Please find the final, published version in IEEE Xplore, DOI: 10.1109/ICRA40945.2020.9196880.

A Fast Configuration Space Algorithm for Variable Topology Truss Modular Robots

Chao Liu, Sencheng Yu, and Mark Yim

***Abstract*— The Variable Topology Truss (VTT) is a new class of self-reconfigurable robot that can reconfigure its truss shape and topology depending on the task or environment requirements. Motion planning and avoiding self-collision are difficult as these systems usually have dozens of degrees-of-freedom with complex intersecting parallel actuation. There are two different types of shape changing actions for a VTT: geometry reconfiguration and topology reconfiguration. This paper focuses on the geometry reconfiguration actions. A new cell decomposition approach is presented based on a fast and complete method to compute the collision-free space of a node in a truss. A simple shape-morphing method is shown to quickly create motion paths for reconfiguration by moving one node at a time.**

I. INTRODUCTION

Self-reconfigurable modular robots are usually composed of many repeated robot elements or modules. Modules can connect with each other to construct or reconfigure themselves into large structures depending on the tasks, environments or robot states [1]. More and more modular robotic systems have been developed in recent years, such as M-Blocks [2], PolyBot [3], Roombots [4] and SMORES-EP [5]. These modular robots are often roughly cubic or spheroidally shaped and modules occupy cells in a lattice nominally or form one or multiple chains for locomotion or manipulation. A newer type of modular robotic systems is composed of prismatic joints as truss members commonly called the variable geometry truss (VGT) [6] and includes TETROBOT [7] and Odin [8]. The variable topology truss (VTT) is similar to the variable geometry truss with additional capability to self-reconfigure its connection between members altering the truss topology. Hence, the variable topology truss system has both the efficiency benefits of VGTs and the flexibility of self-reconfigurable robots [9].

As a modular robotic system, a variable topology truss is capable of adapting itself into different configurations with respect to different requirements. For a VTT system, its *configuration* can be fully defined by the set of truss member link lengths and their node assignments at which point truss members are joined. Different configurations are best suited for different objectives, for example, a variable topology truss in cubic configuration can move around by dynamic rolling, while a configuration in a walker configuration with four legs can explore an environment by walking. There are also some configurations with large reachable workspace which are good at manipulation while some are better applied for shoring buildings or structures in disaster scenarios.

There are two different types of reconfiguration motion: *geometry reconfiguration* and *topology reconfiguration*. Geometry reconfiguration involves changing the length of member modules in a variable topology truss resulting in the motion of corresponding nodes. Topology reconfiguration involves changing the connectivity among members (changing their node assignments). In general, any topology reconfiguration also requires geometry reconfiguration in order to enable the connectivity changes.

A variable topology truss is composed of multiple edge modules which include a linear actuator as the truss member plus the two ends of the member that attach or detach from other ends to form the node [10]. A node is constructed by multiple ends of members by a linkage system with a passive rotational degree-of-freedom (DOF). The node assignments of every member define the topology and the length of every member defines the shape [11].

To be a reconfigurable variable topology truss, there are some constraints on the arrangement of members, including that a VTT has to be a rigid structure to maintain its shape and be statically determinant. In general, a node in a VTT needs at least three members attached to ensure its controllability. In addition, topology-reconfigurable VTT systems require at least 18 members to reconfigure [10] which means they have at least 18 actuated DOF. Thus, motion planning for these systems involves at least 18 and typically more than 21 dimensions. In addition, members forming VTT structure typically span the workspace in a very non-uniform manner as the truss configuration creates a complicated configuration space. Different from common open chain kinematic structures, it is much easier to solve inverse kinematics problem than to solve forward kinematics for parallel robots. Thus, rather than planning the motion of the VTT shape from the member lengths, we solve the shape morphing problem by finding feasible paths for all involved nodes and determine the required member lengths afterward. In addition, multiple nodes in a VTT are usually strongly coupled in the space, namely moving one node can significantly affect the configuration space and obstacle region of other nodes. We relax this constraint by moving one node (3 DOFs) at a time resulting in longer execution times compared to having all DOFs move at the same time. But the advantage is that this very high-dimensional planning problem is converted into multiple 3D problems which is feasible to be solved by graph search algorithm rather than

This work was sponsored by AFOSR grant FA2386-17-1-4656

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, schyu@seas.upenn.edu, yim@seas.upenn.edu

using high DOF search algorithms such as Rapidly-exploring Random Trees (RRT). In this way, the multi-node planning problem can be significantly simplified. However, a new challenge is to do cell decomposition in such a way that graph search algorithms can be applied efficiently.

In this paper, a fast and complete approach to compute the obstacle region and the free space of a node in a variable topology truss is presented. The free space can be divided into multiple convex polyhedrons in which the corresponding node can move freely without considering collision. Then a simple graph search algorithm can be used to plan a path for this node efficiently.

The rest of the paper is organized as follows. Sec. II reviews relevant and previous works. Sec. III introduces the fast and complete geometry reconfiguration algorithm for a single node. Sec. IV presents the shape morphing approach with multiple nodes involved. Some applications and analysis are shown in Sec. V to demonstrate the effectiveness of the algorithm. Finally, Sec. VI talks about the conclusion and future work.

II. RELATED WORK

Reconfiguration planning for self-reconfigurable robots have been developed for a variety of modular robotic systems, such as PolyBot [12], Crystal robot [13] and SMORES [5]. These algorithms work for traditional lattice-type, chain-type or mobile-type reconfiguration systems, but are not applicable to VTT systems which present very different physical constraints to the problem.

The variable topology truss systems were presented in [10], including the concept, hardware design and some basic analysis. To handle the high dimensional problem, sampling-based planning techniques were used in [9] to do geometry reconfiguration planning and a retraction-based RRT was developed for narrow passage problem, a well-known issue in sampling-based planning approaches. However, sometimes waypoints still need to be manually assigned to fully solve the shape morphing problem.

Kinematic control for TETROBOT was introduced in [7] but was limited to tetrahedrons or octahedrons. Linear actuator robots (LARs) are made up of linear actuators connected at universal joints into a network and a corresponding shape morphing algorithm was presented in [14]. Since these robots are in mesh graph topology constructed by multiple convex hulls, self-collision can be avoided easily. But this is not applicable to variable topology truss systems because they can be in any configurations rather than just mesh graph topology and the self-collision is the primary cause for complication in VTTs.

A new reconfiguration planning algorithm for variable topology truss systems was presented in [11]. This work combined topology and geometry reconfiguration, avoiding self-collision by implementing topology reconfiguration at appropriate times. The topological reconfiguration process adds complication and time to the execution process which would be better if avoided whenever possible. Hence, if possible a geometry only motion planning for variable topology truss systems would be desirable and was introduced briefly in [15], and this paper presents the complete algorithm of this idea.

III. SINGLE-NODE PLANNING

A. Variable Topology Truss Configuration

A variable topology truss can be fully represented 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 a property `Pos` to define the Cartesian coordinates of the vertex namely $v\,[\texttt{Pos}] = [v_x, v_y, v_z]^\intercal \in \mathbb{R}^3$ which is also the configuration of the node. Let $q^v = v\,[\texttt{Pos}]$ and it is apparent that the *configuration space* of node v denoted as $\mathcal{C}^v$ is $\mathbb{R}^3$. The state of $e = (v_1, v_2) \in E$ can be defined by q^{v_1} and q^{v_2}.

The shape of a variable topology truss is modified by controlling or moving nodes. For a node $v \in V$ in a VTT $G = (V, E)$, its motion is achieved by changing the length of all attached members denoted as $E^v \subseteq E$. E^v can be regarded as a parallel robot with all members being the joint actuators and, when moving node v, self-collision could happen between some pair of members in E^v or between $e \in E^v$ and $\bar{e} \in E \setminus E^v$.

B. Obstacle Region and Free Space

When moving a single node $v \in V$ in $G = (V, E)$, the state of every $e \in E^v$, denoted as $\mathcal{A}^v(q^v)$, can only be changed by altering q^v. For this node v, the *obstacle region* $\mathcal{C}^v_{obs} \subseteq \mathcal{C}^v = \mathbb{R}^3$ is defined as:

$$\mathcal{C}^v_{obs} = \{q^v \in \mathbb{R}^3 | \mathcal{A}^v(q^v) \cap \mathcal{O}^v \neq \emptyset\} \tag{1}$$

in which $\mathcal{O}^v$ is the obstacle for E^v. Then the *free space* of node v is just the leftover configurations denoted as

$$\mathcal{C}^v_{free} = \mathbb{R}^3 \setminus \mathcal{C}^v_{obs} \tag{2}$$

Theorem 1: For a given node v in $G = (V, E)$, $\mathcal{C}^v_{obs}$ can be fully defined by the states of $\forall e \in E \setminus E^v$.

Proof: Suppose the node to move is $v_i \in V$ then there are three cases where a moving member $e = (v_i, v_j) \in E^{v_i}$ could collide with other members:

1) Member $(v_i, v_j) \in E^{v_i}$ may collide with $(v_m, v_n) \in E \setminus E^{v_i}$ as shown in Fig. 1a, and the obstacle region generated by (v_m, v_n) can be defined by an unbounded polygon formed by member (v_m, v_n) and node v_j (the blue polygon in Fig. 1a). Since one node is connected with at least three nodes in a VTT, $\exists v_l \in V$, $(v_j, v_l) \in E \setminus E^v$. Therefore, the obstacle region in this case is defined by states of members in $E \setminus E^{v_i}$;
2) Member $(v_i, v_j) \in E^{v_i}$ collides with member $(v_j, v_m) \in E \setminus E^{v_i}$ if and only if the trajectory of v_i intersects with r, a ray starting at q^{v_j} and pointing in the direction of $q^{v_m} - q^{v_j}$ as shown in Fig. 1b. Then this ray-shaped obstacle region is defined by state of the member $(v_j, v_m) \in E \setminus E^{v_i}$;

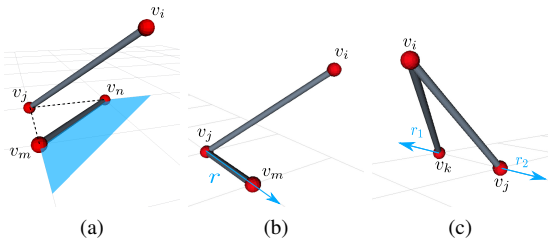


Fig. 1. (a) (v_i, v_j) collides with (v_m, v_n) when v_i is on the blue polygon. (b) (v_i, v_j) collides with (v_m, v_j) when v_i is on the blue ray r. (c) (v_i, v_j) collides with (v_i, v_k) when v_i is on the blue ray r_1 or r_2.

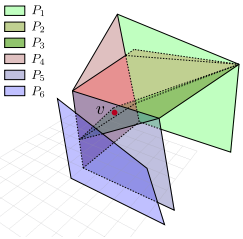


Fig. 3. v is enclosed by some polygons and any polygon outside the enclosure has nothing to do with $\mathcal{C}^v_{free}(q^v)$.

3) Member $(v_i, v_j) \in E^{v_i}$ collides with $(v_i, v_k) \in E^{v_i}$ if and only if the trajectory of v_i intersects with two rays r_1 or r_2 where r_1 starts from q^{v_k} and points in the direction of $q^{v_k} - q^{v_j}$ and r_2 starts from q^{v_j} and extends in the opposite direction, as shown in Fig. 1c. Since $\exists v_m \in V$, $(v_m, v_k) \in E \setminus E^{v_i}$, r_1 is contained in the obstacle region caused by (v_m, v_k) and (v_i, v_j). Hence r_1 is fully defined by the states of members in $E \setminus E^{v_i}$. And similar approach can be applied to r_2. Therefore, these obstacle regions are fully defined by the states of members in $E \setminus E^{v_i}$.

■

With Theorem 1, $\mathcal{C}^v_{obs}$ is constructed by all polygons generated from $E \setminus E^v$. For a simple VTT shown in Fig. 2a, the obstacle region $\mathcal{C}^{v_0}_{obs}$ for node v_0 is shown in Fig. 2b. The number of these polygons is of $O(|E|^2)$.

C. Free Space Boundary

All the polygons defined by $E \setminus E^v$ form the obstacle region $\mathcal{C}^v_{obs}$ for node v. If v is on any polygon, collision must happen among members. However, node v may not be able to move to any location inside $\mathcal{C}^v_{free}$ from its initial configuration q^v_i because $\mathcal{C}^v_{free}$ may be not fully connected and can be separated by $\mathcal{C}^v_{obs}$. For example, $\mathcal{C}^v_{obs}$ is composed of six polygons shown in Fig. 3, but node v is only able to move freely inside the space enclosed by polygon P_1, P_2, P_3, P_4 and P_5 starting from its current location. P_6 as well as parts of polygons P_1 — P_5 are outside the enclosed space. This enclosed space is the subset of $\mathcal{C}^v_{free}$ denoted as $\mathcal{C}^v_{free}(q^v)$ where q^v is the initial location of node v. It is necessary to find the boundary of $\mathcal{C}^v_{free}$ in order to do motion planning for v.

1) Polygon Intersection: Given a polygon P_i with α sides denoted as s_{ij} where $j \in [1, \alpha]$, if P_i is connected with a set of polygons denoted as $\mathcal{N}_{ij}$ through s_{ij}, then $\mathcal{N}_{ij}$ is called the neighbor set of P_i at side s_{ij}.

When checking intersections, there are four possible cases in terms of the intersection set of two polygons. For the cases that the intersection set is empty and the intersection set only contains a single point, no further computation is needed.

Another case that the intersection of two polygons is a line segment or a ray is shown in Fig. 4a. In this case, both polygons would be cut into two pieces by the intersection line. Let the two intersected polygons be P_1 and P_2, and the resulted separated polygons are $\{P'_1, P''_1\}$ and $\{P'_2, P''_2\}$, respectively. Assume that P_1 has α_1 sides denoted as $S_1 = \{s_{1j} \mid j \in [1, \alpha_1]\}$. Similarly, we have S'_1 for P'_1. Then, for any side $s'_{1j} \in S'_1$ such that $s'_{1j} = s_{1m} \in S_1$, namely this edge is inherited from P_1, the neighbor set $\mathcal{N}'_{1j}$ of P'_1 is equal to the neighbor set $\mathcal{N}_{1m}$ of P_1. This is called *inheritance process*. And for any side $s'_{1j} \in S'_1$ such that $\exists s_{1m} \in S_1$, s'_{1j} is a part of s_{1m}, namely s_{1m} in P_1 is cut and P'_1 only inherits part of s_{1m}, we check every polygon in $\mathcal{N}_{1m}$ and all those polygons that are connected with P'_1 compose $\mathcal{N}'_{1j}$. This is called the *recheck process*. Finally, for the side $s'_{1j} \in S'_1$ which coincides with the cutting line, let $\mathcal{N}'_{1j} = \{P''_1, P'_2, P''_2\}$. This is called the *adding process*. The same approaches will also be applied to P''_1, P'_2 and P''_2 to compute their neighbor sets. In addition, there are special situations for the third case. If the intersection lies on a side

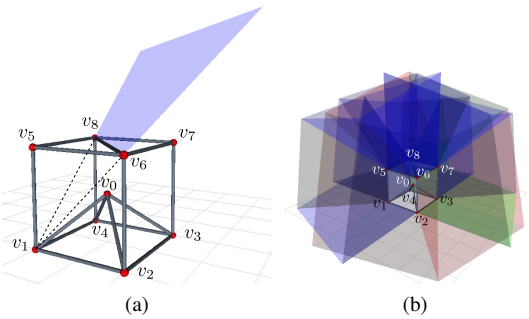


Fig. 2. (a) Given node v_0, one of its neighbors v_1 and a member (v_6, v_8) can define the blue polygon. This polygon is part of $\mathcal{C}^{v_0}_{obs}$. (b) The obstacle region $\mathcal{C}^{v_0}_{obs}$ of v_0.

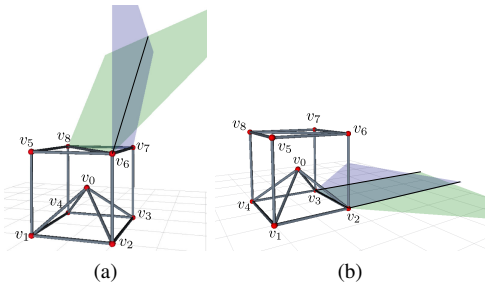


Fig. 4. (a) The intersection between the polygon generated by node v_1 with member (v_6, v_8) and the polygon by node v_2 with member (v_6, v_7) is a ray. (b) The intersection between the polygon generated by node v_1 with member (v_2, v_3) and the polygon by node v_4 with member (v_2, v_3) is also a polygon which is the region between the two parallel black lines.

of one polygon, say P_1, and inside another, say P_2, then only P_2 is cut and P_1 only requires the adding process. If the intersection lies on a side of both polygons, only adding process is applied to both.

The last case occurs when two polygons lie on the same plane and their intersection is a polygon (e.g. Fig. 4b). For this case, one of the polygons, e.g. P_1, remains unchanged and the other one, e.g. P_2, can be divided into a set of convex polygons, $\mathcal{R}_2$, so that no polygon in $\mathcal{R}_2$ overlaps P_1. To do this, starting from any $s_{1m} \in S_1$, we cut P_2 into P_2' and P_2'' using the line on which s_{1m} lies and apply above three processes to compute the neighbors of P_2' and P_2''. Only one of the resulted two polygons overlaps P_1, e.g. P_2''. We put P_2' into $\mathcal{R}_2$ and, if it is a neighbor of P_1, add it to neighbor sets of P_1 and add P_1 to neighbor sets of P_2'. Then continue to apply this operation to cut P_2'' using another side $s_{1n} \in S_1 \setminus \{s_{1m}\}$. Repeat this process until there is a newly generated polygon that is fully contained in P_1 and remove it from neighbors sets of polygons in $\mathcal{R}_2$.

The motion of a VTT node is kept inside the boundary of the workspace. The above operations are also applied to the boundary of the workspace. And after this process, we would obtain a set of polygons $\mathcal{P}_{obs}$, forming the whole $\mathcal{C}^v_{obs}$ and the boundary of workspace. Since each pair of polygons is checked, this process has quadratic time complexity in the number of polygons and therefore, $O(|E|^4)$.

2) Boundary Search: For a polygon P_i with α_i sides and all of its neighbor sets $\{\mathcal{N}_{ij} \mid j \in [1, \alpha_i]\}$, it is fully connected if and only if $\forall j \in [1, \alpha_i]$, $\mathcal{N}_{ij} \neq \emptyset$. The resulting polygons after intersection process are separated into two sets: the set of all fully connected polygons $\mathcal{F}$ and the set of all not fully connected polygons $\mathcal{U}$. Suppose the workspace is closed, it can be seen that the boundary of $\mathcal{C}^v_{free}(q^v)$, a polyhedron, can only be constructed from fully connected polygons. For any polygon P_i, there are two normal vectors in different directions perpendicular to the plane it lies on. The one pointing inside $\mathcal{C}^v_{free}(q^v)$ is called the inner direction vector denoted as $\mathbf{n}_i$. The distance between node v and a polygon is the minimum distance between v and any points on this polygon. The polygon P_s with the shortest distance to v (if multiple, choose the one with the maximum distance between v and the plane that the polygon lies on) must be a boundary polygon and we can find $\mathbf{n}_s$ regardless of whether $\mathcal{C}^v_{free}(q^v)$ is convex or not.

Then, Algorithm 1 is used to search for a set of boundary polygons $\mathcal{P}_b$ of $\mathcal{C}^v_{free}(q^v)$, shown in Fig. 5a. Since every polygon is checked at most once, this process takes $O\left(|E|^2\right)$ time. After this process, the set $\mathcal{U}$ has to be refined by removing polygons that are outside the obtained boundary.

Algorithm 1: Boundary Search Algorithm

Input: q^v current position of node v
$\mathcal{P}_{obs}$ the set of all polygons
Output: The set of boundary polygons $\mathcal{P}_b$ of $\mathcal{C}^v_{free}(q^v)$

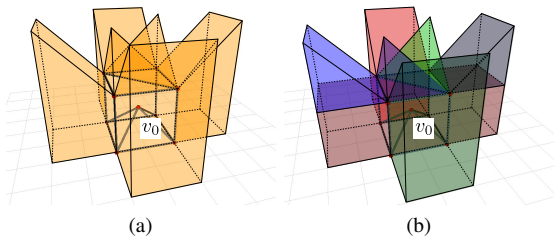
$P_s \leftarrow$ polygon closest to node v;
$\mathcal{P}_b \leftarrow \emptyset$;
$\mathcal{Q}_P \leftarrow \emptyset$;
$\mathcal{Q}_P$.enqueue(P_s);
while $\mathcal{Q}_P \neq \emptyset$ **do**
  $P_i \leftarrow \mathcal{Q}_P$.dequeue();
  $\mathcal{P}_b \leftarrow \mathcal{P}_b \cup \{P_i\}$;
  for $s_{ij} \in S_i$ **do**
    $\overline{P}_{ij} \leftarrow$ the innermost polygon in $\mathcal{N}_{ij}$ along $\mathbf{n}_i$;
    Compute inner direction vector $\bar{\mathbf{n}}_{ij}$ of $\overline{P}_{ij}$;
    if $\overline{P}_{ij} \notin \mathcal{P}_b$ *and* $\overline{P}_{ij} \notin \mathcal{Q}_P$ **then**
      $\mathcal{Q}_P$.enqueue($\overline{P}_{ij}$);
return $\mathcal{P}_b$

D. Cell Decomposition

The enclosed free space $\mathcal{C}^v_{free}(q^v)$ is not necessarily convex which can be further decomposed into several convex polyhedrons. The problem to decompose a non-convex polyhedra into a minimum number of convex pieces is known to be NP-hard [16]. We pass all polygons in $\mathcal{P}_b$ into a function of the Computational Geometry Algorithms Library (CGAL) to decompose the space into $O(r^2)$ where r is the number of edges that have two adjacent facets that span an angle of more than $180°$ with respect to the interior of the ployhedron [17]. The result is shown in Fig. 5b. Each convex polyhedron denoted as c is a cell in which the node can move freely without considering collision. However, some cells may intersect with some polygon P in $\mathcal{U}$ (set of not fully connected polygons) and these cells need to be further separated into two cells by the plane on which P lies (this is because CGAL ignores this case when doing convex decomposition). This checking process takes also $O(|E|^4)$ time, so the total time complexity of boundary construction for $\mathcal{C}^v_{free}(q^v)$ is $O(|E|^4)$.

E. Path Planning

With all decomposed cells, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be constructed where $\mathcal{V}$ is the set of all decomposed cells and $\mathcal{E}$ is the set of edges with each edge representing the connection of two adjacent cells. The cost of an edge is the distance of the trajectory the node has to traverse from one cell to another cell. Two cells are adjacent if and only if they are



(a) (b)

Fig. 5. (a) $\mathcal{C}^{v_0}_{free}(q^{v_0})$ is bounded by polygons. (b) $\mathcal{C}^{v_0}_{free}(q^{v_0})$ is decomposed into multiple convex polyhedrons.

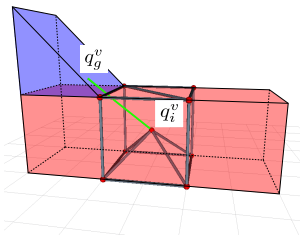


Fig. 6. The path planned for v_0 to move from its initial location q_i^v to q_g^v is shown as the green line as well as the involved cells shown as a blue polyhedron and a red polyhedron traversed by the trajectory .

not separated by polygons in $\mathcal{U}$. When traversing from one cell c' to its adjacent cell c'' in $\mathcal{G}$, the trajectory is a straight line connecting the centers of two adjacent cells if it is inside $\mathcal{C}^v_{free}(q^v)$. Otherwise, the trajectory is from the center of cell c' to the center of the intersection polygon between c' and c'', then to the center of c''. For node v, given its initial location q_i^v and goal location q_g^v, find the cell c_i and c_g containing q_i^v and q_g^v respectively. Then a path from c_i to c_g in $\mathcal{G}$ can be found using Dijkstra Algorithm. For example the planned path for v_0 in a simple VTT is shown in Fig. 6.

F. Completeness for Single Node Planning

We claim that this single-node planning algorithm is complete, i.e. it must compute a (continuous) *path*, $\tau : [0,1] \rightarrow \mathcal{C}^v_{free}$, such that $\tau(0) = q_i^v$ and $\tau(1) = q_g^v$, or correctly report that such a path does not exist [18].

According to Theorem 1, the obstacle region $\mathcal{C}^v_{obs}$ of a node v is fully defined, as well as free space $\mathcal{C}^v_{free} = \mathbb{R}^3 \setminus \mathcal{C}^v_{obs}$. Recall that for a configuration of a node q^v, $\mathcal{C}^v_{free}(q^v)$ is the enclosed space containing q^v and its boundary is formed by either $\mathcal{C}^v_{obs}$ or the workspace boundary, thus v cannot go outside $\mathcal{C}^v_{free}(q^v)$ from q^v, or it will traverse the obstacle region or the workspace boundary. Hence, if the goal location $q_g^v \notin \mathcal{C}^v_{free}(q^v)$, there is no feasible path. Otherwise, there must be a feasible path τ connecting q_g^v with q_i^v.

$\mathcal{C}^v_{free}(q^v)$ is a polyhedron that can be decomposed into T convex polyhedrons denoted as $\{c_t | t = 1, 2, \cdots, T\}$ by convex decomposition operation, and q_i^v and q_g^v must be in some cells. Let $q_i^v \in c_i$ and $q_g^v \in c_g$. Node v can move freely from q_1^v to q_2^v as long as $q_1^v \in c_t$ and $q_2^v \in c_t$ where $t = 1, 2, \cdots, T$. Then, for adjacent cells c_m and c_n, there must exist a feasible path $\tau_{mn} : [0,1] \rightarrow \mathcal{C}^v_{free}(q^v) \subseteq \mathcal{C}^v_{free}$ such that $\tau_{mn}(0) = q_m^v$ and $\tau_{mn}(1) = q_n^v$ where $q_m^v \in c_m$ and $q_n^v \in c_n$ since node v can always move from q_m^v to the common boundary of c_m and c_n freely and then move to q_n^v freely. Recall that all of the convex polyhedron cells construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and Dijkstra Algorithm is complete to find the shortest path, then it is guaranteed to find a sequence of convex cells from c_i to c_g. Hence, the path $\tau : [0,1] \rightarrow \mathcal{C}^v_{free}$ such that $\tau(0) = q_i^v$ and $\tau(1) = q_g^v$ is derived.

IV. SHAPE MORPHING APPROACH

With the ability to find a path of a single node in VTT, shape morphing can be achieved by a sequence of single-node motion. Assume there are n nodes v_1, v_2, $\cdots$, v_n that should be moved from $q_i^{v_1}$, $q_i^{v_2}$, $\cdots$, $q_i^{v_n}$ to $q_g^{v_1}$, $q_g^{v_2}$, $\cdots$, $q_g^{v_n}$ respectively to achieve a shape morphing task. We first compute $\mathcal{C}^{v_1}_{free}(q_i^{v_1})$ and move v_1 to $q_g^{v_1}$, then compute $\mathcal{C}^{v_2}_{free}(q_i^{v_2})$. If $q_g^{v_2} \in \mathcal{C}^{v_2}_{free}(q_i^{v_2})$, move v_2 to $q_g^{v_2}$. Otherwise, try move the next candidate. The process is repeated until all nodes are moved to their destination or return failure. Thus the problem reduces to be finding a sequence of node motions. In the worst case, we have to try $n!$ times which is the permutation of the number of moving nodes. The efficiency of single-node planning makes this approach applicable for the multi-node planning problem. The benefit of doing this is that we don't need to do the free space computation and cell decomposition frequently. However, the drawback is that it may not be able to find the solution for some extreme cases even there exists a feasible path.

V. EXPERIMENTS

The algorithm is implemented in C++ with the Boost Graph Library (BGL) [19] and the Computational Geometry Algorithms Library (CGAL) [17]. We use CGAL to do convex decomposition of a non-convex polyhedron and use BGL to do graph search. To demonstrate the algorithm, two experiments were conducted, both on a single core of an Intel i7 processor at 2.20 GHz. The first experiment is a single-node planning and the second one is a multi-node planning.

A. Single-Node Experiment

Several tests were used to evaluate our approach and do comparison with RRT, which is previously used for VTT geometry motion planning. The VTT used in this test is shown in Fig. 7. For each node v, a goal location $q_g^v \in \mathcal{C}^v_{free}(q_i^v)$ satisfying $\|q_g^v - q_i^v\| < 1$ is selected randomly. The results are shown in the table I.

From this test result, our approach is faster than RRT. In addition, one significant feature of VTT is able to achieve large workspace. But RRT can take more than 40 s to search a path with 4 times larger workspace, while our method is

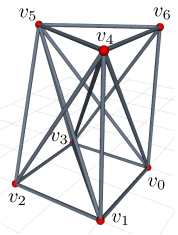


Fig. 7. A VTT Constructed by 18 Members

TABLE I
COMPARISON OF OUR METHOD WITH RRT

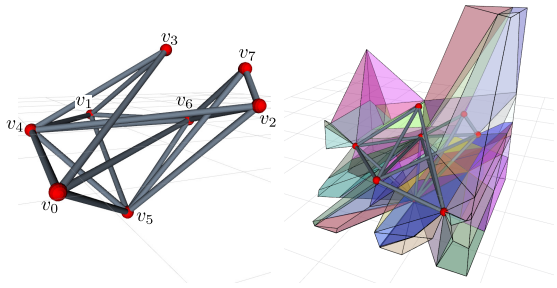
Test Node	Our Method / Cell Decomposition (s)	RRT (s)
v_0	0.3031 / 0.2301	0.3050
v_1	0.3581 / 0.2740	1.0245
v_2	0.5145 / 0.4383	0.9092
v_3	0.1292 / 0.0552	2.2419
v_4	0.1013 / 0.0294	0.7646
v_5	0.0578 / 0.0327	0.8052
v_6	0.2796 / 0.2022	0.6283

not affected by the workspace size. Also most time in our approach is taken by CGAL for cell decomposition which can be imporved by using a faster cell decomposition method but may end up with more convex cells generated.

A VTT shown in Fig 8a, constructed from 17 members, is used for a difficult task test. The task is to move node v_7 from its initial configuration $q_i^{v_7}$ to a goal configuration $q_g^{v_7}$ shown in Fig. 9d. This is extremely hard for RRT because the space is narrow. The result of cell decomposition is shown in Fig. 8b with 56 cells in total. For this task, node v_7 needs to traverse three cells as shown in Fig. 9. The planning process takes $1.054\,\mathrm{s}$, among which CGAL takes $0.959\,\mathrm{s}$.

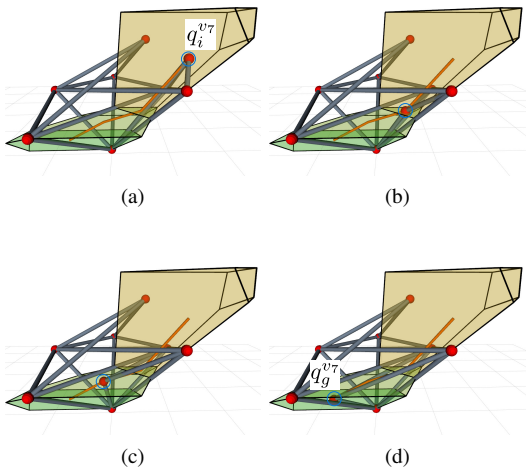
B. Multi-Node Experiment

The experiment for multi-node planning is to change the shape of a cubic VTT shown in Fig 10a to a tower VTT shown in Fig. 10b. The VTT is composed of 21 members and there are four nodes v_1, v_3, v_5 and v_6 involved in this shape morphing process. The approach is able to find a valid sequence of moving each node and the result is



(a) (b)

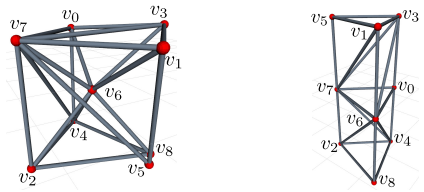
Fig. 8. (a) A VTT is constructed from 17 members with 8 nodes. (b) $\mathcal{C}_{free}^{v_7}(q_i^{v_7})$ is decomposed into 56 cells in total.



(a) (b)

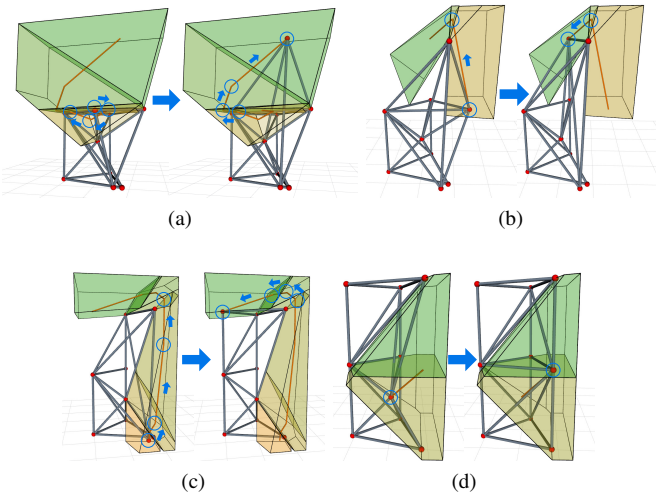
(c) (d)

Fig. 9. The task is to move node v_7 from its initial configuration $q_i^{v_7}$ shown in (a) to a goal configuration $q_g^{v_7}$ shown in (d). The three cells and the complete path node v_7 has to traverse are shown. v_7 is moved to the intersection between the first cell and the second cell shown in (b), then to the center of the second cell shown in (c), and finally to the goal location inside the third cell.



(a) (b)

Fig. 10. The motion task is to change the shape of a VTT from (a) a cubic truss for rolling locomotion to (b) a tower truss for shoring.



(a) (b)

(c) (d)

Fig. 11. The nodes are all encircled by "○" and their complete paths are shown as "—" path. (a) For node v_1, there are 61 cells generated after cell decomposition process and it has to traverse five cells to go to the destination. (b) For node v_3, there are 87 cells generated after cell decomposition process and it has to traverse three cells to go to the destination. (c) For node v_5, there are 40 cells generated after cell decomposition process and it has to traverse seven cells to go to the destination. (d) For node v_6, there are 34 cells generated after cell decomposition process and it has to traverse two cells to go to the destination.

to move v_1, v_3, v_5 and v_6 in order. The motions for all involved nodes are shown in Fig. 11a, Fig. 11b, Fig. 11c and Fig. 11d respectively. It takes $4.004\,\mathrm{s}$ to find the path to do shape morphing and $3.509\,\mathrm{s}$ is consumed by CGAL. This experiment is also tested in [9] and, with the retraction-based RRT algorithm, a waypoint that node v_5 has to be moved higher than node v_1 needs to be added manually to mitigate the narrow passage problem.

VI. CONCLUSION

A fast and complete approach is presented to compute the enclosed free space of a given node in a variable topology truss which makes it possible to do cell decomposition more efficiently and accurately. Then a simple graph-based searching algorithm can be used to generate an optimal path for a single-node motion task. Based on this approach, a shape morphing algorithm for VTT systems is introduced by changing the locations of multiple nodes. Some useful applications are used to demonstrate the effectiveness of the algorithm. Future work will focus on the high-level topology reconfiguration in a large scale by applying this technique.

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] J. W. Romanishin, K. Gilpin, S. Claici, and D. Rus, "3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1925–1932.

[3] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: a modular reconfigurable robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, April 2000, pp. 514–520 vol.1.

[4] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A. Ijspeert, "Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1016 – 1033, 2014, reconfigurable Modular Robotics.

[5] C. Liu, M. Whitzer, and M. Yim, "A distributed reconfiguration planning algorithm for modular robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4231–4238, Oct 2019.

[6] 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.

[7] G. J. Hamlin and A. C. Sanderson, "Tetrobot: a modular approach to parallel robotics," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 42–50, March 1997.

[8] A. Lyder, R. F. M. Garcia, and K. Stoy, "Mechanical design of odin, an extendable heterogeneous deformable modular robot," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2008, pp. 883–888.

[9] 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.

[10] 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.

[11] C. Liu and M. Yim, "Reconfiguration motion planning for variable topology truss," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019, pp. 1941–1948.

[12] A. Casal and M. Yim, "Self-reconfiguration planning for a class of modular robots," in *Proc. SPIE*, vol. 3839, 1999, pp. 3839–3839–12.

[13] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for lattice-based self-reconfigurable robots," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 919–937, 2004.

[14] 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.

[15] C. Liu, S. Yu, and M. Yim, "Shape morphing for variable topology truss," in *2019 16th International Conference on Ubiquitous Robots (UR)*, June 2019.

[16] B. Chazelle, "Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm," *SIAM Journal of Computing*, vol. 13, no. 3, pp. 488–507, 1984.

[17] (2019) The computational geometry algorithms library. [Online]. Available: https://www.cgal.org/

[18] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[19] (2019) The boost graph library (bgl). [Online]. Available: https://www.boost.org/doc/libs/1_71_0/libs/graph/doc/index.html