A Linking Invariant for Truss Robot Motion Planning

Alexander Spinos and Mark Yim, *Member, IEEE*

***Abstract*—In this paper, we introduce a new invariant of C-space components for truss robots: the link-augmented graph. This invariant uses techniques from knot, link, and spatial graph theory to encode the linking information between different closed chains in the robot. For robots with a disconnected free configuration space, this invariant serves as a tool to distinguish robot configurations that lie in different connected components of C-space from each other. This can be used to eliminate goal positions that are unreachable by any collision-free motion, without needing to perform any probabilistic planning. This invariant can also be used to find appropriate assignments of node labels in a specified goal position. We demonstrate the advantages of using this invariant in conjunction with a probabilistic planner, and introduce a variant of RRT-Connect to simultaneously search for all valid goal labelings.**

***Index Terms*—Motion and path planning, computational geometry, cellular and modular robots.**

I. Introduction

In THIS paper, we introduce a method for characterizing the configurations of certain types of robots. This technique is particularly useful for truss robots, in which the kinematic structure is that of a rigid truss. However, it can also be applied any robotic system that exhibits some closed kinematic structure that may become tangled, such as chain-style modular robots [1]. Variable Geometry Trusses (VGTs) are formed by taking a truss structure and replacing some or all of the members with linear actuators [2]. This type of robot has classically been used for parallel manipulators [3], long chain actuators, collapsible structures for space applications [4], [5], and locomotion platforms. More recently, other actuation schemes have been explored to create soft truss robots [6]. Tensegrity robots are a popular class of truss robot that combine flexible cables with rigid struts to create flexible and robust robotic structures [7].

Trusses naturally have a modular structure. Many others have explored making the truss elements out of composable modules that can be combined to form various shapes [8]–[10]. Our prior work has introduced the Variable Topology Truss (VTT), a modular self-reconfigurable truss robot [11], [12]. VTT can change its shape by merging and splitting the truss nodes in addition to changing the length of the truss members. A hardware prototype is shown in Fig. 1.

Manuscript received September 9, 2021; accepted December 19, 2021. Date of publication January 4, 2022; date of current version January 12, 2022. This letter was recommended for publication by Associate Editor J. Yu and Editor S. J. Guy upon evaluation of the reviewers' comments.*(Corresponding author: Alexander Spinos.)*

The authors are with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: spinos@seas.upenn.edu; yim@grasp.upenn.edu).

Digital Object Identifier 10.1109/LRA.2021.3139941

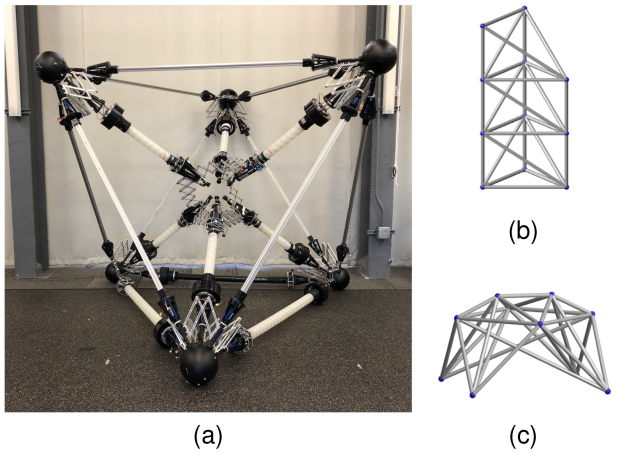

Fig. 1. One example of the type of robot this work applies to: the Variable Topology Truss system. (a) A hardware prototype demonstrating the reconfiguration capability. (b), (c) Two configurations that a 27-member VTT system could transform between.

Truss robots are highly structurally efficient and have many of the benefits associated with parallel robots, including increased stiffness and precision. However, motion planning for truss robots is very difficult due to the self-collision obstacles between the members. The dimensionality of the configuration space is very high, and often the members are tangled in such a way that certain motions are impossible. In these cases, the configuration space may consist of many disconnected regions.

In our prior work, we have demonstrated the success of probabilistic planners with VTT when the problem can be reduced to moving a small number of nodes at a time [13]. This prior work leverages the ability of VTT to avoid self-collision obstacles by reconfiguring its topology. However, the reconfiguration process typically bears a high cost, being both time consuming and error-prone. Most modular robots, including VTT, need to precisely align their connectors to successfully dock two modules. Not only that, but the vast majority of truss robots cannot reconfigure their topology at all. Therefore, we are interested in finding out precisely when collision-free paths to a desired goal exist without requiring reconfiguration.

One could attempt to run a probabilistic planner to answer this question, but probabilistic planners can rarely determine that a path does *not* exist. The high dimensionality and many narrow passages in the configuration space of most truss robots mean

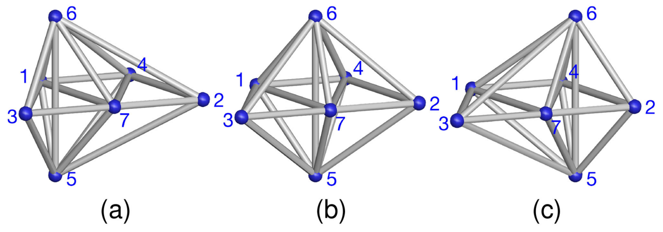

Fig. 2. Three truss robot configurations with the same topology.

that checking the existence of a path with a probabilistic planner to a high degree of confidence can be very expensive. For this reason, we want to avoid the use of probabilistic planners unless we have high confidence that a path to the goal does exist.

In this paper, we introduce a C-space component invariant—the link-augmented graph—that can be calculated from a given truss configuration. We explore two useful applications of the invariant in this work. First, the link-augmented graph can be used to quickly identify situations where a path from a start configuration to a goal configuration is impossible. Second, in cases where the robot consists of re-labelable modules, this graph can be used to generate label choices that could result in a possible path.

II. Introducing an Invariant for Robots With Linked Kinematic Structure

We start by giving a concrete example. Fig. 2 shows three configurations of a truss robot. We can see that there is no straightforward collision-free path from configuration 2(a) to configuration 2(b). The truss member (5, 6) would collide with truss member (1, 7), and attempting to move around this obstacle only results in further collision. In fact, it turns out that the configuration space of this robot is disconnected. In a non-reconfigurable VGT, there is no path between configurations 2(a) and 2(b). In a self-reconfigurable truss robot such as VTT, this obstacle could be avoided using topological reconfiguration.

Similarly, there is also no collision-free path from configuration 2(a) to configuration 2(c) without reconfiguration. However, this situation is slightly different. In robotic systems with interchangeable robot modules, the specific labeling of modules is not important. We can see that if we were to swap the label of node 1 with node 4, node 2 with node 3, and node 5 with node 6—that is, apply the permutation $(1\,4)(2\,3)(5\,6)$—then configuration 2(c) can be reached simply by rotating configuration 2(a). By allowing this relabeling, a previously impossible path for a VGT was made possible, and a path with many reconfiguration steps for a VTT was simplified to involve no reconfiguration.

From a multi-robot systems perspective, this can be considered a variation of the goal assignment problem [14]. However, the tight coupling of the self-collision obstacles to the node positions in a tangled truss robot mean that general purpose multi-robot planning techniques are ill-suited. From a modular robotics perspective, this can be considered a variation of the configuration matching and mapping problem, where the goal configuration may be selected from some library of configurations and the current configuration needs to be relabeled to match the goal [15]. However, the prior work in modular robotics only considers the graph structure of the modular system, i.e. the logical description of which modules are connected to which other modules. For truss robots such as VGTs or VTT, we refer to this graph structure as the *topology* of the truss. However, in the given example, all of the configurations already have a matching topology. The topology alone fails to capture important spatial characteristics of the robot configuration. In this work, we provide a method for encoding this spatial information along with the robot topology, which forms a C-space component invariant of the robot configuration.

A. *Invariants of Configuration Space Components*

We define the C-space of the robot to exclude self-collision obstacles, so the C-space may consist of several connected components. We then define a C-space component invariant to be any property that can be computed from a given robot configuration that remains constant throughout any self-collision-free motion of the robot. This concept is analogous to the concept of a knot invariant in knot theory [16]. If two robot configurations yield different results when computing the value of the invariant, then there is certainly no collision-free path between them. They must lie in different connected components of the robot C-space. If two robot configurations yield the same result, then there *may* be a path between them. Either they lie in the same connected component of C-space, or two different connected components happen to yield the same value of the invariant. An invariant is considered relatively *stronger* when it is less likely that two different connected components yield the same value, and *complete* if every connected component yields a unique value.

Configurations of truss robots such as VGTs or VTT can be fully represented by a graph $G(V, E)$ and an embedding function $P : V \mapsto \mathbb{R}^3$. The vertices (V) of the graph correspond to the set of truss nodes and the edges (E) correspond to the set of truss members. The embedding function P specifies the location of each truss node in 3D space. Other modular robot systems can be represented in a similar way.

B. *Invariants of Spatial Graphs*

A more general notion of a graph embedded in 3D space is known as a *spatial graph*. Prior work on spatial graphs defines equivalence with respect to ambient isotopy [17]. The edges of a spatial graph may follow any tame arc (any arc equivalent to a piecewise linear path) through space between the endpoints. In this case, the embedding function P must also specify the path of each edge. Truss robots belong to the stricter class of linear spatial graphs, where the edges must be straight lines. Thus, any invariant of a spatial graph is also an invariant of a linear spatial graph.

A simple but powerful invariant of spatial graphs was introduced by Kauffman [17]. For a spatial graph (G, P), the invariant $T(G, P)$ is the collection of knots and links embedded in the graph. A knot is simply a single tame closed curve in space, whereas a link can have multiple closed curves. To extract a knot or link from a spatial graph (G, P), we can delete edges in the graph until each vertex has at only two or zero incident edges, and the result will be a closed cycle or set of closed cycles. $T(G, P)$ is computed by applying all possible choices of deleting edges in this way, identifying the result, and adding it to the collection.

This method is particularly useful because it converts a problem about spatial graphs to a problem about knots and links, which are much more well understood. In contrast, methods that operate on the spatial graph directly tend to be difficult to compute or are limited to graphs with specific vertex types. For example, the Yamada polynomial is a spatial graph invariant that is limited to flat vertex graphs or pliable vertex graphs with maximum degree less than four [18]. The invariant introduced in this paper is suitable for the most general type of spatial graph—pliable vertex graphs with unbounded maximum degree.

The invariant introduced in this paper will begin with the same decomposition of the graph to a set of knots or links. However, $T(G, P)$ discards important information about the graph structure. For example, $T(G, P)$ makes no distinction between a simple loop formed by three edges and a simple loop formed by four edges. Although these shapes are equivalent in an ambient isotopic sense, in reality there is no way for a chain of three robot modules to smoothly change to a chain of four robot modules. Our new invariant captures this distinction, which is particularly useful for robot systems that exhibit greater diversity in their graph structure than they do in their ambient isotopic structure, such as chain-style modular robots [19].

C. The Link-Augmented Graph

In this section, we introduce a new invariant for truss robots, the link-augmented graph, denoted $L(G, P)$, and its canonical isomorph $C(L(G, P))$. An example of constructing this invariant is given in Fig. 3. The input to the procedure is a linear spatial graph that represents the robot configuration. The input used in the example is a simplified version of the truss robot shown in Fig. 2. The number of truss nodes is the same, but several members have been removed to make the calculation manageable by hand.

We first compute all edge-induced subgraphs such that all vertices have degree two or zero. In the given example, there are three such subgraphs, shown in Fig. 3(b). The result is a set of graphs in which each graph is composed of one or more cycles. That is, each graph corresponds to a link diagram when taken with its embedding. At this stage, one could compute $T(G, P)$. In the example, the first two subgraphs correspond to the Hopf link, while the third corresponds to the unlink. The value of $T(G, P)$ is then simply $\{\texttt{HopfLink}, \texttt{UnLink}\}$.

When computing $L(G, P)$, we make the following simplification for computational efficiency. Rather than fully identifying the resulting knots and links, we simply compute pairwise linking between cycles in each subgraph. This is done by calculating the linking number for each pair of cycles using Gauss's linking integral. The orientation of the cycles is not well-defined, so we do not consider the sign of the linking number. We also do

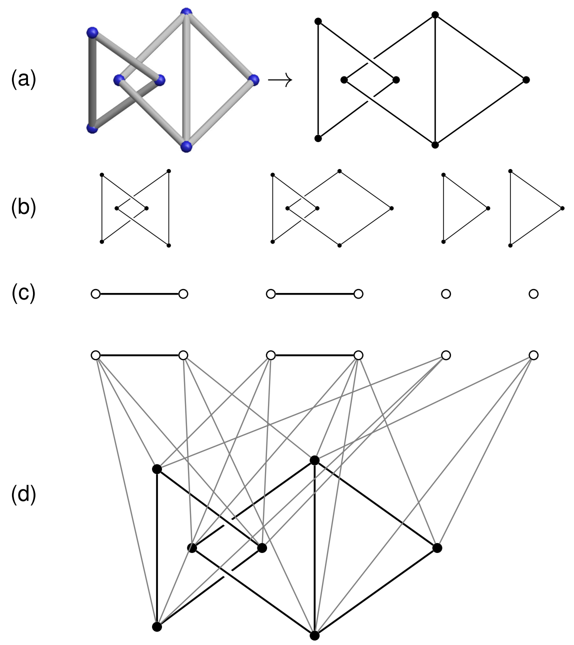

Fig. 3. Generating the link-augmented graph. (a) The truss configuration is represented as a linear spatial graph. (b) Generate all possible edge-induced subgraphs that result in a set of cycles. (c) Compute a graph representing the linking between each pair of cycles in each subgraph. (d) Connect these linking graphs to the original graph.

not consider multiply linked loops, saving only the binary state of linked or unlinked, as the multiply linked cases are fairly rare. One advantage of this simplification is that the linking integral can be computed directly on the three-dimensional vertex locations of the spatial graph. More sophisticated topological tools, such as knot polynomials, typically require finding a regular projection of the structure and computing the crossing information in the projection [16].

We can represent the result of the pairwise linking calculation by another set of graphs, which we term linking graphs. Each vertex in these linking graphs corresponds to a cycle in the original graph G, and the vertices are connected by an edge if the cycles are linked. The set of linking graphs for the given example is shown in Fig. 3(c). The concept of using the linking graph as a topological invariant for a set of loops has been used in computer graphics to verify calculations on linked structures such as chainmail [20].

Finally, the link-augmented graph $L(G, P)$ is constructed. Each vertex in the linking graphs is added to the original graph G by connecting it to all of the vertices in the cycle that it represents. These newly added vertices are given a unique color to distinguish them from the original vertices in G. This coloring must be preserved during isomorphism checking or canonicalization. Note that $L(G, P)$ is not a spatial graph. The embedding function P is used for calculating $L(G, P)$, but $L(G, P)$ itself is not embedded in any space.

If relabeling the nodes in the graph is not allowed, then $L(G, P)$ suffices as an invariant. This corresponds to the case where each robot module is unique, and module labels must be preserved. However, if we are allowed to relabel the robot modules, then the isomorphism class of $L(G, P)$ is the relevant invariant. Returning to the example from Fig. 2, we can denote the three configurations as (G, P_A), (G, P_B), and (G, P_C). Then, $L(G, P_A)$, $L(G, P_B)$, $L(G, P_C)$ will be different graphs, but $L(G, P_A)$ will be isomorphic to $L(G, P_C)$ and not isomorphic to $L(G, P_B)$. To conveniently compare the isomorphism class of $L(G, P)$, we can put the graph into its canonically labeled form: $C(L(G, P))$. If two graphs are isomorphic, then their canonically labeled forms will be identical.

This invariant can be slightly modified to support cases where the robot system consists of a heterogeneous set of modules. If only modules of the same module type may interchange labels, the vertices of the original graph G can be colored according to module type. Then, the relevant isomorphism class is restricted to isomorphisms that preserve this coloring.

D. Limitations

The link-augmented graph is not a complete invariant of the robot configuration. That is, there may be pairs of start and goal configurations that are not connected by a collision-free path, and yet yield the same link-augmented graph. One weakness is that we do not attempt to classify graph cycles as knots. Therefore, a truss that embeds a trefoil in one of its cycles may be indistinguishable from one that only embeds an unknotted loop. Similarly, we only check for pairwise linking between cycles. This fails to distinguish more complicated types of linking—such as the Whitehead link or Brunnian links—from unlinked loops [16]. This invariant could be strengthened by detecting these cases and assigning special colors to the nodes and edges of the linking graphs, at the cost of considerable added complexity.

III. Relabeling Goal Configurations Using the Link-Augmented Graph

Given a start configuration (G_S, P_S) and a goal configuration (G_G, P_G), we can use this invariant to determine when a collision-free path may be possible. If a path might possible under some node relabeling, we also compute this relabeling. The goal graph G_G must be isomorphic to G_S, but it need not be identical. In this work, we relabel the goal configuration to be achievable by the start configuration. That is, the goal configuration will be transformed to the form $(\pi(G_G), \pi(P_G))$, where π is some permutation that takes G_G to G_S. In some applications, it may be desirable to relabel the current configuration to match the goal configuration instead.

We begin by computing the link-augmented graph of the start and goal configurations, and placing each into canonical form. If the resulting canonical link-augmented graphs are not equivalent, then no collision-free path exists between the configurations. If the canonical link-augmented graphs are equivalent, then a collision-free path may exist. However, some relabeling may be necessary first.

The canonical forms of the link-augmented graphs induce permutations on the original graphs G_S and G_G. That is, by computing $C(L(G_S, P_S))$, we obtain the permutation σ that transforms $L(G_S, P_S)$ to $C(L(G_S, P_S))$. We can restrict σ to a permutation $\hat{\sigma}$ on only the original vertices in G_S. This restricted permutation is closed over the vertices in G_S because the canonicalization preserves the graph coloring. Similarly, we obtain the permutation γ that transforms $L(G_G, P_G)$ to $C(L(G_G, P_G))$ and its restriction $\hat{\gamma}$. Then, permuting the goal configuration by $\hat{\sigma}^{-1}\hat{\gamma}$ yields a configuration with the same link-augmented graph as the start configuration, yet in the same shape as the goal position.

However, this permutation is not necessarily the only one that yields a goal position with the correct value of the invariant. In fact, for the example given in Fig. 2, there are two permutations that yield the correct link-augmented graph: $(1\,4)(2\,3)$ and $(1\,4)(2\,3)(5\,6)$. Of these, only the latter results in a goal configuration that is reachable by the start configuration. This is because the link-augmented graph of this configuration has some symmetry—more precisely, it has a nontrivial automorphism group. When restricted to the original graph, this induces a nontrivial automorphism group which corresponds to the permutation $(5\,6)$.

We can account for this by enumerating all automorphisms of $L(G_G, P_G)$. Then, restrict those automorphisms to automorphisms on the original nodes in G_G. Finally, permute each of those automorphs of G_G by $\hat{\sigma}^{-1}\hat{\gamma}$ to get a set of possible goals. If there is a node permutation that results in a reachable goal configuration, then it must be one of these possible goals.

A. Computational Complexity and Practical Running Time

The first step of calculating the link-augmented graph is finding all of the relevant edge-induced subgraphs that produce sets of cycles. If a node in the graph has degree d, there are $d(d-1)/2$ choices to delete edges at that node. These choices must be made for each node, so the number of graphs produced can be exponential in the number of nodes. For most graphs that correspond to practical physical structures, edges tend to be deleted fast enough that this step terminates in a reasonable time. In the examples shown in this paper, the running time for this step is always less than 1 s. Additionally, this calculation does not depend on the embedding of the truss. When evaluating multiple configurations with the same topology, this set of graphs only needs to be computed once and can be reused thereafter.

In this work, the linking number is computed using Gauss's integral. This calculation is straightforward and efficient for paths defined by a small number of straight-line edge segments [21]. For larger and more complicated structures, more sophisticated techniques exist for accelerating this computation [20].

We use `nauty` to put the link-augmented graph in canonical form. This calculates a set of generators for the automorphism group as a side effect of canonicalization [22]. Graph canonicalization is theoretically very complex in the worst case, but `nauty` is quite efficient for graphs with a relatively small number of nodes. To reduce the number of nodes in the link-augmented graph, we slightly modify the algorithm as described in Section II. If a linking graph contains no edges, we do not add it to the link-augmented graph. This does not discard any information that could not be recovered from G alone, so the strength of the invariant is the same. However, trusses that are not highly linked

TABLE I
CALCULATING $C(L(G, P))$ FOR 100,000 RANDOM CONFIGURATIONS

	Truss 1	Truss 2	Truss 3	Truss 4
Num Unique Values	4	567	2782	25945
Num to Cover 50%	1	8	7	152
Chance of Match	0.44	0.042	0.061	0.026
Time to Calc Subgraphs	0.013 s	0.19 s	0.12 s	0.20 s

produce link-augmented graphs with far fewer nodes, which can be processed by `nauty` more efficiently.

IV. Planning With Multiple Possible Goals

If a path can be found to any of the possible goal positions, then the goal configuration can be reached and planning is a considered a success. Any standard probabilistic planner can be used to sequentially attempt to find a path to each possible goal. However, if a single-query planner is used, this wastes time re-exploring the space near the start position for each attempt.

To improve the efficiency of this process, we introduce RRT-Connect-Any, a variant of RRT-Connect [23]. Standard RRT-Connect builds two search trees, one rooted at the start and one at the goal. The fundamental step is to grow one tree, then attempt to connect the other tree to the new vertex using a greedy heuristic. The roles of the trees are then swapped and the process repeats, alternating between the two trees until a connection is made.

In RRT-Connect-Any, we build one tree rooted at the start and additional trees rooted at each of the possible goals. The start tree always participates in each step, while the goal trees are cycled through one at a time. This way, 50% of the samples are used to build the start tree, and the remaining 50% of samples are distributed among the different goal trees. This strategy prevents pathological behavior when the number of possible goals is very large; in such cases RRT-Connect-Any will behave similarly to unidirectional RRT. When there is only one possible goal, RRT-Connect-Any is identical to RRT-Connect.

V. Results

A. Invariant Strength

In this section, we present some results that demonstrate the usefulness of the canonical link-augmented graph as an invariant. First, we investigate the distinguishing power of the invariant. Fig. 4 shows four example trusses. For each of these trusses, we generate 100,000 random configurations and check how many different canonical link-augmented graphs we find. Truss configurations are generated by randomly drawing each node position from a uniform distribution within the unit cube. Configurations that are in collision or very close to collision are discarded and replaced.

The results are given in Table I. For each truss, we give the total number of different values of the invariant that were found. This number gives a lower bound on the number of disconnected regions in the robot configuration space. We can see that trusses

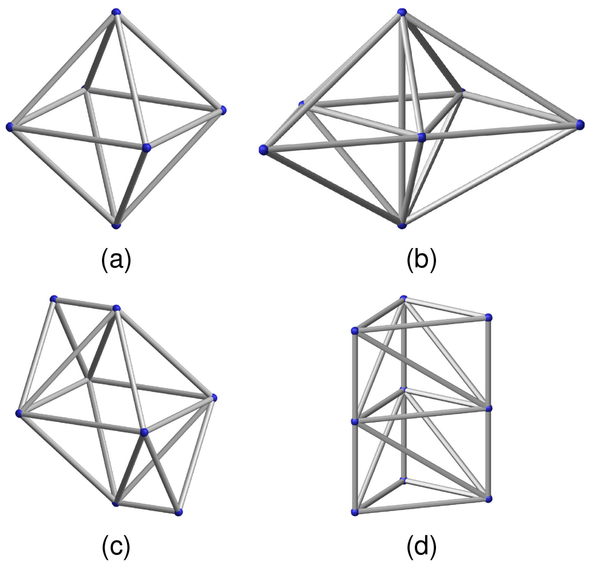

Fig. 4. Four example trusses. (a) An octahedron with 6 nodes and 12 members. (b) An overconstrained truss with 7 nodes and 18 members. (c) An octahedron with two attached tetrahedra, totaling 8 nodes and 18 members. (d) A tower of stacked tetrahedra, totaling 9 nodes and 21 members.

with more nodes have a highly disconnected configuration space, with thousands of different regions.

These values do not occur with equal probability. Some occur quite frequently, whereas others correspond to quite delicate and rare configurations. The table gives the number of the most common values that account for at least 50% of configurations. The table also gives the probability that two randomly sampled configurations will have a matching invariant value. It is interesting that Truss 3 has a higher chance that two random configurations share the same value compared to Truss 2, despite having a larger total number of possible values.

Finally, the table lists the time to compute the set of edge-induced subgraphs for each graph. This only needs to be calculated once for each of the example trusses.

In this paper, we are interested in a general planning query where the goal position may be anywhere in the configuration space of the robot. The results in Table I show that this space is highly disconnected, motivating the use of the link-augmented graph. Since it is unlikely that two random configurations will have a matching value, the link-augmented graph can provide a quick answer to the majority of planning queries. On the other hand, if the goal position is selected so that it is likely to lie in the same component of C-space, then the link-augmented graph may be less useful—for example, by sampling very close to the start position.

B. Planning

Next, we investigate using this invariant along with a simple probabilistic planner. For each of the example trusses, we generate 10,000 pairs of random configurations and attempt to plan a path between them. The first step is to compute the canonical link-augmented graph for both start and goal configurations. We

TABLE II
PERFORMANCE OF 10,000 RANDOM PLANNING ATTEMPTS

	Truss 1	Truss 2	Truss 3	Truss 4
Mean Time per Query	0.51 s	0.31 s	0.39 s	0.54 s
Breakdown:				
Time to Calculate $C(L)$	0.019 s	0.024 s	0.030 s	0.080 s
Time to Run RRT-C-A	1.1 s	6.3 s	5.9 s	16.4 s
Case 1: Found Path	0.13 s	3.7 s	3.0 s	8.4 s
Case 2: Timed Out	4.7 s	8.5 s	9.6 s	17.5 s
Number of Occurrences:				
$C(L)$ Values Matched	4480	455	602	279
RRT-C-A Found a Path	3536	209	334	36
Result is Certain	9056	9754	9732	9757

TABLE III
COMPARISON OF THREE PLANNING APPROACHES ON 5,000 RANDOM START-GOAL PAIRS WITH TRUSS 2

	Single	All	Match
Mean Time per Query	8.53 s	8.51 s	0.28 s
Breakdown:			
Time to Calculate $C(L)$	–	–	0.025 s
Time to Run RRT	8.53 s	8.51 s	6.39 s
Case 1: Path Found	2.32 s	2.72 s	4.13 s
Case 2: Timed Out	8.55 s	8.56 s	8.59 s
Number of Occurrences:			
$C(L)$ Values Matched	–	–	199
Planner Found a Path	13	43	98
Result is Certain	13	43	4899

assume that the appropriate set of edge-induced subgraphs has been computed beforehand, and doesn't need to be recomputed for each query. If the canonical link-augmented graphs don't match, planning is skipped because a path is impossible. If the canonical link-augmented graphs do match, then the relevant permutations and the list of automorphisms are calculated to produce a set of possible goals. We then run RRT-Connect-Any to try to reach any goal from the start point. If any goal is reached, we consider the planning a success. If the maximum number of samples (10,000) is reached without finding a goal, the planner times out.

The results are given in Table II. First, the table gives the overall average execution time for a query. This is broken down into the times for each part of the algorithm. It gives the average time to compute the canonical link-augmented graphs, denoted $C(L)$, then the average time to run RRT-Connect-Any if necessary. The running time of RRT-Connect-Any is further broken down into average times for success and time-out. The results show that computing the value of this invariant is much faster than running a probabilistic planner.

Next, Table II gives the number of times the invariant values matched, which corresponds to the number of attempts to run RRT-Connect-Any. Of those, it gives the number of planning attempts that successfully found a path. Finally, it gives the number of times that the result is certain; that is, either it found a path or it found that no path can exist.

The number of times the invariant matched during these planning attempts corresponds with our expectations from Table I. Attempts to run RRT-Connect-Any are likely to succeed for Trusses 1–3, which suggests that the invariant can distinguish reasonably well between different connected components of the configuration space in these cases. However, the success ratio begins to drop for Truss 4, which has more nodes and members than the other examples. This is likely because knots and link types that are not distinguished by this invariant become more prevalent as the number of members grows. For example, it is known that every embedding of the K_7 graph contains a trefoil knot [24].

Another contributing factor is that all of the planning attempts in this experiment use the same sampling limit. We might expect that more complicated trusses will require more samples to have an equivalent success ratio, even in cases where the start and goal are in the same connected component. However, the ratio of planning time for a success to that of a time-out is fairly similar for Trusses 2–4, so we do not expect this factor to have a large impact.

Although the probabilistic planning success ratio is reduced for Truss 4, the link-augmented graph is still powerful enough to skip 9,721 out of the 10,000 planning queries. If every failure of the probabilistic planner were due to spurious collision in the invariant value, then a perfectly complete invariant could skip at most 243 additional planning attempts. This high distinguishing power is because Truss 4 can exhibit a rich variety of ways that its loops can be pairwise linked. We would expect this particular invariant to be less useful in cases where the robot is more likely to be knotted than linked, such as in a chain-style modular robot system with long strings of modules and few branches [1].

C. Comparison With Other Methods

In this section, we demonstrate the advantage of the invariant-based technique over more traditional techniques. Using Truss 2, we generate 5,000 pairs of random configurations. On each pair, we try three methods to find whether a path between them exists. For the first method, *Single*, we simply run RRT-Connect between the start and the goal. This is simple, but it misses out on the opportunity to find goals that are only reachable by relabeling. For the second method, *All*, we run RRT-Connect-Any using all possible automorphisms of the goal configuration. This only uses the graph topology, and does not compute any linking information. For Truss 2, there are 12 automorphisms of the graph, so each planning attempt will simultaneously search for 12 goals. This should eventually find a path if one exists, but it will spend much more time investigating the space around unreachable goal points. This will increase the running time, and the less efficient use of samples means that the planner is more likely to terminate before it finds a solution, should one exist. Finally, we use the procedure from Section V-B, which computes the link-augmented graph to avoid searching for unreachable goals. This method is denoted *Match* in the table. This should have a success ratio at least as good as that of the second method, with a much faster running time.

The results are shown in Table III. Since the maximum number of samples is fixed, the average planning times for *Single* and *All* are about the same. However, *Match* can skip planning when the invariants do not match, so the average time is much lower.

Match also has an even higher success ratio than expected. This is likely because the much more efficient use of samples means that more complicated paths can be discovered before the planner hits the sampling limit. Of the 199 planning attempts for *Match*, 173 of them only needed to search for one or two goals.

Lastly, the traditional probabilistic methods can only answer with certainty when a path is found. In many cases, *Match* can provide a definitive answer that no path exists, in which case it is included as a certain result at the bottom of Table III.

VI. Conclusion

The link-augmented graph is a powerful C-space component invariant for truss robots, and it can be extended to other robot systems that feature linked kinematic chains. The planning results demonstrate that the invariant is useful for evaluating the reachability of goal positions. It also provides relabelings of the goal position that might result in a path, if any exist. This can be used in conjunction with a simple probabilistic planner to greatly improve planning performance.

Although the link-augmented graph is a much stronger tool than the graph structure alone, it still has some limitations. This invariant cannot distinguish between more complicated classes of knots and links that tend to appear as the number of modules increases. However, the framework used to construct the link-augmented graph by combining linking information with graph structure can easily be extended to incorporate more powerful knot-theoretic information. Future work will explore more sophisticated methods of detecting the knots and links embedded in the robot kinematic structure to create even stronger invariants.

References

[1] J. Seo, J. Paik, and M. Yim, "Modular reconfigurable robotics," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 2, pp. 63–88, 2019.

[2] V. A. Reinholtz and L. T. Watson, "Enumeration and analysis of variable geometry truss manipulators," Comput. Sci., Virginia Polytech. Inst. State Univ., Tech. Rep. TR-90-10, 1990.

[3] Y. Patel *et al.*, "Parallel manipulators applications-A survey," *Modern Mech. Eng.*, vol. 2, no. 03, pp. 57–64, 2012.

[4] K. Miura, H. Furuya, and K. Suzuki, "Variable geometry truss and its application to deployable truss and space crane arm," *Acta Astronautica*, vol. 12, no. 7/8, pp. 599–607, 1985.

[5] M. D. Rhodes and M. M. Mikulas Jr, "Deployable controllable geometry truss beam," NASA Langley Res. Center, Tech. Rep. TM-86366, 1985.

[6] N. S. Usevitch, Z. M. Hammond, M. Schwager, A. M. Okamura, E. W. Hawkes, and S. Follmer, "An untethered isoperimetric soft robot," *Sci. Robot.*, vol. 5, no. 40, 2020, Art. no. eaaz0492.

[7] D. S. Shah *et al.*, "Tensegrity robotics," in *Soft Robotics*. Berlin, Germany: Springer, 2021.

[8] G. J. Hamlin and A. C. Sanderson, *Tetrobot: A Modular Approach to Reconfigurable Parallel Robotics*, vol. 423. Berlin, Germany: Springer, 2013.

[9] C.-H. Yu, K. Haller, D. Ingber, and R. Nagpal, "Morpho: A self-deformable modular robot inspired by cellular structure," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2008, pp. 3571–3578.

[10] Z. M. Hammond, N. S. Usevitch, E. W. Hawkes, and S. Follmer, "Pneumatic reel actuator: Design, modeling, and implementation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 626–633.

[11] A. Spinos and M. Yim, "Towards a variable topology truss for shoring," in *Proc. 14th Int. Conf. Ubiquitous Robots Ambient Intell.*, 2017, pp. 244–249.

[12] A. Spinos, D. Carroll, T. Kientz, and M. Yim, "Topological reconfiguration planning for a variable topology truss," *J. Mechanisms Robot.*, vol. 13, no. 4, 2021, Art. no. 040901.

[13] C. Liu, S. Yu, and M. Yim, "Motion planning for variable topology truss modular robot," in *Proc. Robot, Sci. Syst.*, 2020.

[14] S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Trans. Robot.*, vol. 22, no. 4, pp. 650–665, Aug. 2006.

[15] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *Int. J. Robot. Res.*, vol. 27, no. 3/4, pp. 403–421, 2008.

[16] C. C. Adams, *The Knot Book*. Providence, RI, USA: Amer. Math. Soc., 1994.

[17] L. H. Kauffman, "Invariants of graphs in three-space," *Trans. Amer. Math. Soc.*, vol. 311, no. 2, pp. 697–710, 1989.

[18] S. Yamada, "An invariant of spatial graphs," *J. Graph Theory*, vol. 13, no. 5, pp. 537–551, 1989.

[19] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: A modular reconfigurable robot," *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 1, pp. 514–520, 2000.

[20] A. Qu and D. L. James, "Fast linking numbers for topology verification of loopy structures," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 106:1-106:19, Aug. 2021.

[21] Z. Arai, "A rigorous numerical algorithm for computing the linking number of links," *Nonlinear Theory Appl. IEICE*, vol. 4, no. 1, pp. 104–110, 2013.

[22] B. D. McKay and A. Piperno, "Practical graph isomorphism, II," *J. Symbolic Computation*, vol. 60, pp. 94–112, 2014.

[23] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," *Proc. Millennium Conf.. IEEE Int. Conf. Robot. Automat. Symposia Proc.*, vol. 2, pp. 995–1001, 2000.

[24] J. H. Conway and C. M. Gordon, "Knots and links in spatial graphs," *J. Graph Theory*, vol. 7, no. 4, pp. 445–453, 1983.