

Synthesizing Functional Mechanisms From a Link Soup*

Pouya Tavousi^{†1}, Kazem Kazerounian², and Horea Ilies³

^{1,2,3}Department of Mechanical Engineering, University of Connecticut

Abstract

The synthesis of functional molecular linkages is constrained by difficulties in fabricating nano-links of arbitrary shapes and sizes. Thus, the classical mechanism synthesis methods, which assume the ability to manufacture any designed links, cannot provide a systematic process for assembling such linkages.

We propose a new approach to build functional mechanisms with prescribed mobility by only using elements from a predefined “link soup”. First, we enumerate an exhaustive set of topologies, while employing divide-and-conquer algorithms to control the generation and elimination of redundant topologies. Then, we construct the linkage arrangements for each valid topology. Finally, we output a set of feasible geometries through a positional analysis step that minimizes the error associated with closure of the loops in the linkage while avoiding geometric interference. The proposed systematic approach outputs the ATLAS of candidate mechanisms, which can be further processed for downstream applications. The resulting synthesis procedure is the first of its kind that is capable of synthesizing functional linkages with prescribed mobility constructed from a soup of primitive entities.

1 INTRODUCTION

The classical approach to mechanism synthesis involves three steps: (a) *type synthesis* to select the type of mechanism realizing a specific motion requirement; (b) *number synthesis* to determine the topology of the mechanism fulfilling the prescribed mobility requirements; and (c) *dimensional*

*An earlier version of this manuscript appeared in the proceedings of IDETC.

[†]Address all correspondence to this author.

synthesis to specify the geometric dimensions of the synthesized mechanism [1]. One of the key assumptions of this classical synthesis procedure is that the resulting links output by step (c) can be manufactured to the designed dimensions.

However, fabricating links at the naonscale, for example as needed by molecular machines, is notoriously difficult [2] and the shapes and sizes of the links that can be potentially manufactured in a lab environment are limited. In fact, the most viable approach that is currently available for synthesizing nano-mechanisms is to use existing molecular components (e.g. rigid fragments of amino acids) as building blocks for constructing new molecular machinery. Therefore, a feasible molecular mechanism synthesis strategy can only use a finite number of types of links that either exist in nature or perhaps could be fabricated. This restriction forces the replacement of the *dimensional synthesis* step of the traditional method with the *positional analysis* stage, which investigates the closure of kinematic loops. It also reformulates the design task into seeking functional mechanisms that can be assembled by only picking elements from a *link soup* of primitive entities. Observe that this new design task should result in an ATLAS of candidate mechanisms, which are kinematically plausible.

In this paper, we propose a linkage synthesis method (Fig. 1) aimed at supporting mechanism synthesis at the nano-scale for a prescribed number of degrees of freedom (*DOF*). The proposed method:

1. *enumerates all valid topologies* that satisfy certain mobility conditions in terms of the desired *DOF* of the final mechanism;
2. generates all possible *linkage arrangements* by assigning links from the link soup to each topology, and
3. performs *positional analysis* for every *linkage arrangement*, to extract the geometrically feasible solutions.

The topology enumeration phase of the traditional mechanism synthesis aims at producing topologies represented by graphs, followed by detecting cases of isomorphism to yield a canonical set of non-redundant solutions satisfying prescribed topological criteria [1]. However, graph isomorphism detection is an NP-complete problem[3]. Comprehensive reviews of the conventional methods for topology enumeration and isomorphism detection in the context of mechanism synthesis are provided in [4, 5], and a recent review of the graph matching literature appears in [6]. In general, the task of discarding redundant topologies requires a quadratic number of pairwise

comparisons. A different approach is taken in [7] to avoid the generation of redundant topologies based on permutation groups. However, this technique cannot address the redundancies of linkage arrangements, i.e. when graph vertices carry additional information about the link properties as well as the joint connectivity of adjacent links. Furthermore, this method enumerates topologies containing only single edges, which can become problematic when multiple connections need to be formed among constituents, a common occurrence at the molecular level, where the building blocks and the interactions between them are more intricate than at the macro scale. One example of such a case is presented in [8] showing two molecular domains connected by multiple hydrogen bonds.

We propose a generic technique for generating non-isomorphic graphs with prescribed topological characteristics (Fig. 1). The proposed divide and conquer method controls the redundant topologies that are generated, which reduces the overall number of required isomorphism checks. By detecting and eliminating the graph redundancies as soon as they emerge, we reduce the exponential growth down to linear growth. Our method first generates the *link families*¹ that are associated with the prescribed *DOF* [1], and satisfy prescribed bounds on the number of links or connections/joints for each link selected from the link soup. Then, for each link family, we enumerate all feasible topologies as discussed in section 2. The hierarchical structure induced by the divide and conquer approach can not only control the propagation of redundant topologies, but may also be used to store the link families corresponding to a prescribed mobility as well as the topologies belonging to a link family. These solutions can then be retrieved and reused efficiently as needed during the current or subsequent design processes. It must be noted that the general topology enumeration technique proposed here is based on prescribed requirements and its functionality does not depend on the availability of a structural element set. Also, note that although the proposed computation of the link families, as an intermediate enumeration product, expedites the topology enumeration process, it can also produce topologies with undesired *DOF* due to emergence of over-constrained subgraphs [9]. Thus, the topology enumeration step is followed by a mobility analysis² as a post-processing step. In this paper we use the method presented in [11] to identify the over-constrained subgraphs, and replace them with rigid links, although we note that other approaches to detect anomalies in local mobility have been documented in the literature

¹We use the term *link family* according to the definition provided in [1].

²See also [10].

[12, 13]³.

In principle, any well-established isomorphism detection technique can be used to test for redundancies at each level of the divide-and-conquer hierarchy. However, we propose a new heuristic similarity detection technique that eliminates the pairwise-comparison between topologies by assigning a characteristic matrix to every topology as discussed in section 2, leading to a more computationally efficient process. This characteristic matrix acts as a “topological signature,” and allows us to evaluate the presence of isomorphism in a set of candidate topologies. We illustrate the efficiency of the proposed isomorphism detection for our problem through the benchmark tests discussed in section 3. A slightly modified version of the similarity detection method is also used to identify similarities during the generation of linkage arrangements (Fig. 1), when link components are assigned from the link soup to the vertices of the candidate topologies.

The final step of the proposed approach is the positional analysis, when the joint variables in the linkage arrangements are adjusted to satisfy two sets of requirements: (1) simultaneous closure of kinematic loops (Fig. 1), and (2) no geometric interference. This step can be formulated as an optimization problem, and any traditional optimization method proposed in the literature [14, 15, 16] can in principle be used. In section 2 we describe one specific implementation of an optimization algorithm, which takes advantage of the particular attributes of the method that we propose. The objective function is defined as an error function that takes into consideration the kinematic loop closure as well as the amount of geometric overlap between the links.

To the best of our knowledge, the resulting tool is the first synthesis tool aimed at finding linkages that can be constructed from a soup of primitive entities that may be available to the designer. Although the scope of this technique is rather general, its immediate application is the design of molecular machines which have to be assembled from nano-links that either exist or can be manufactured at that scale. As it is commonly the case, the molecular designers can use their knowledge of chemistry and biochemistry to compile a set of chemical building blocks that can be considered to be rigid links and to form the link soup. We illustrate the effectiveness of the

³Identifying overconstrained regions is a relatively minor task in the proposed proposed enumeration process, and, in practice, its efficiency has a relatively small impact on the overall computational efficiency. The method used here has been developed by this research group, which allowed us to rapidly develop the implementation. The authors, however, acknowledge the existence of more efficient techniques, which can be used in subsequent developments.

approach by reproducing the regular typologies reported in [17], and its efficiency by providing a comparison with a traditional isomorphism detection approach. We also provide validation test for the positional analysis implementation. Finally, we used the synthesis tool to design a one *DOF* 7-bar protein-based linkage from existing molecular components and to compute its range of motion. The divide-and-conquer method proposed here augmented by the heuristic graph isomorphism detection technique results in a significantly faster synthesis process than the conventional mechanism synthesis methods. Furthermore, the two step redundancy elimination (redundant topologies and redundant linkage arrangements) has a critical effect in improving the computational efficiency during the positional analysis phase. Finally, the specific formulation and implementation of the error minimization during positional analysis leads to an effective functional mechanism synthesis procedure for a prescribed mobility requirement.

2 METHODS

Figure 1 provides a bird’s eye view of the proposed synthesis technique that outputs the ATLAS of candidate mechanisms along with their ranges of motions. The methods presented in this section are designed and developed to address the specific nano-design problem at hand. Our focus was on rapid development of a robust and practical design tool, but fine-tuning its computational efficiency may require further investigations other implementations that exist for solving specific tasks.

2.1 Enumerating Topologies for Prescribed Mobility

Mobility, which by definition, captures the degrees of freedom or, alternatively, the number of independent variables needed to completely describe the configuration of a linkage [1], acts as one of the main inputs for our synthesis tool. Given the geometry of a mechanism, one can easily determine its associated *DOF*. However, the inverse problem is critical in the mechanism synthesis process. The Grübler-Kutzbach criterion suggests a formula for quick determination of *DOF* based only on the knowledge of the number of links and number and types of constraints in the mechanism. This must be noted however, that due to existence of over-constrained regions in some mechanisms, the suggested formulae may report mobilities that are lower than the true values. In other words, Grübler-Kutzbach yields only a lower bound for the *DOF* of a mechanism [18]. Here, we use this criterion as a “bridge” between mobility and topology. Specifically, assuming that the

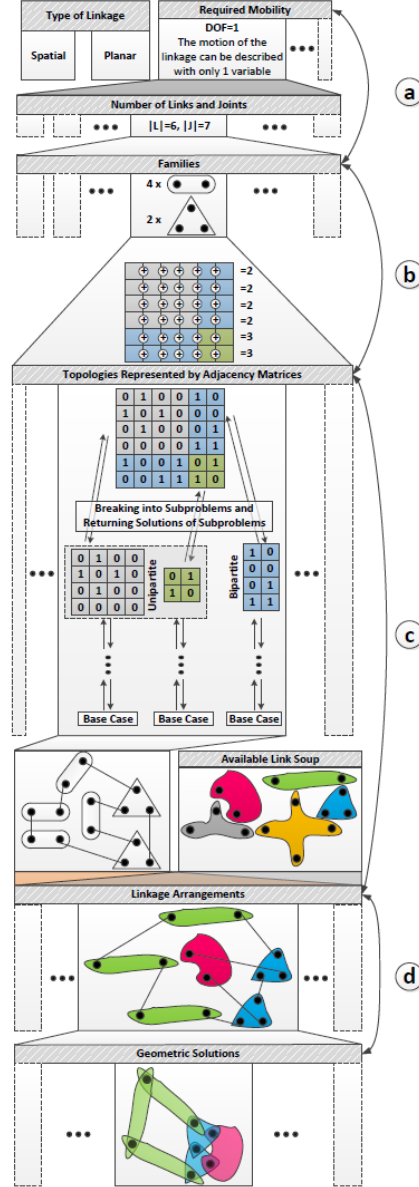


Figure 1: The proposed mechanism synthesis procedure: (a) generating link families for DOF (b) generating topologies for link family (c) generating linkage arrangements for topology and link soup (d) generating geometric solutions for linkage arrangement.

criterion reports the correct value of DOF for a mechanism, an initial set of topologies whose mobility is at least the desired DOF value is generated. This is followed by a mobility analysis of the resulting topologies to filter out the candidates that contain over-constrained regions.

Without loss of generality, we account only for revolute and prismatic joints, since other joints can be replaced by a combination of joints from these two types. Under these assumptions, the Grübler-Kutzbach criterion becomes

$$DOF = \lambda |L| + (1 - \lambda) |J| - \lambda. \quad (1)$$

In this equation, L and J are respectively the sets of links and joints and $|X|$ reflects the cardinality of a set X . Parameter λ indicates the DOF of a single rigid link before its motion is constrained. This number equals 3 for planar and spherical mechanisms, and 6 for spatial linkages.

Equation (1) yields the degrees of freedom of a given topology. To address the inverse problem, we first find a list of *link families* that satisfy equation (1), and then enumerate topologies for each family. By definition, a link family or *link assortment* is a tuple of numbers that indicate how many links of each type are present in the mechanism, so a link family represents multiple topologies. Let $L_i \subset L$ represent the set of links in a mechanism with degree i , or, alternatively, with i attachment nodes. An example of a link family in the plane is $(|L_2| = 4, |L_3| = 2)$, which represents all the 6-bar planar mechanisms that have 4 binary and 2 ternary links.

2.1.1 Enumerating Link Families For a Given DOF

Given a DOF we construct the link families in two steps. First, we determine all possible pairs of $(|L|, |J|)$ that satisfy equation (1) by considering that $|L|$ and $|J|$ must be integers, which implies that equation (1) can be rewritten as:

$$|L| = (\lambda - 1)k + DOF + 1 \quad (2a)$$

$$|J| = \lambda k + DOF \quad (2b)$$

where k can be any non-negative integer number. Next, for each pair $(|L|, |J|)$ we generate a set of link families such that the following two equa-

tions hold:

$$|L| = \sum_i |L_i| \quad (3a)$$

$$2|J| = \sum_i i \cdot |L_i| \quad (3b)$$

where $|L_i|$ is the number of links with degree i . In order to generate the link families, equations (3a) and (3b) must be solved simultaneously, which can be viewed as determining different $|L_i|$ values. This is accomplished by recursively breaking the problem into two subproblems: (1) selecting the $|L_i|$ value for the smallest i (2) selecting the rest of $|L_i|$ values. The base case in the recursion happens when only one $|L_i|$ remains to be selected, which is trivial.

For example, consider the case of planar and spatial mechanisms with $DOF = 1$, which contain up to 8 links. The following $(|L|, |J|)$ pairs will simultaneously satisfy equations (2a) and (2b): $\{(|L| = 2, |J| = 1), (|L| = 4, |J| = 4), (|L| = 6, |J| = 7), (|L| = 8, |J| = 10)\}$ for planar and $\{(|L| = 2, |J| = 1), (|L| = 7, |J| = 7)\}$ for spatial mechanisms. For the $(|L| = 6, |J| = 7)$ pair, it is known that $|J| - |L| + 2$ is an upper bound for the number of nodes on each link of a generic mechanism, which equals 3 in this case. Therefore, the highest order link would be ternary, and equations (3a) and (3b) become:

$$6 = |L_1| + |L_2| + |L_3| \quad (4a)$$

$$14 = |L_1| + 2|L_2| + 3|L_3| \quad (4b)$$

By assigning an integer value to $|L_1|$ between 0 and 2, equations (4) become a system of two equations with the two unknowns $|L_2|$ and $|L_3|$, which will produce three families: $\{(|L_1| = 0, |L_2| = 4, |L_3| = 2), (|L_1| = 1, |L_2| = 2, |L_3| = 3), (|L_1| = 2, |L_2| = 0, |L_3| = 4)\}$. Note that assigning other values to $|L_1|$ results in invalid answers (i.e. negative $|L_i|$'s).

2.1.2 Enumerating Topologies For Each Link Family

We can now proceed to enumerate topologies that belong to each family. In the graph representation of the linkages, vertices and edges correspond to links and joints, and the degree of a vertex specifies the type of the link. Also, each vertex or edge has an assigned label. Two topologies that result from *different* label assignments of the *same* connectivity between links are called isomorphic. It is worthwhile noting that topologies produced

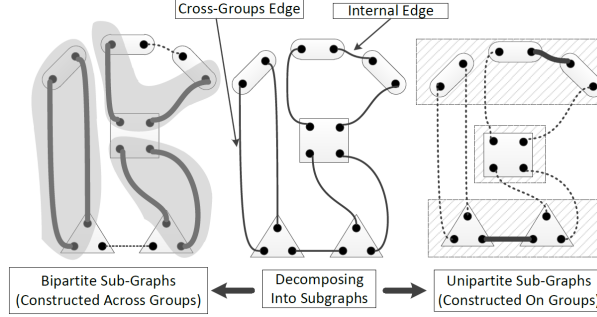


Figure 2: Vertices are grouped by their degrees and the graph is decomposed into two sets of subgraphs: the *unipartite* subgraphs formed within a degree group, and the *bipartite* subgraphs formed across degree groups.

from different families cannot be isomorphic. However, topologies that are generated from the same family are often isomorphic, so by eliminating isomorphic topologies during the enumeration one would dramatically reduce the number of computations required during the later stage of this synthesis process. We address this problem by using a divide-and-conquer algorithm with extensive branching, and aim to fragment the problem into the largest possible number of non-overlapping subproblems as discussed next.

Clustering Graph Vertices By Their Degree Given a link family, the first step is to cluster the graph vertices by their degree into the so called *degree groups*. Enumerating all graphs belonging to a link family is equivalent to finding all possible ways that connections (edges) may be formed between vertices of the graph such that the degrees of all vertices are matched. At this point, we introduce a decomposition of the full graph into two different sets of subgraphs: the *unipartite* subgraphs, constructed only with the vertices belonging to one degree group (Figure 2), and the *bipartite* subgraphs formed by connecting vertices from two different degree groups (Figure 2). Taking a divide-and-conquer approach, we enumerate the resulting graphs in two steps: first, we find all the different ways in which the edges can be distributed among the subgraphs; then we perform the graph enumeration for each case.

Distributing Edges Among Subgraphs Figure 3 demonstrates the *coarse adjacency matrix* of a generic graph for which the vertices are partitioned into n groups based on their degrees. Let element $\bar{a}_{ij}$, represent

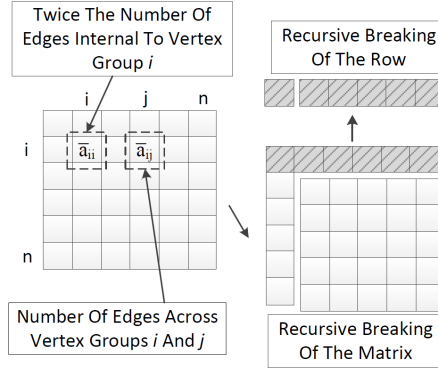


Figure 3: Different edge distributions among the subgraphs are computed in a divide-and-conquer fashion to specify the coarse adjacency matrix.

the number of connections between vertices of group i and group j . When $i = j$, $\bar{a}_{ii}$ equals twice the number of the internal edges in group i . Finding different edge distributions among the subgraphs is equivalent to producing different allocations of positive numbers to the elements of this matrix in such a way that the three following conditions hold: (1) the summation of elements of row i equals the summation of degrees of vertices in group i (2) the matrix is symmetric, and (3) the numbers allocated to the diagonal elements are even. Thus, we can generate matrices satisfying these conditions in a straightforward divide-and-conquer fashion. At each level, the problem of specifying the matrix elements is partitioned into two subproblems: assign values to the first row (column), and allocate numbers to the submatrix resulting from the removal of the first row and the first column of the original matrix. The base case happens when the matrix carries only one element, when the assignment becomes trivial. Completing a row is also performed using a divide-and-conquer method where a row is split into its first element and a sub-row with the remaining elements in the original row. The base case here happens when the row has only a single element.

As an example, consider one of the previously identified link families: $(|L_1| = 0, |L_2| = 4, |L_3| = 2)$. Since $|L_1| = 0$, the vertices are grouped into two classes corresponding to binary and ternary links. The coarse adjacency matrix is a 2×2 matrix with three independent elements: (1) $\bar{a}_{11}$ being twice the number of edges internal to the group of binary vertices (2) $\bar{a}_{12}$ or equivalently $\bar{a}_{21}$ being the number of cross edges between binary and ternary vertices (3) $\bar{a}_{22}$ being twice the number of edges internal to the group of ternary vertices. Now, we must assign values to these three elements. To

insure that the assigned values produce a valid solution, two conditions must be satisfied: $\bar{a}_{11} + \bar{a}_{12} = 8$ and $\bar{a}_{12} + \bar{a}_{22} = 6$. As described earlier, we proceed by: (1) specifying the elements of the first row, namely, allocating numbers to $\bar{a}_{11}$ and $\bar{a}_{12}$, and (2) by specifying the elements of the submatrix obtained by removing the first row and column, namely, allocating a number to $\bar{a}_{22}$. The first sub-problem, itself is broken down into selecting a value for $\bar{a}_{11}$ and then for $\bar{a}_{12}$. For instance, let's have $\bar{a}_{11} = 2$. This will imply $\bar{a}_{12} = 6$, which in turn yields $\bar{a}_{22} = 0$. The other valid solutions are $(\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2)$, $(\bar{a}_{11} = 6, \bar{a}_{12} = 2, \bar{a}_{22} = 4)$ and $(\bar{a}_{11} = 8, \bar{a}_{12} = 0, \bar{a}_{22} = 6)$.

Setting the Elements Of the Fine Adjacency Matrix Although the coarse adjacency matrix informs us about the number of edges in different subgraphs, it does not specify the vertices defining these edges within each of these subgraphs. Therefore, the next step in the graph enumeration process is to capture all different patterns for formation of these subgraphs, given a coarse adjacency matrix. This is equivalent to producing different combinations of elements in the *fine adjacency matrix*. Figure 4 shows the fine adjacency matrix for a generic graph. In this matrix, element of row k and column l , a_{kl} , corresponds to the number of edges drawn between vertices k and l . A fine adjacency matrix represents a valid graph if the following criteria are met: (1) the summation of elements of row k must equal the degree of vertex k , and (2) the matrix must be symmetric. We accomplish the graph enumeration as follows: (a) on each vertex group i , we construct all possible unipartite subgraphs that carry $\bar{a}_{ii}/2$ edges, which is equivalent to listing all the solutions for the square submatrices placed on the diagonal of the fine adjacency matrix, as illustrated in Figure 4; it should be noted that the solution for each of these subgraphs is independent from the solution for any other subgraph at this step; and (b) for each two groups i and j , we form bipartite graphs across their vertices with $\bar{a}_{ij}$ edges. This is equivalent to listing solutions for the remaining rectangular submatrices in the fine adjacency matrix. In contrast to the previous step, the solution for each of these subgraphs is impacted by the solution of other subgraphs in this step and those from the previous step. In the following, we elaborate each of these steps in detail.

As an example, consider one of the coarse adjacency matrices we listed above for a 6-bar linkage: $(\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2)$. Let v_1, v_2, v_3 and v_4 represent the binary vertices and v_5 and v_6 be the ternary vertices. The task of building graphs for the given coarse adjacency matrix is split into generating instances for three subgraphs: (1) those on $\{v_1, v_2, v_3, v_4\}$

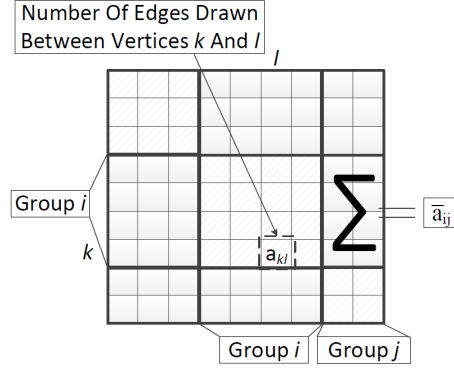


Figure 4: Graph enumeration is equivalent to finding solutions for the fine adjacency matrix. The task is split into generating two sets of subgraphs: the ones that are constructed on the vertices of the unipartite subgraphs, which correspond to square submatrices along the diagonal of the fine adjacency matrix, and the bipartite subgraphs, which correspond to the remaining rectangular submatrices. Note that the summation of elements in the submatrix associated with the intersection of groups i and j reflects $\bar{a}_{ij}$.

with exactly 2 edges, such that the maximal degree of any vertex is 2; (2) subgraphs on $\{v_5, v_6\}$ with exactly 1 edge such that the maximal degree of any vertex is 3, and (3) bipartite subgraphs across the two groups with exactly 4 edges such that the maximal degree of any vertex from the first group is 2 and the maximal degree of any vertex from the second group is 3. Note that degrees are counted by accounting for edges that are formed in this step and also in the two previous steps.

Similarity in Graphs and Isomorphism Detection Before we proceed with detailing the steps of graph enumeration or equivalently generating adjacency matrices, we need to review the concept of vertex *similarity* in graphs. We use the concept of vertex similarity to infer the occurrence of graph isomorphism in different levels of recursion. Note that the sooner the isomorphism is identified in the recursion, the larger the eliminated chunks of the design space would be, leading to a more efficient synthesis procedure.

Two vertices u and v in a graph G are similar if some automorphism of G maps u onto v [19]. In other words, two similar vertices are indistinguishable up to their labels. Vertex similarity is also directly related to the concept of graph isomorphism. For any two isomorphic graphs, one can find a one-

to-one similarity relation between the sets of vertices and edges of the two graphs. In other words, detecting similarities in graphs can be used to detect or even avoid isomorphism in our graphs.

We know that different assignments of labels to one network of vertices results in isomorphic topologies. Equivalently, isomorphic topologies can be generated by shuffling the appropriate rows and columns of the corresponding fine adjacency matrix. This fact can be used in detecting similarity or isomorphism by simply examining different valid rearrangements of rows and columns in the adjacency matrices. Nevertheless, the computational complexity of doing so exhaustively can be significant. Instead, we offer a heuristic method for detecting similarities in the graph, which collects specific topological information about vertices and vertex pairs. These values are collected in a symmetric matrix, the so-called *characteristic matrix*, in which the off-diagonal element $n_{i,j}$ captures information about the topological ‘relationship’ between vertices i and j , and the diagonal element $n_{i,i}$ reflects the topological characteristic of vertex i in the graph. Importantly, if two vertices correspond to the same diagonal value in this matrix, they will be declared similar. In addition, if two pairs (v_1, v_2) and (v_3, v_4) correspond to the same off-diagonal value, then the topological ‘relationship’ between the first pair of vertices is similar to that of the second pair. To populate the characteristic matrix, we iteratively update its elements a predefined number of times, and our experience shows that 3 updates are sufficient.

The two key components of this process are: (1) a set of topological variables that are easy to evaluate at every stage, such as degree of a vertex, the number of neighbors of a vertex, the common neighbors between two vertices, as well as the value of the characteristic matrix elements from the previous iteration; and (2) a real-valued characteristic function that takes these topological variables as input and outputs a number that is the same for similar vertices or vertex pairs, and is different for dissimilar vertices or vertex pairs. The characteristic matrix is initialized to zero, then iteratively updated by re-evaluating this characteristic function for every element of the matrix. As an example, consider the graph shown in Fig 5 along with the characteristic matrix determined with this heuristic method. The numeric values have been replaced with letters in this example to simplify the observation of similarities. The diagonal of the characteristic matrix suggests that vertices are grouped into three similarity categories: $\{v_1, v_3\}$, $\{v_2, v_4\}$ and $\{v_5\}$, which can be intuitively observed by looking at the graph. In addition, the off-diagonal values of the characteristic matrix inform us that all 10 vertex pair relations fall into 6 different categories: $\{(v_1, v_2), (v_3, v_4)\}$, $\{(v_2, v_3), (v_1, v_4)\}$, $\{(v_4, v_5), (v_2, v_5)\}$, $\{(v_1, v_5), (v_3, v_5)\}$,

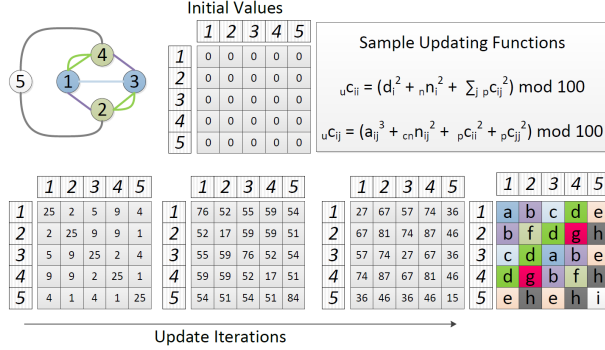


Figure 5: The characteristic matrix captures the similarities between vertices or vertex pairs. In the sample update functions, $p_{c_{ij}}$ and $u_{c_{ij}}$ are respectively the previous and updated values of the element of row i and column j of the characteristic matrix, n_i is the number neighbors of vertex i , $c_n n_{ij}$ is the number of common neighbors between vertices i and j and a_{ij} is the element of the the adjacency matrix, reflecting the number of edges across vertices i and j .

$\{(v_2, v_4)\}, \{(v_3, v_1)\}$.

It is worth emphasizing that the selection of an effective characteristic function has a key influence on the number of update iterations. As an example, consider choosing a characteristic function $f(x_1, x_2)$, which takes two variables as its input, and should differentiate between, say, $(2, 6)$ and $(3, 5)$. It is obvious that $f(x_1, x_2) = x_1 + x_2$ is not a good choice, since it cannot reflect the differences between the two sets of inputs, but $f(x_1, x_2) = 2 \times x_1 + 3 \times x_2^2$ might be a better selection. Also note that as we update the characteristic matrix, additional topological information is generated between the graph entities, resulting in more topological details about the graph. Our selection process is as follows: a list of functions, whose forms suggest their potential to distinguish between different sets of inputs are compiled in a list; then tests are performed on them to measure their efficacy in correctly finding similarities; the best performing functions are selected to be used for isomorphism detection.

The nature of the described process implies the possibility of false positive occurrence in the isomorphism test, if the updating functions are not selected carefully, meaning that two non-isomorphic topologies may be identified as isomorphic. Although we have illustrated in section3 that with good selection of the updating functions, not even a single such case is observed, we also propose a solution for when the functions are poorly chosen: the rows

and columns of the adjacency matrices associated with each pair of topologies which have been identified isomorphic are rearranged as suggested by their corresponding characteristic matrices and then the resulting adjacency matrices are compared to test the isomorphism hypothesis. This additional step unlike the traditional shuffling method does not require exhaustive investigation of all possible rearrangements and thus is less expensive than it by orders of magnitude.

Constructing Unipartite Subgraphs As already mentioned, once the number of edges allocated to each subgraph is known (i.e. when we have the coarse adjacency matrix), the first phase in the graph enumeration stage is to generate unipartite subgraphs (i.e. specifying the elements of the square submatrices located on the diagonal of the fine adjacency matrix). We again apply the divide and conquer paradigm to classify subgraphs constructed on a group of vertices into distinct link families. The link family here is determined by the vertices of a given degree group, and the degrees are internal to that particular group, i.e. they reflect the number of connections for each vertex inside the vertex group. Thus, the task is to find all the valid link families for the group, and we follow the same process as the one described above. However, here vertices of the group are partitioned based on their degree within the subgraph. To exemplify, consider the case of enumerating graphs for the 6-bar linkage with $(\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2)$ as its coarse adjacency matrix, for which the first two tasks focus on constructing unipartite subgraphs. For the first degree group, we have $|L| = 4$ and $|J| = 2$, which gives us the following families: $(|L_0| = 0, |L_1| = 4, |L_2| = 0)$, $(|L_0| = 1, |L_1| = 2, |L_2| = 1)$ and $(|L_0| = 2, |L_1| = 0, |L_2| = 2)$. For the second degree group, we have $|L| = 2$ and $|J| = 1$, for which we can generate the following families: $(|L_0| = 1, |L_1| = 0, |L_2| = 1)$ and $(|L_0| = 0, |L_1| = 2, |L_2| = 0)$.

Base Cases for Unipartite Graphs The base case for the vertex partitioning into degree groups is reached when all vertices in each group have the same internal degree, when the problem becomes one of generating *regular* graphs [20]. By definition, an r -regular graph is one for which the degree of all the vertices is equal to r . The following three scenarios may arise in the process:

1. $r \leq 2$:
 - (a) $r = 0$: only one graph can be generated, which consists of $|L|$ single isolated vertices;

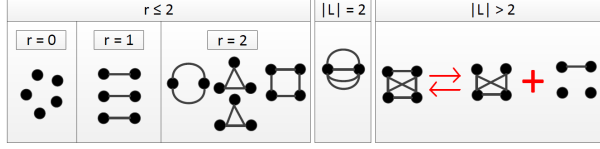


Figure 6: The base cases that appear in the recursive process of topology enumeration.

- (b) $r = 1$: can result in a valid graph only when $|L|$ is an even number. The only feasible graph would have $|L|/2$ edges with two vertices on the two ends of each;
 - (c) $r = 2$: the graph would be a collection of isolated circular sub-graphs from different *sizes* reflecting the number of vertices involved in the cycle. Enumerating graphs for this case is equivalent to specifying the number of cycles from each size. For this, we again employ a divide-and-conquer approach: let C_i denote the set of cycles with i vertices. We set the value of $|C_i|$ for the smallest i , followed by solving the subproblem for the remaining $|C_i|$ numbers. For instance, let's say we have $|L| = 5$ and $r = 2$. Since self-loops⁴ are not allowed unless we are dealing with contracted graph, we know that $2 \leq i \leq 5$. Selecting $|C_2| = 0$, we come up with $|C_3| = 0$, $|C_4| = 0$ and $|C_5| = 1$. This set of numbers represents a cycle with 5 vertices. However, if $|C_2| = 1$, we have $|C_3| = 1$, $|C_4| = 0$ and $|C_5| = 0$ which reflects two cycles with sizes 2 and 3.
2. $r > 2$ and $|L| = 2$: only one graph can be generated without self-loops, which consists of two vertices connected by r parallel edges.
 3. $r > 2$ and $|L| > 2$: This situation can be addressed by decomposing the regular graph into two subgraphs: (1) one in which two vertices v_1 and v_2 have degree $r - 1$ and the remaining vertices have degree r (2) one edge between v_1 and v_2 . We first enumerate all possible cases for the first subgraph, as its vertices can be partitioned into two groups. Next we add one edge to each subgraph, wherever one is missing. These cases are summarized in Fig 6.

⁴A self-loop is an edge that connects a vertex to itself [21]

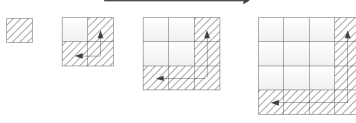


Figure 7: Starting from the subgraphs constructed on the first vertex group, one group is added at each step. For the new group, the unipartite graphs are constructed on the vertices of the group. As well, bipartite subgraphs are constructed between the vertices of the group and the vertices of groups preceding it, one at a time.

Enumerating Bipartite Subgraphs Following the construction of unipartite subgraphs, the next phase in the graph enumeration process is to form the bipartite graphs or, equivalently, specifying the elements of the remaining (off-diagonal) rectangular submatrices in the fine adjacency matrix. The order of specifying the elements of submatrices in the fine adjacency matrix is illustrated in Fig 7. Starting from the solutions for the submatrix associated with the subgraph constructed on the first group, at each level we add one new vertex group and accomplish two tasks: (1) choosing the elements of the diagonal square matrix from the previously determined solutions for unipartite subgraphs (2) specifying the elements of the rectangular submatrices associated with the bipartite subgraphs constructed across vertices of the new group and the vertices of groups preceding it, one at a time. The set of solutions obtained after adding each subgraph (submatrix) is tested for isomorphisms; if an isomorphism is detected it is eliminated before the next update. For the intermediate adjacency matrix solutions associated with intermediate graph solutions formed in the enumeration process, the elements of the submatrices that are not specified are set to zero, reflecting the fact that no connections are yet established in those regions of the graph.

Constructing bipartite subgraphs is equivalent with specifying the elements of the rectangular submatrices (Figure 8). Let L_i and L_j represent two vertex groups, across which we want to form bipartite graphs. From the coarse adjacency matrix, we have the total number of edges present in the subgraph equal to $\bar{a}_{ij}$. This means that the summation of elements in the corresponding submatrix must be $\bar{a}_{ij}$. Due to the symmetry of the fine adjacency matrix, two rectangular submatrices, which are transpose of each other, represent the subgraph. Assuming that we are considering the one in the upper triangle, the summation of elements in a row of the submatrix

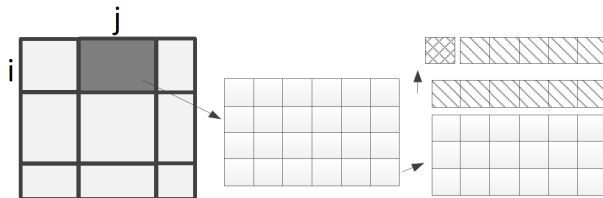


Figure 8: The divide-and-conquer approach for specifying the elements of the rectangular submatrix.

reflects the degree of the corresponding vertex in group i that is internal to the bipartite graph. Similarly, the summation of elements in a column of the submatrix relates to the degree of the associated vertex in group j internal to the bipartite graph. Specific criteria must be met for specifying the elements of the submatrix to represent a valid bipartite subgraph solution. The internal degree of each vertex, from either of the two groups, plus its computed degree from the previously formed subgraphs, must not exceed the prescribed overall degree of the vertex in the graph. This condition will impose upper bounds for the summation of elements in individual rows and columns of the submatrix which then will be used toward finding solutions for the rectangular submatrix. Also if j reaches its limit (i.e. when we are considering the bipartite subgraph across the last group and other groups), an additional condition must be considered, namely the overall degree of vertices of group i must be matched.

Given the upper bounds for the summation of elements in individual rows and columns of the submatrix, the enumeration of solutions is recursively partitioned into two sub-problems: (1) define the elements of the first row (2) define a smaller submatrix, which is obtained by removing the first row from the original submatrix. The base case happens when we reach a submatrix with only one row. Solving for a row matrix is also done by taking a divide-and-conquer approach in which, at each step, a row is divided into its first element and a sub-row with one less element. The base case happens when the row has only one element, finding a solution for which is trivial (Fig. 8). Note that, when we break the problem into sub-problems, the upper bounds on the summation of elements of rows and columns are updated accordingly.

As an example, consider the case of the 6-bar linkage. We showed that several coarse adjacency matrices could be produced for such a linkage, one of which is given by $(\bar{a}_{11} = 4, \bar{a}_{12} = 4, \bar{a}_{22} = 2)$. Consequently, for this case there are 2 edges internal to the group of binary vertices, 1 edge internal to

the group of ternary vertices and 4 cross-edges belonging to the bipartite subgraph across the two vertex groups (Fig. 9). We showed that different internal families could be found for the two unipartite subgraphs constructed on the two vertex groups of binary and ternary. Specifically, consider $(|L_0| = 1, |L_1| = 2, |L_2| = 1)$ for the first group and $(|L_0| = 0, |L_1| = 2, |L_2| = 0)$ for the second group (Fig. 9). The group of binary vertices has been divided into three subgroups, while the group of ternary vertices carries only one subgroup. In other words, for the group of ternary vertices we have reached the base case, and the two vertices of the group can be connected in a unique way (see Figure 6). However, for the group of binary vertices we continue the divide and conquer partitioning and generate coarse adjacency matrices for the sub-groups of the binary vertices group. The only possible coarse adjacency matrix is the one shown in Fig 9. The non-zero values of the matrix imply that there must be two connections between the sub-group with internal degree 2 and the sub-group with internal degree 1, as can be seen in Figure 9.

So far, the internal edges have been placed inside the vertex groups. The next step is to make 4 connections across the binary and ternary vertex groups, for which we need to define a 4×2 rectangular matrix. Each vertex has to have a number of connections with other vertices equal to its degree. Some of these come from the internal degrees, while the remaining connections will follow from those formed in the bipartite graphs. We note that the elements of the rectangular submatrix must be prescribed according to the summation criteria shown in Fig. 9. By doing so, we obtain a set of rectangular matrices, which are then combined with the rest of the adjacency matrix, and followed by isomorphism detection and elimination as outlined above. By allowing only for single edges, this process outputs one solution, namely the Watt topology of a 6-bar linkage.

Contraction/Expansion In most practical cases, the minimum vertex degree in a topology is 2. Also binary links are more common in the enumerated families, i.e. $0 < |L_2|/|L| < 1$, in which case it is a common practice to employ the *contraction/expansion* method [1], which drastically speeds up the enumeration process. By definition, a contracted topology is obtained by replacing each binary link chain of a topology with one single edge. The contraction/expansion consists of enumerating contracted families, enumerating topologies for them and then expanding the topologies by distributing binary vertices among their edges (Fig 10). Note that, using a contraction/expansion technique requires self-loops in the contracted

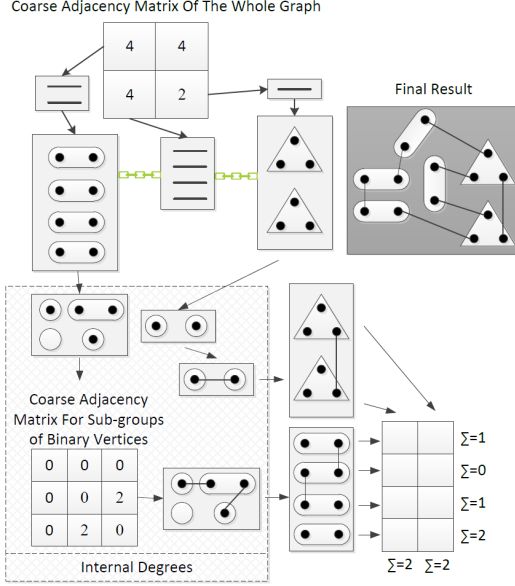


Figure 9: An example of a graph for a link family.

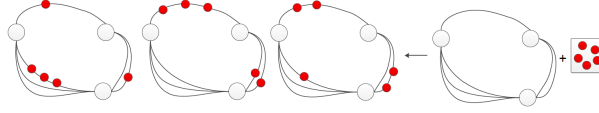


Figure 10: Distributing 5 binary vertices among the edges of a contracted topology.

topologies, which would require slight modifications to the process described above.

Eliminating Topologies With Undesired DOF As we mentioned earlier, the Grübler-Kutzbach criterion may fail to report the true mobility of mechanism when over-constrained regions occur in the corresponding topologies. These cases can be detected and eliminated by performing an additional mobility analysis, and we use the technique presented in [11].

2.2 Generating Linkage Arrangements

The process described above produces valid candidate topologies. The next step is to construct, for each candidate topology, the corresponding linkage

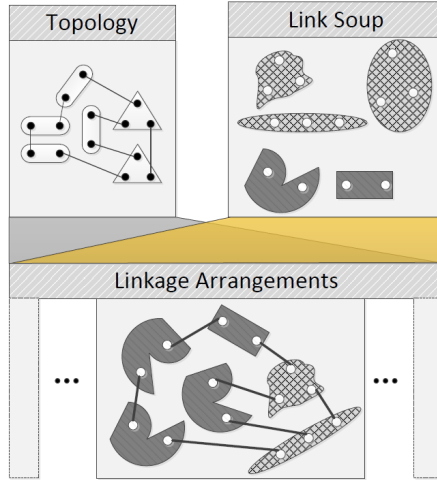


Figure 11: An example of a linkage arrangement for the Watt topology with links selected from a given link soup.

arrangements to specify: (1) which link soup component can be assigned to which vertex of the topology, and (2) what is the order in which we can pair up the connection points in adjacent links. In other words, this stage is not focused on the detailed geometry, but only on the types of links and connections between them.

We observe that the existence of similarities in a given topology can lead to redundancy in the linkage arrangement enumeration. For this reason, we enumerate the valid linkage arrangements in two steps: first, we enumerate all possible linkage arrangements, and then we use a slightly modified version of our heuristic method for similarity detection (the link type and the pairing information are inputs at each iteration), to identify and eliminate redundancies. Figure 11 shows one linkage arrangement for the Watt topology by using link components from a link soup that has two binary links and three ternary links.

2.3 Geometry

The open question that remains to complete our synthesis procedure is the following: given a linkage arrangement, can one find valid geometric solutions that would close the kinematic loops to satisfy the prescribed DOF and avoid geometric interference between the physical links? To answer this question, we first produce a reference geometry of the linkage, and then solve

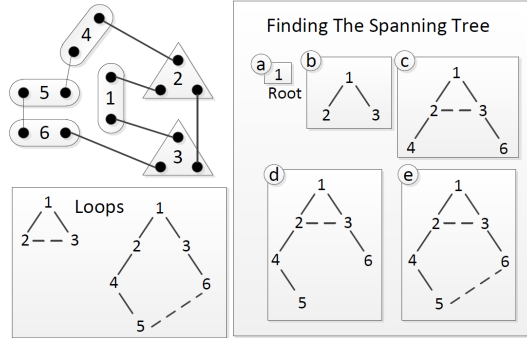


Figure 12: Finding a spanning tree for the Watt topology of the 6-bar linkages. Every (topological) edge that is not present in the spanning tree (shown with dotted lines) closes a loop, and is called a *chord*.

for the geometric constraints as described below.

Reference Geometry of the Linkage The dependencies between the joint variables that are created by the loops prevents an arbitrary selection of the joint variables. Consequently, we determine a set of *independent* joint variables by constructing a spanning tree for the corresponding topology. Starting from an arbitrary vertex as the root, we find the spanning tree of the topology using a breadth-first search protocol (Fig. 12). The collection of joint variables associated with the edges of the spanning tree describes a set of independent joint variables that can be used to fully describe the configuration of the linkage with respect to a reference geometry. In turn, the reference geometry is defined using the breadth-first search algorithm as follows: starting from a fixed geometric pose of the root link, we adjust at each level of the hierarchy the poses of the children to match the joint alignment between child and parent. This process is repeated until we find the reference poses for all links (Fig 13). The driving parameters of all joint variables (angle for revolute joints and distance for prismatic joints) are set to zero. Every edge of the (topological) graph that is not present in the spanning tree closes a loop in the graph, and these edges are called *chords*. By starting from the two ends of the chord, we traverse the tree upwards until the two paths intersect at one node, the so called loop root. The union of nodes and edges that have been visited in this process form the loop (Fig. 12).

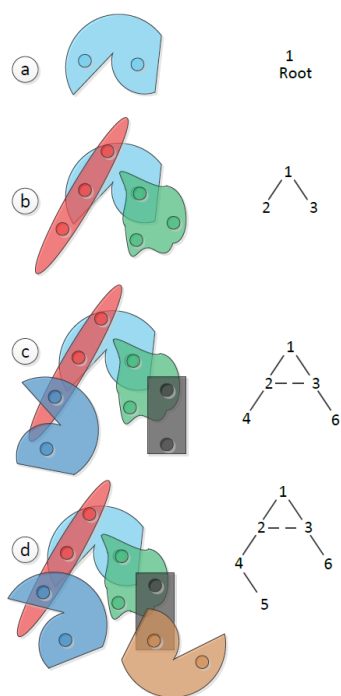


Figure 13: Determining the reference geometry for the linkage arrangement for the Watt topology of the 6-bar linkage.

Loop Closure Through Geometric Constraints The geometric description of the linkage now allows us to solve for the geometric constraints. The joint alignment condition has already been satisfied for the connections between children and parents in the spanning tree. This leaves us with two remaining geometric constraints: (1) joint alignment must be satisfied for the chords, which corresponds to the closure of the loops, and (2) physical clashes must be avoided.

The valid solutions are provided by those sets of joint variables for which the two constraints are simultaneously satisfied. Since an analytical solution is in general unavailable, we formulate the linkage closure as an optimization problem that aims to minimize an error measuring how ‘far’ a given configuration of the mechanism is from the configuration that simultaneously satisfies the constraints. Among the existing optimization techniques, we have selected a coordinate descent method (we discussed why in optimization section) as a minimization procedure, which can be trapped in local minima. We used established random sampling methods as initial conditions for the minimization step.

Defining the Objective The objective function is defined in terms of a geometric error that approaches zero as the configuration of the mechanism approaches the ideal constraint satisfying configuration. The threshold for an acceptable error varies depending on the selection of the link soup. For instance, for the case of molecular components, the tolerances of bond angles and bond lengths about the average values are the determining factors. Alignment of a revolute joint requires collinearity of the 4 points that define the joint axes on the two components of the joint, the axes to be in opposite directions and the corresponding ‘base’ points to be at a distance prescribed by the specific link soup (Fig. 14(a)). The alignment of a prismatic joint also needs the collinearity of the points defining the joint axes and the axes to be in opposite directions. In addition it requires the so called *guide unit vectors* (perpendicular to the joint axes) on the two joint components to be parallel (Fig. 14(b)). Standard distance evaluations are used to eliminate configurations that produce clashes between components, which very well fits the case of molecular components made of spherical atoms (Fig. 14(c)). The total error is defined as a weighted sum of all the individual errors.

Optimization Starting from each sample point, we run an optimization procedure to minimize the predefined error. Our strategy is to include only the loop closure criteria in the objective function, i.e. to ignore the effect of

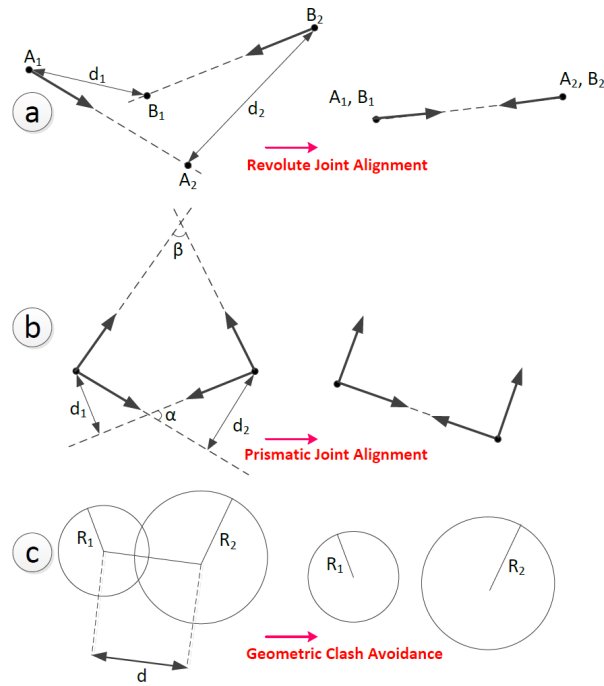


Figure 14: (a) For revolute joint alignment, A_1 must coincide with B_1 and A_2 must coincide with B_2 . (b) For prismatic joint alignment, α , β , d_1 and d_2 must be zero. (c) For clash to be avoided between a pair of spheres, d should be greater than $R_1 + R_2$.

geometric clashes during the optimization process, followed by an investigation of the collisions in the final configuration and the corresponding range of motion. Adjusting the joint variables to close loops is one of solving the inverse kinematic (IK) problem. In [22], advantages as well as limitations of several algebraic, geometric, and iterative IK solutions have been reviewed. Specifically a comparison has been made in [23] of various aspects of jacobian transpose method and the cyclic coordinate descent (CCD) method, including their convergence times. For our application, we have selected a CCD method. In general, CCD methods converge to solution for the majority of initial conformations, and fewer parameters must be tuned for them [24]. We iteratively lock all but one of the joint variables at one instantaneous configuration of the mechanism, and seek the optimum value for that one joint variable [16]. We then iterate through all the independent joint variables identified by the spanning tree described above, and we evaluate the closure error as the joint variables are being modified. The procedure that we use here has certain computational advantages as described below.

Two different formulations are introduced for finding the optimum value of the joint variable depending on whether the joint is revolute or prismatic. For a revolute joint, we define a local frame at the base of the joint, for which the joint variable must be adjusted. The z axis of the frame will be in the direction of the joint unit vector. We transform the problem of joint alignment into an alignment of a number of pairs of points. Let $\{(P_i, Q_i)\}$ ($1 < i < n$) represent the coordinates of the set of point pairs whose distances must be minimized in the local frame, where P_i is the moving element and Q_i is the fixed one in space. If we denote by θ the amount of change in the local joint variable, and by P_i^0 the position of P_i associated with $\theta = 0$, we can obtain P_i at any given θ :

$$P_i = RP_i^0 \quad (5)$$

where R is the rotation matrix about the joint unit vector and can be written as:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The closure error for any value of θ can be defined as:

$$e = \sum_{i=1}^n |Q_i - P_i|^2 \quad (6)$$

By substituting equation (5) into (6) we obtain

$$\begin{aligned}
e = & \sum_{i=1}^n (|Q_i|^2 + |P_i^0|^2) \\
& - \left(\sum_{i=1}^n (2Q_{i,y}P_{i,y}^0 + 2Q_{i,x}P_{i,x}^0) \right) \cos\theta \\
& - \left(\sum_{i=1}^n (2Q_{i,y}P_{i,x}^0 - 2Q_{i,x}P_{i,y}^0) \right) \sin\theta,
\end{aligned} \tag{7}$$

which can be rewritten in the following form:

$$e = a - b\cos\theta - c\sin\theta. \tag{8}$$

By defining $\sin\alpha = c/\sqrt{b^2 + c^2}$ and $\cos\alpha = b/\sqrt{b^2 + c^2}$, we have

$$e = a - \sqrt{b^2 + c^2} \cos(\theta - \alpha) \tag{9}$$

in which e attains its minimum value when $\theta = \alpha + 2k\pi$.

For a prismatic joint, we define a local frame whose z axis is again alongside the joint axis. If q is the amount of change in the prismatic joint variable, and P_i^0 is the position of P_i associated with $q = 0$, we have:

$$P_i = P_i^0 + \begin{bmatrix} 0 \\ 0 \\ q \end{bmatrix}.$$

The error value is defined similarly:

$$e = \sum_{i=1}^n |Q_i - P_i^0|^2 + \sum_{i=1}^n 2(Q_{i,z} - P_{i,z}^0)q + \sum_{i=1}^n q^2 \tag{10}$$

which can be rewritten in the following form:

$$e = aq^2 + bq + c \tag{11}$$

whose minimum occurs when $q = -b/2a$.

For those synthesized mechanisms whose loops close without geometric interference we must finally determine the range of valid motions, i.e., the range of joint variables for which the constraints on loop closure and interference free motions hold.

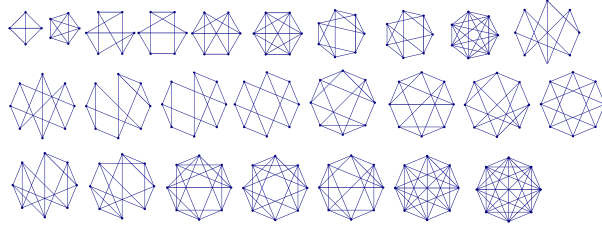


Figure 15: r -regular graphs with $r > 2$, with up to 8 vertices, and containing only single edges.

3 RESULTS

We developed several tests to investigate the viability and efficiency of the individual modules of the proposed method. We have also run our implementation on a more realistic test case to examine the practicality of the proposed method.

Testing Validity Of The Topology Enumeration Module As a benchmark for the topology enumeration technique, we generated the r -regular graphs with $r > 2$, up to 8 vertices, and containing only single edges, which are shown in Fig. 15. The results perfectly matched the results reported in [17] and show that the proposed method enumerates all such graphs.

Testing the Heuristic Similarity Identification Module The viability of the heuristic method in capturing the topological similarities and isomorphism detection was tested by generating pairs of topologies: the first topology was generated randomly; the second one could either be obtained by shuffling the rows/columns in the adjacency matrix of the first topology, resulting in a pair of isomorphic topologies, or could be produced randomly and independently of the first topology. Each pair was input to the proposed heuristic technique as well as a traditional implementation of the isomorphism detection (brute force trial of various rearrangements of rows/columns [1]). The results showed that the proposed heuristic method produced the correct answer in all test cases. Furthermore, the average CPU times of the two methods for 20 pairs of topologies with increasing numbers of vertices are shown in Figure 16. The results indicate that the average CPU time of the traditional method grows exponentially as the size of the topology increases in contrast to the CPU time of the proposed heuristic method.

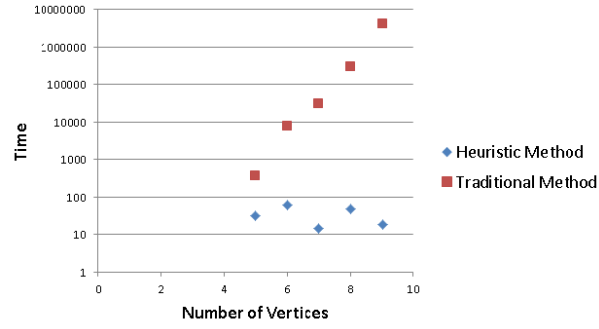


Figure 16: Average CPU time (measured in CPU CLOCKS) of the traditional and the proposed heuristic technique for isomorphism detection in 20 pairs of topologies.

Topology	Random Linkage with Closed Loops	Scrambled Linkage	Solution1	Solution2	Solution3

Figure 17: Closing open loops with known solutions.

Validating The Optimization Module For Loop Closure To test the capabilities of the optimization module, we constructed pre-closed linkages by picking the base points and rotation axes for each joint of the linkage, which provided linkages for which loop closure was satisfied by construction. Then the loop closure was perturbed by randomly modifying the joint variables to produce an open kinematic loop. The optimization module was then used to search for the loop closure configuration of these linkages. For each linkage, a number of solutions were obtained, which may or may not fall into the same solution branch for the mechanism. Figure 17 demonstrates this procedure for a number of spatial topologies, and we show the first three solutions that we generated for each topology. Note that the binary links are represented with single bars with two spheres at the ends, whereas the ternary links are represented by triangles consisting of 3 bars and spheres at the corners. The line segments protruding from the spheres serve as the joint axes. As can be seen in the figure, the closed loop configuration displays aligned joint axes, corresponding to a zero closure error. Note that even though we started from a known closure configuration, a given linkage may have different branches of solutions, or the same closure can occur for different sets of joint variables⁵, both of which are reflected in the configurations displayed in Figure 17. Our tool has a module that captures the ranges of motion for these solutions, based on which solutions that fall into the same branch are identified and redundant solutions are discarded.

Constructing 1-DOF Linkages From Molecular Components One of the major applications of the developed technique is in the synthesis of novel molecular machinery. The critical aspect in constructing molecular linkages with constrained motion is the feasibility of loop formation. Fortunately, several approaches to loop formation are known, such as head-to-tail cyclization [25] and disulfide bond formation [26] in proteins. Many molecules, other than proteins, can have loops such as synthetic cyclic polymers, circular DNAs, cyclic polysaccharides and Rotaxanes [27]. Figure 18 depicts how the compiled link soups from the molecular components⁶ are combined with topologies to yield linkage arrangements. The open-loop configurations of linkages are then set up based on these arrangements and then passed to the optimization module to get the kinematic loops closed. Finally the range of motion is computed for each linkage. The resulting collection

⁵For example, a 4-bar mechanism maintains the closure during a range of values of the joint variables.

⁶We used Avogadro open source software to extract the link components [28]

can be found advantageous particularly in synthetic chemistry, as it narrows down the vast design space into a limited number of possibilities, with simulated behaviours for each candidate. Specifically, combining the unique capabilities of this package with the insight of the chemists/biochemists can lead to more efficient and more structured design processes in the smaller scales.

Macro Scale Mechanism Synthesis An example in which the proposed method can be found useful at the macro scale is when a limited number of standard parts must be used for constructing different mechanisms in order to limit the manufacturing costs. In such a case, one can input multiple candidate link soups to the enumeration tool to produce the optimum set of possible functional mechanisms constructed from the available links. A key strength of the proposed technique that is exploited here is that the search for mechanisms is conducted at different levels of topological complexity, starting from the simplest one. Thus, if a certain motion or function was not achievable with simple topologies, more complex ones are explored until a solution is found or we reach the limit set for complexity.

4 CONCLUSION

We have presented a novel method to synthesize machines from existing or manufacturable components, with important applications in the design of molecular machinery. The three-step process enumerates all valid topologies that satisfy prescribed mobility requirements, generates all possible linkage arrangements by assigning links to each topology from a set of available links (called the link soup), and performs a positional analysis for every linkage arrangement to produce all geometrically feasible solutions. Moreover, we determine and eliminate redundant topologies as these topologies are being generated in a computationally efficient manner. Importantly, this prompt elimination of redundant topologies significantly decreases the computational cost of the proposed design procedure. The preliminary tests suggest that the approach produces a complete set of non-isomorphic topologies and can be used in the practical design of nano, micro as well as macro functional mechanisms that have a prescribed number of degrees of freedom. The resulting synthesis procedure is the first procedure capable of synthesizing functional linkages with prescribed mobility constructed from a soup of primitive entities. At the same time, the proposed systematic procedure outputs an ATLAS of candidate mechanisms, which can be useful in the

synthesis of mechanisms for new application domains.

References

- [1] Tsai, L.-W., 2010. *Mechanism design: enumeration of kinematic structures according to function*. CRC press.
- [2] Smith, M. B., 2011. *Organic Synthesis, Third Edition*, Vol. 1. Academic Press, An Imprint of Elsevier.
- [3] Michael, R. G., and Johnson, D. S., 1979. “Computers and intractability: A guide to the theory of np-completeness”. *WH Freeman & Co., San Francisco*.
- [4] Mruthyunjaya, T., 2003. “Kinematic structure of mechanisms revisited”. *Mechanism and machine theory*, **38**(4), pp. 279–320.
- [5] Bunke, H., 2000. “Recent developments in graph matching”. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, Vol. 2, IEEE, pp. 117–124.
- [6] Foggia, P., Percannella, G., and Vento, M., 2014. “Graph matching and learning in pattern recognition in the last 10 years”. *International Journal of Pattern Recognition and Artificial Intelligence*, **28**(01).
- [7] Yan, H.-S., and Hwang, Y.-W., 1990. “Number synthesis of kinematic chains based on permutation groups”. *Mathematical and Computer Modelling*, **13**(8), pp. 29–42.
- [8] Magnasco, V., McWeeny, R., and Maksic, Z., 1991. Theoretical models of chemical bonding, part 4: Theoretical treatment of large molecules and their interactions.
- [9] Wampler, C. W., Hauenstein, J. D., and Sommese, A. J., 2011. “Mechanism mobility and a local dimension test”. *Mechanism and Machine Theory*, **46**(9), pp. 1193–1206.
- [10] Gogu, G., 2005. “Mobility of mechanisms: a critical review”. *Mechanism and Machine Theory*, **40**(9), pp. 1068–1097.
- [11] Shahbazi, Z., Ilies, H. T., and Kazerounian, K., 2010. “Hydrogen bonds and kinematic mobility of protein molecules”. *Journal of Mechanisms and Robotics*, **2**(2), p. 021009.

- [12] Sljoka, A., Shai, O., and Whiteley, W., 2011. “Checking mobility and decomposition of linkages via pebble game algorithm”. In ASME Design Engineering Technical Conferences.
- [13] Demirtas, A., and Shahbazi, Z., 2014. “An optimized kinematic mobility analysis of protein molecules”. In ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. V05AT08A014–V05AT08A014.
- [14] Kazeroonian, K., Latif, K., and Alvarado, C., 2005. “Protofold: A successive kinetostatic compliance method for protein conformation prediction”. *Journal of Mechanical Design*, **127**, p. 712.
- [15] Lotan, I., Schwarzer, F., Halperin, D., and Latombe, J.-C., 2004. “Algorithm and data structures for efficient energy maintenance during monte carlo simulation of proteins”. *Journal of Computational Biology*, **11**(5), pp. 902–932.
- [16] Canutescu, A. A., and Dunbrack, R. L., 2003. “Cyclic coordinate descent: A robotics algorithm for protein loop closure”. *Protein science*, **12**(5), pp. 963–972.
- [17] Meringer, M., 1999. “Fast generation of regular graphs and construction of cages”. *Journal of Graph Theory*, **30**(2), pp. 137–146.
- [18] Servatius, B., Shai, O., and Whiteley, W., 2010. “Combinatorial characterization of the assur graphs from engineering”. *European Journal of Combinatorics*, **31**(4), pp. 1091–1104.
- [19] Godsil, C. D., and Kocay, W., 1982. “Constructing graphs with pairs of pseudo-similar vertices”. *Journal of Combinatorial Theory, Series B*, **32**(2), pp. 146–155.
- [20] Curtin, B., 2005. “Algebraic characterizations of graph regularity conditions”. *Designs, Codes and Cryptography*, **34**(2-3), pp. 241–248.
- [21] Balakrishnan, R., and Ranganathan, K., 2012. *A textbook of graph theory*. Springer Science & Business Media.
- [22] Chin, K. W., Von Konsky, B. R., and Marriott, A., 1997. “Closed-form and generalized inverse kinematics solutions for the analysis of human motion”. In Engineering in Medicine and Biology Society, 1997.

- Proceedings of the 19th Annual International Conference of the IEEE,
Vol. 5, IEEE, pp. 1911–1914.
- [23] Welman, C., 1993. “Inverse kinematics and geometric constraints for articulated figure manipulation”. PhD thesis, Simon Fraser University.
 - [24] Olsen, A. L., and Petersen, H. G., 2011. “Inverse kinematics by numerical and analytical cyclic coordinate descent”. *Robotica*, **29**(04), pp. 619–626.
 - [25] Craik, D. J., 2006. “Seamless proteins tie up their loose ends”. *Science*, **311**(5767), pp. 1563–1564.
 - [26] Witt, D., 2008. “Recent developments in disulfide bond formation”. *Synthesis*(16), pp. 2491–2509.
 - [27] Semlyen, J. A., 1996. *Large ring molecules*. John Wiley & Son Ltd.
 - [28] Hanwell, M. D., Curtis, D. E., Lonie, D. C., Vandermeersch, T., Zurek, E., and Hutchison, G. R., 2012. “Avogadro: An advanced semantic chemical editor, visualization, and analysis platform.”. *J. Cheminformatics*, **4**(1), p. 17.

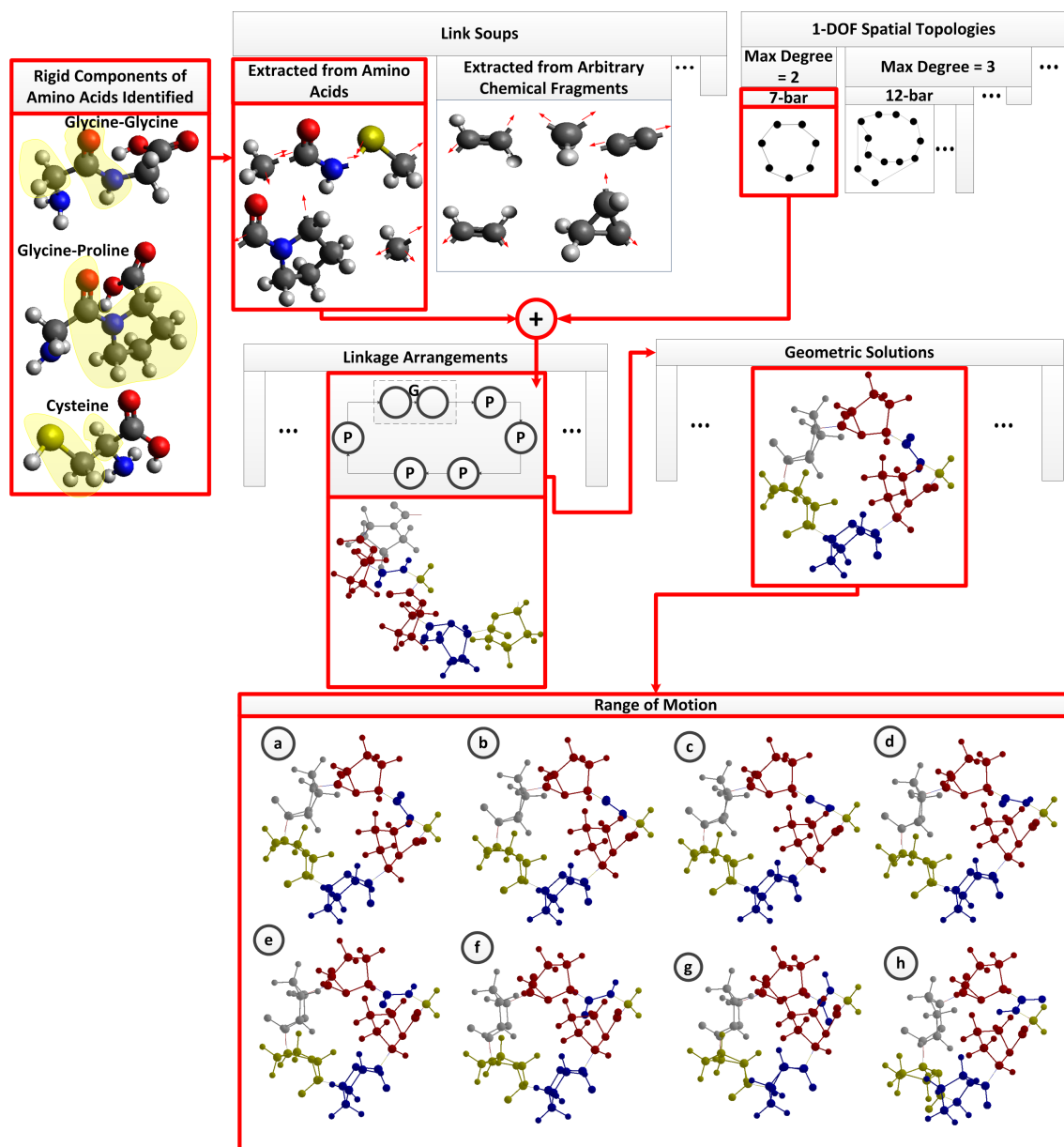


Figure 18: A 7-bar linkage is synthesized from rigid fragments of amino acids. Even though the linkage closes geometrically, the stability of the molecule is yet to be investigated.