper bound of the grid space is $[4,6,4]^\intercal$. This shows that how we decompose the space depending on the motion task. For this motion task, finer cells in y axis and z axis are generated with the same parameter δ. With higher resolution, there are more grid locations, in this example 175. The number of all possible arrangements for this set of edge modules is $175 \times 175 \times (C_6^3 C_3^3/2) = 206250$ and the size

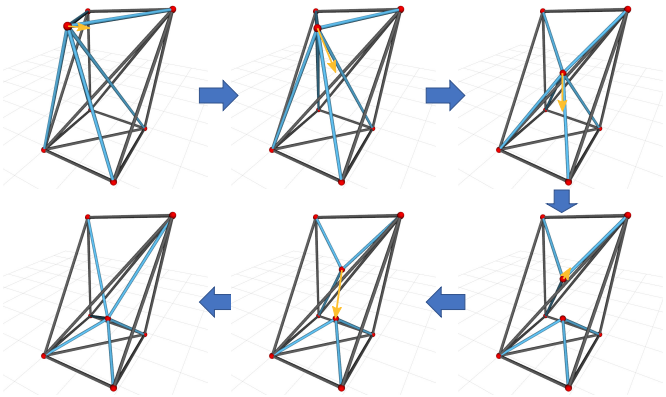

Fig. 12. The sequence to change the states of involved edge modules is shown and the motion directions are denoted as →. First move the intersection node to the center by two `Move` actions, and then split them into two separate nodes and move them in different directions to go around the obstacle member. In the end, merge these two nodes into a single one.

of action space is $306250 \times 52 = 15925000$ which is much larger. However, the algorithm only needs to explore 8146 VTT configurations.

The solution is illustrated in Fig. 12. Similarly the conversion between grid space and Cartesian space is computed accordingly. Firstly, move node v_5 to $[1.525, 1.05, 2.9]^\intercal$ and then to $[1.1, 1.2, 1.9]^\intercal$ with a larger step because this part of space is very sparse, and then split this node into a pair of nodes so that six edge modules are separated into two groups to go across the obstacle. Move one of them to the goal location and move the other one to $[0.55, 1.05, 1.9]^\intercal$ to bypass the obstacle, and then merge them at the goal location.

From these two examples, we can observe that there is no uniform motion stepsize because the space occupancy is different. This makes the reconfiguration planning algorithm efficient to explore VTT configurations using graph search algorithm.

VI. CONCLUSIONS

In this paper, we present a new reconfiguration planning algorithm for a new class of modular robot — the variable topology truss. This new robot is based on a truss structure that enables a wide variety of applications. The reconfiguration planning problem is challenging due to its large topology configuration space. Being a truss presents a variety of constraints to the inherent parallel robot planning problems. We present a new method to discretize the space called a grid space, considering the overall shape of the robot and the motion task which results in a more efficient decomposition of the configuration space. A robot is then modeled in both Cartesian space and this non-uniform grid space. Based on this new model, discrete reconfiguration actions can be applied and an efficient collision checking method is developed, and then a transition model is derived. We can explore the VTT configuration space efficiently using a graph search algorithm with a heuristic function to do motion planning combined with topology reconfiguration.

The future work will focus on the effect of different cost models and heuristic functions. Also more constraints on reconfiguration actions need to be studied since the system is over-constrained. We will also extend our work to more complicated reconfiguration tasks and demonstrate our algorithm on real hardware.

REFERENCES

[1] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular Self-Reconfigurable Robot Systems: Grand Challenges of Robotics," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, March 2007.

[2] K. Miura, "Design and Operation of a Deployable Truss Structure," in *NASA. Goddard Space Flight Center The 18th Aerospace Mech. Symp.*, Greenbelt, Maryland, May 1984, pp. 49–63.

[3] G. J. Hamlin and A. C. Sanderson, "TETROBOT Modular Robotics: Prototype and Experiments," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, vol. 2, Nov 1996, pp. 390–395 vol.2.

[4] A. Spinos, D. Carroll, T. Kientz, and M. Yim, "Variable Topology Truss: Design and Analysis," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 2717–2722.

[5] S. Jeong, B. Kim, S. Park, E. Park, A. Spinos, D. Carroll, T. Tsabedze, Y. Weng, T. Seo, M. Yim, F. C. Park, and J. Kim, "Variable Topology Truss: Hardware Overview, Reconfiguration Planning and Locomotion," in *2018 15th International Conference on Ubiquitous Robots (UR)*, June 2018, pp. 610–615.

[6] S. Park, E. Park, M. Yim, J. Kim, and T. Seo, "Optimization-Based Non-Impact Rolling Locomotion of a Variable Geometry Truss," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 747–752, April 2019.

[7] J. J. Champoux, "DNA Topoisomerases: Structure, Function, and Mechanism," *Annual Review of Biochemistry*, vol. 70, no. 1, pp. 369–413, 2001, pMID: 11395412.

[8] A. Spinos and M. Yim, "Towards a Variable Topology Truss for Shoring," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, June 2017, pp. 244–249.

[9] A. Casal and M. Yim, "Self-Reconfiguration Planning for a Class of Modular Robots," *Proc. SPIE*, vol. 3839, 1999.

[10] C. A. Nelson, "A Framework for Self-Reconfiguration Planning for Unit-Modular Robots," Ph.D. dissertation, Purdue University, West Lafayette, 2005.

[11] F. Hou and W.-M. Shen, "Distributed, Dynamic, and Autonomous Reconfiguration Planning for Chain-Type Self-Reconfigurable Robots," in *2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 3135–3140.

[12] M. Asadpour, M. H. Z. Ashtiani, A. Sproewitz, and A. Ijspeert, "Graph Signature for Self-Reconfiguration Planning of Modules with Symmetry," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2009, pp. 5295–5300.

[13] F. Hou and W.-M. Shen, "Graph-Based Optimal Reconfiguration Planning for Self-Reconfigurable Robots," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1047 – 1059, 2014, reconfigurable Modular Robotics.

[14] C. Liu, M. Whitzer, and M. Yim, "A Distributed Reconfiguration Planning Algorithm for Modular Robots," *IEEE Robotics and Automation Letters*, 2019.

[15] C. Liu and M. Yim, "Configuration Recognition with Distributed Information for Modular Robots," in *IFRR International Symposium on Robotics Research*, 2017.

[16] N. Usevitch, Z. Hammond, S. Follmer, and M. Schwager, "Linear Actuator Robots: Differential Kinematics, Controllability, and Algorithms for Locomotion and Shape Morphing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 5361–5367.

[17] C. Liu, S.-C. Yu, and M. Yim, "Shape Morphing for Variable Topology Truss," in *2019 16th International Conference on Ubiquitous Robots (UR)*, June 2019.

[18] M. Herman, "Fast, Three-Dimensional, Collision-Free Motion Planning," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, 1986, pp. 1056–1063.

[19] T. Möller and B. Trumbore, "Fast Minimum Storage Ray/Triangle Intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21 – 28, 1997.