AN ABSTRACT OF THE THESIS OF

Weifeng Huang for the degree of Master of Science in Mechanical Engineering presented on June 3, 2015.

Title: Automated Synthesis of Planar Mechanisms with Revolute, Prismatic and Pin-In-Slot Joints

Abstract approved: ____

Matthew I. Campbell

The intent of this research is to explore the entire design space of mechanism topologies by using a graph grammar synthesis approach. A new graph representation of planer mechanism has been developed to represent the planar mechanism with revolute (R), prismatic (P), and pin-in-slot (RP) joints. Following Gruebler's equation, the graph grammar rules are designed to increase the complexity of the linkage and avoid changing the default mobility. The "recognize" and "apply" process of graph grammar rules is done through the computer, so that the design space of mechanism topologies can be fully explored automatically.

The design space of four to fourteen bar R-joint 1-DOF topologies is obtained through this research. Each of the linkages in this space is valid and does not contain any rigid sub-structure as ensured by the additional graph grammar rules. The higher DOF topologies are also enumerated by degenerating the 1-DOF results. In order to increase the diversity of the topologies design space, P- and RP-joint substitution rules are used to replace the revolute joints in the topologies. With additional functions, the rotatability of the linkage can be preserved after P-joints are introduced. The research results in a total of 159,526 unique mechanism topologies that are each saved as

independent computer files. Additionally, a topology design exploration tool is created in this study to provide a convenient approach to generate complex 1-DOF linkage design.

Automated Synthesis of Planar Mechanisms with Revolute, Prismatic and Pin-In-Slot Joints

by

Weifeng Huang

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 3, 2015
Commencement June 2015

Master of Science thesis of Weifeng Huang presented on June 3, 2015.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial and Manufacturing Engineering

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Weifeng Huang, Author

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor Dr. Matthew I. Campbell. This thesis could not have be finished without your support and guidance, and I really appreciate all of the help that you have been providing to my research and study in this one and half years. I also would like to thank Dr. Ravi Balasubramanian for willing to be my committee member and review my thesis. I am also thankful to Dr. Yanming Di for participating my defense as my minor statistics major Professor. Meanwhile, I am grateful to Dr. Daniel Borello for being the GCR in my defense.

Additionally, my lab mates in Design Engineering Lab also give me a lot of valuable suggestions and reviews for improving my works. I am really thankful for their helps.

Finally, I cannot go this far without the support and love from my family and my wife. Thank you so much.

TABLE OF CONTENTS

LIST OF FIGURES

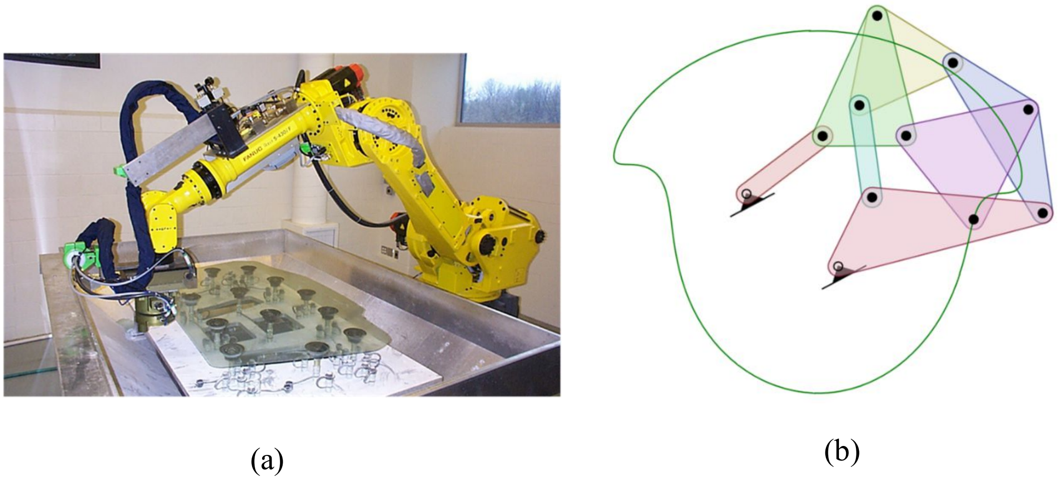
LIST OF FIGURES (Continued)

LIST OF TABLES

1 Introduction

It is well known that planar mechanisms like four-bars and cranks-and-sliders have been used for centuries to accomplish different tasks. While many planar mechanisms are common knowledge to mechanical engineers, the mechanisms can be highly complex and innovative. Planar mechanisms – whether they are simple or complicated – are all part of the same language follows the same restrictions. Since Gruebler's equation governing the mobility of planar mechanisms is straightforward to solve (with simply the number of links and joints), it would appear possible that the entire valid space of one degree-of-freedom (DOF) mechanisms can be captured explicitly. In this space, each of the linkages is controlled by a single input. Meanwhile, linkages that contain rigid structures or static links are considered as violations and removed from this space. Various other authors[1][2][3][4] have attempted such endeavors, but the arguments about the size of valid linkage designs space still exist. More than that, the topologies of these complex designs are not available. Due to this reason, it is difficult to obtain more special kinematic properties of the complicated linkage designs. So in this research, we intent to represent all planar mechanism topologies as a library of graphs that each of them could be simulated as a real linkage design. As the number of links and joints in a linkage increase, the complexity of the kinematic properties also increases – beyond what is produced by common 4 and 6-bar linkages. This is important to help engineers explore new possibilities for accomplishing complex tasks.

In recent decades, the focus on robotic systems has led many to pursue multi-actuator systems for complex movements as alternative design instead of the simpler one DOF mechanisms. Although it is not substantiated, such single degree-of-freedom mechanisms may prove to be lighter, more robust, and more efficient than high DOF robotic "open chains." Also, they may provide alternative solutions lead to less cost and engineering. Two path synthesis tasks that are accomplished by a robot arm and an eight-bar linkage are shown in Figure 1. The task for the robot arm is to polish the edge of a flat glass in Figure 1a. This process could be simplified as the path synthesis design, and 4 servo motors are used to control this robot arm and ensure the grinding head can keep following the edge. In figure 1b, a similar path is generated by an output joint in an eight-bar linkage with a constant input link. Although the path synthesis and kinematic property are not what we focus on in this study, this example does give us an idea that the utilization of linkage designs could be expanded by increasing their complexity. Obviously, the path in Figure 1b is not a traditional path, which cannot be synthesized by 4- or 6-bar linkages. So this requires us to increase the structural complexity of the linkages for accomplishing more difficult tasks. Regardless, exploring the space of planar mechanisms has always been challenging. With a repository of mechanisms, engineering designers could more easily to explore the space of possibilities. Furthermore, it is possible that computational search could optimize the dimensions of one or more of the many valid topologies to best meet the desired kinematic behavior.



(a) (b)

Figure 1: A robot arm and an 8 bar linkage are used in two path synthesis tasks. [5][6]

In this research, a formal and implemented approach is developed to capture the valid space of planar mechanisms. This approach not only synthesizes revolute (R) joint linkages as shown by previous authors; it additionally includes prismatic (P) and pin-in-slot (RP) joints. This is accomplished by a set of graph transformations – referred to as grammar rules – that construct graphs in the systematic method presented by Tsai [7]. The rules also transform simpler dyadic mechanisms into more complex linkages, like the double butterfly mechanism, that lack any dyadic chains. By using these rules, the space of 1 degree-of-freedom (DOF) planar mechanism can be fully explored. After that, linkages with higher DOF can also be obtained from this space by degeneration process.

Graph grammar rules are used as the design criteria to achieve automated exploration of the topologies. The function of graph grammar rule is to transform graphs by addition, deletion and relocation of the nodes and arcs. Refer to graph representation of linkage, graph grammar rules can create new linkages by modifying the structure of the existing topologies. All of these graph grammar rules are design base on Gruebler's equation:

$$3(L-1) - 2J_1 - J_2 = M \tag{1}$$

In this equation, L represents the number of links or rigid bodies, J_1 represents the number of 1-DOF joints (such as R and P), and J_2 represents the 2-DOF joints (such as RP and gear mate). The result M, is also known as the overall mechanism degrees-of-freedom or as the mechanism mobility. According to this equation, graph grammar rules can be designed to vary the linkage topology without changing the mobility, and the design process of them will be discussed in the later section.

We are also interested in synthesizing complex linkages with prismatic joints in this research because this is a challenging problem and there are little studies attempt to enumerate the more complicated P-joint topology designs. When P-joint is involved to a linkage design, it can limit the rotatability of the links, which may lead to an invalid design. Hence, when introducing the P-joints to the topology design, we cannot treat it as a simple graph synthesis problem. Instead, the kinematic property of each of the P-joint topologies needs to be concerned after it is created. Meanwhile, a method is developed to ensure the P-joint will not affect the validity of the linkage. But the process of modeling and simulation for each of the individual complex P-joint designs is very time-consuming. Due to this reason, an efficient approach is necessary for examining each of the P-joint results. In this research, Planar Mechanism Kinematic Simulator (PMKS) is used for producing the simulations. This simulator is developed by Campbell [8], and it can simulate the behavior of planar mechanisms with its position, velocity and acceleration. More importantly, all of the topology results can be opened and simulated directly in PMKS. So the examination process for the P-joint linkage designs is simplified significantly. Meanwhile, three constraints are used in this research to prevent violations after placing P-joint to a linkage design. So the exploration of the complex P-joint topology space is become achievable. After P-joint is placed, the design space of RP-joint linkages can be easily discovered by a RP-joint substitution graph grammar rule.

With all the graph grammar rules and their applications, the design space of mechanism topology can be populated. Also, a procedure is designed to eliminate any isomorphic results and capture the rigid structures. The rigid structures are valuable that they can be used to detecting the rigid sub-part within a movable linkage. So obtaining all these truss structures can ensure the validity of the results. After that, a breadth-first tree search method is used for searching the topologies design. This search starts with a single pendulum mechanism since it is the simplest 1-

DOF linkage. The invoked graph grammar rules will be applied to it and create new topologies. Each of these topologies is set as a host graph, and more new topologies can be synthesized by these graph grammar rules and sorted in the next level of the tree. Then the valid design space of topology is fully discovered.

Due to high difficulty for constructing a complex linkage model, a design exploration tool is also developed in this thesis to provide a quick and simple approach for synthesizing complex linkage with P- and RP-joints designs. Based on the established rules, a planar mechanism design problem can be transferred to a graph creation problem. With the specification of desired number of link and types of joint, this tool can create a topology by using graph grammars and satisfies these requirements. After that, the desired topology will be simulated by PMKS, and the kinematic behavior of this linkage cam be observed.

2 Related Work

There are many researchers who have been working on exploring the design space of planar mechanisms, and the results for 6, 8 and 10-bar mechanisms with only revolute joints are well established [1][2][7][9]. These publications present the total numbers of the possible valid topologies designs in each levels. However, when the topologies have higher links or higher DOF, contradictions of the size of these space exist[2][3].

When the linkages synthesis involves P-joints, there are additional challenge to complete the exploration of the space. Sardain provides a method to avoid the rotatability being affected by P-joints [10]. In his research, a linkage is divided into two different kinds of components: primary components and secondary components. The primary components are those components that can be calculated from the dimensions at the beginning, and the synthesis of secondary components, which depend on the results from primary components. If a link with a P-joint needs to be added to the primary components, the link must be a binary link. Also, Sardain sets the limit that it can only have one P-joint on each link. Based on these rules, 43 topologies for 4 and 6-bars that contain P-joints are found. In this thesis, it is shown that the space with P-joint is significantly bigger.

In order to achieve this exploration, graph theory and graph grammar rules are used in this study, and the linkage design process is seen as the process of graph synthesis. Some engineering design problem could be viewed as graph transformation problems, and graph grammar rules can represent component, function or structure in real design. Hence, the design will be closer to the completion by applying these grammar rules to the graph[11]. This graph synthesis approach have been used for solving problems of component selection[12], sheet metal design[13], and synthesis of gear trains [14]. For this study, once the graph grammar rules are defined, the computer can use these rule to explore the design space and synthesis new linkage for a given engineering design problem.

The idea of the transformation from graphs to the real linkages is based on Tsai's graph representation of linkages[7]. In his research, complex mechanisms are represented by very simple graphs that only contain four elements: nodes, joints, input label and letter of joint type. Meanwhile, some detailed information such as dimension of the linkages or locations of each joint are not included in his study. The methodology of using graphs to present the design of linkage mechanism is very valuable since this study is related to graph-based searching process, and it provides a connection between graphs and real mechanisms. Also, the design of the graph grammar rules will become easier since less information need to be concerned. Once this graph representation of linkages is established, graph grammar rules could be applied and modifies the graph to explore new linkage.

Graph synthesis is a tedious process to do by hand because it involves graph and graph recognition, which requires all of the valid locations in the graph that the graph grammar rule can be applied can be discovered. Depend on the size of the graph and the complexity of the grammar rule, the recognition process could be extremely difficult to complete by human hand. Hence, the software Graphsynth that have been developed by Campbell [15] is used in this research. When all of the rules that are defined, the recognition and apply function will automatically execute to generate new graphs.

3 Graph Representation

Graph theory is applied for representing and exploring linkage topologies in this study. Graphs can be seen as combinations of node and arcs, and they can be used to model the relationships between different objects in a system. By using graphs, linkage mechanisms could be translated into a graphical language that could be recognized by the computer for automated synthesis.

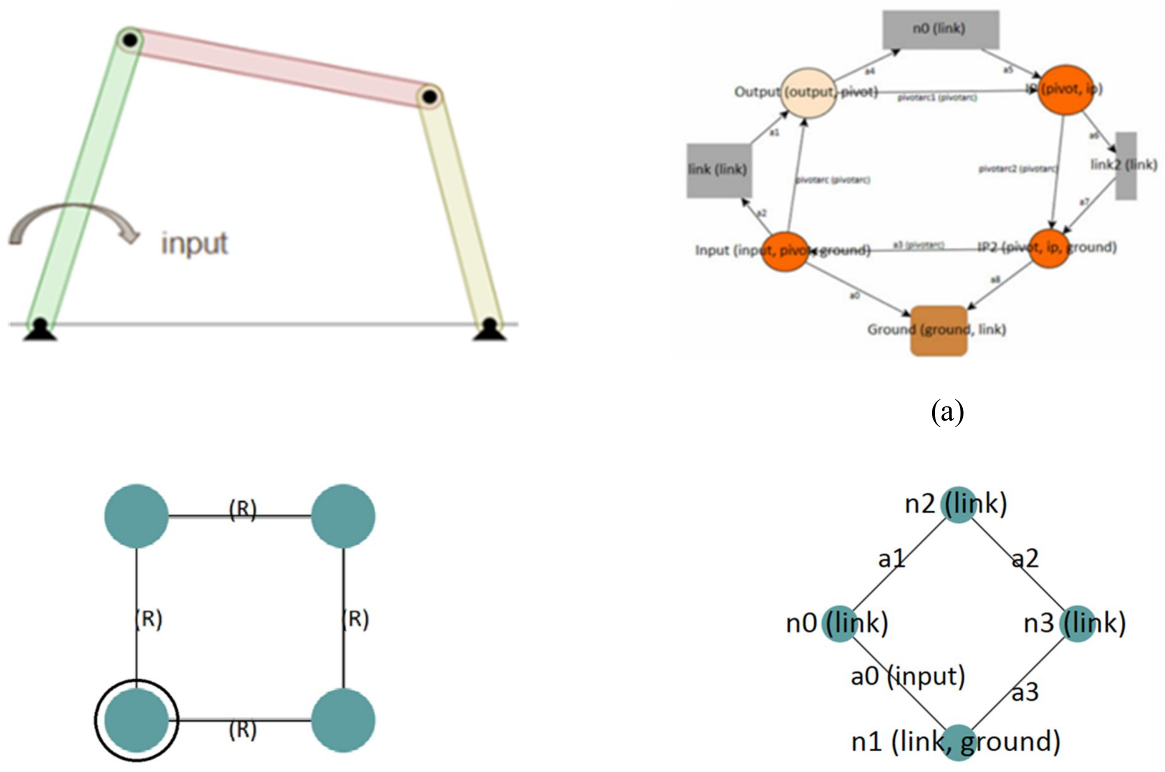
Designing the graph representation schema of linkages is the first task in this study. The way in which the linkage is modeled will directly affect the construction of the rules, the computing speed of the search, and the conversion process from topologies to kinematic simulations. Hence, the representation is carefully defined in the first step of this study and all of the other efforts is based on this representation system.

After the graph representation of linkage is established, graph grammar rules can be introduced to create new topology. The general function of grammar rules is to change the combination of a graph so a new graph can be generated. This process is completed by adding new nodes or arcs to a graph. Also, it could be done by deleting or relocating nodes or arcs in an existing graph. These grammar rules only apply to the location that they recognized and modify the graph based on the how they are designed. Hence, automatic synthesis of linkage is achievable once these design graph grammar rules are built.

3.1 Selecting Graph Representation

Two different graph representations are discussed in this thesis. In Radhakrishnan’s study [8], the graph representation includes many details that relate to a real linkage design. His representation approach for a 4-bar linkage is illustrated in Figure 2a. The local labels indicate the function of nodes and arcs. For example, the local label “link” represents a link or rigid body and “pivot” represents a revolute or R joint which connects two links. While concrete objects are represented by nodes, the arcs are used only to represent the flow of energy.

The other graph representation investigated is by Tsai [2]. Here in Figure 2b, nodes indicate links in the linkage, and arcs indicate the joints. Local labels only appear in arcs to show the type of joints. The ground link is not indicated. Instead, the input link is known as the node that surrounded by a circle. Since this graph representation is used for structural analysis, physical dimensions of the linkages are not considered.



(b) (c)

Figure 2: Three different graph representations for a single input 4-bar linkage.

Radhakrishnan's approach is more descriptive than Tsai's because it includes additional elements needed to define a mechanism. By representing the joints as nodes, two aspects of the mechanism can be more clearly defined. First, since nodes are typically attributed to Cartesian coordinates in representing graphs, the joints can be explicitly positioned by the corresponding joint node. Second, the limitation of the arc-as-joint in Tsai's representation prevents representing three or more links joined by the same joint. Radhakrishnan's can explicitly represent this as its own topology.

In this study, the graph representation is based on Tsai's work, with some modifications towards Radhakrishnan's approach, and it is presented in Figure 2c. For this representation, local labels are used to describe joints as is done in Tsai's but are also used to indicate the ground link. The default joint type is revolute, so instead of including an "R" label, it is simply left blank. Additional labels for "P" and "RP" are added to represent prismatic sliding joints and pin-in-slots half-joints respectively. Instead of setting the driver link as the input of the linkage, we consider the joint that drives this prime link as the prime mover of the linkage. Because in real designs, the input links of linkages are usually connected and driven by rotational or linear motors. Hence, the change we make can distinguish what type of input joints is used in the linkage, and it is defined by adding the joint type to the input arc.

Tsai never explicitly captures the RP joint but rather sees this as a special case when an R and a P occupy the same location. Our approach explicitly captures half-joints as unique topologies since the goal is to synthesize realizable mechanisms from the graph representation. Tsai developed his approach more as a method of analysis and categorization, so the avoidance of RP is justified. Interestingly, Tsai does introduce the gear, or "G" half joint, so his method does not completely avoid the complications caused by these interactions. When a linkage contains RP joints, there is an added complexity due to a lack of symmetry in the joint. One of the links serves as the slide or P component of the RP-joint while the other link serves as the R, revolute, or pin in the slot. This is worth distinguishing given the significant effect on the kinematics of the linkage. For example in Figure 3, the two three-bar linkages that contain RP-joints have the same topologies and dimensions, except how the two slots are located. The slot is located at the ternary link in Figure 3a. Meanwhile, the other slot is set on ground in Figure 3b. The two paths indicate the movements of tracer points on the ternary plates, and it is clear that the paths are different on the both mechanisms. For this reason, the graph representation of the RP-joint should be able to capture the location of the slot and the direction of the arc is used in the RP-joint arc to indicate its design. In Figure 3a, the RP-joint is represented as an arc going from ground to the ternary link. With the tail of the arc corresponding to R and the head as the P, the R is on the ground and the

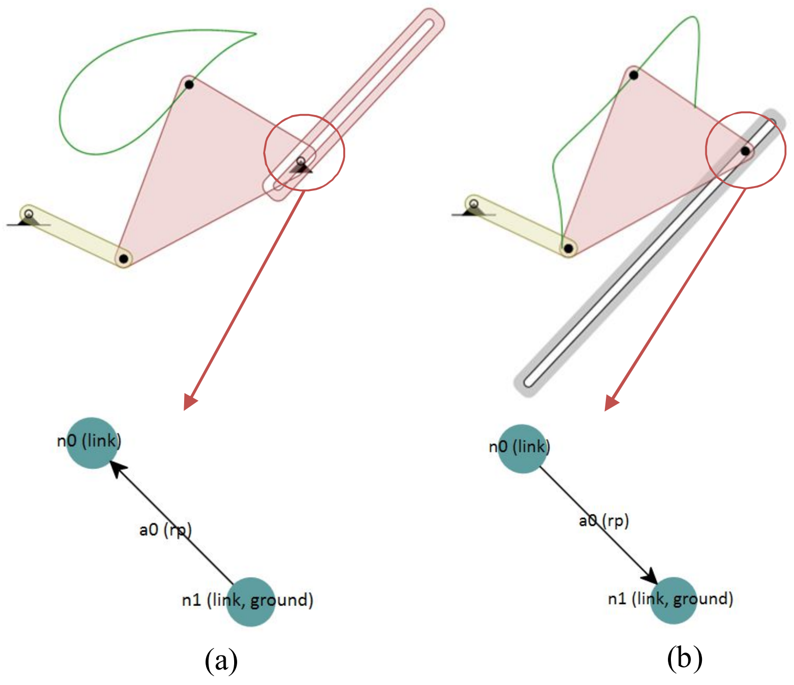


Figure 3: Two RP-joint arcs with different direction of arrow represent the order of the pin and slot in the real linkages[16][17].

slide is part of the ternary plate. In Figure 3b, the arc direction is switched. In our augmentation of Tsai's representation, arc direction will be used to indicate different mechanisms topologies.

3.2 Design of Graph Grammar Gules

After establishing the graph representation, graph grammar rules can be developed to generate new topologies. This methodology is described in Campbell's study [11], in which graph grammar rules serve as a production rule system to create a tree of graph topologies.

A grammar rule is constructed of a Left Hand Side (LHS) and a Right Hand side (RHS). The functions these two parts are to identify which part in the host graph can be changed and how the change is produced. The LHS of the rule is used for the recognition process. In this process, the computer tries to recognize the all of the locations that the rule can be applied to. If it succeed, this rule is set as an option and we can use it to change the graph. Hence, we can design what part of the linkage can be made change to. After that, the apply process is to modify a host graph based on the components of RHS. The RHS of the rule indicates how to add or remove nodes, arcs and/or labels from the recognized part in the graph. Respectively, the process of applying graph grammar is to add components or make modifications to an existing linkage resulting in a new valid linkage rules in this study.

Figure 4 shows an example of a graph grammar and a 4-bar linkage is synthesized from a pendulum by using this rule. This process starts from a host graph that has two nodes *n0* and *n1* and an arc *a0*. The local label *ground* in *n1* means this link is connected to the ground and it is driven by an input pin joint which is specified by the label "*input"* in *a0*. This link and ground need to be connected by a dyad to form a 4-bar linkage. So the design of the LHS for the rule is constructed by two nodes *n0* and *n6*, which means this rule will search any pair of nodes in this

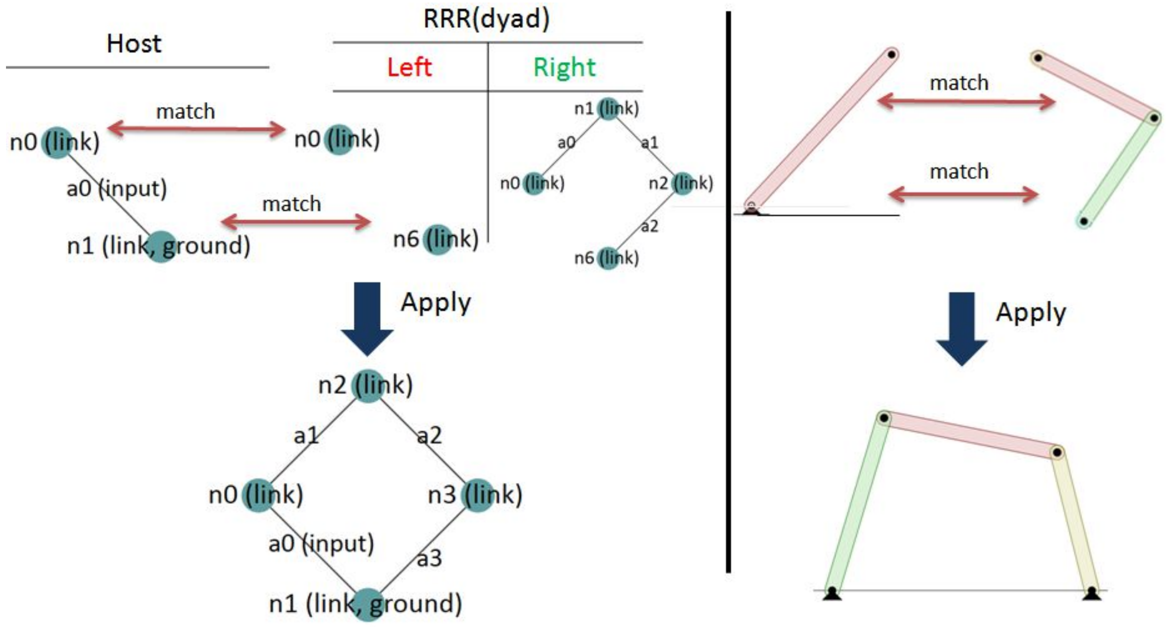


Figure 4: Applying a dyad rule to generate a 4-bar linkage with its representation in real linkage design.

host graph. In this case, two nodes *n0* and *n1* matched the LHS and becomes a valid location for applying this dyad rule. This matching process is referred to graph recognition. After that, the host graph is modified according to how the right hand side (RHS) of the rule differs from the left. In this example, the RHS of the rule retains nodes *n0* and *n6* from LHS with new additions *a0, n1, a1, n2* and *a2*. These additions are referred as a dyad component which has 2 links and 3 joints, and it is added to recognized location *n0-n1*. This result in a four-bar topology. In this study, different grammar rules are used to encapsulate the constraints of planar mechanisms and define how more complex mechanism graphs can be created from simpler ones.

Based on the design of the LSH, a rule could be applied to the same location repeatedly. So avoiding these redundant applications in the searching process is important in this study. Take the example in Figure 4 again, the valid location could be recognized twice with two different sequences which are n0-n1 and n1-n0, and the computer will return this two same location as options that the rule can be applied to. In this study, the dyad that is represented in the RHS of the rule is a symmetrical component. So applying this dyad rule to these two options leads to generating two same four-bar linkages. To solve this problem, options need to be checked to see if any have the same collections of elements. If the collections are the same, only one option is retained and the others are considered as duplicate options and are removed. When a graph tree search is involved, the same results will develop indistinguishable branches in the tree, which may cost time and memory. So these repeated applications of rules should be avoided. Also, another method is used to check and remove isomorphic graphs during the synthesis process. The details of is will be discussed in later section.

Related to how a four-bar linkage is generated by the dyad rule, the full design space of mechanism topologies can be explored and all of the rigid structures are able to be identified with an appropriate collection of grammar rules. The fundamental criterion of designing these rules is based on the Gruebler's equation. The DOF of a linkage can be easily calculated by using this equation. This equation ensures the graph grammar rules will not change the DOF of a new linkage. The rules in this research are divided into four categories based on their functions in synthesis process: Generation, Transformation, Detection and Joint Replacement. The function of Generation rules is to add components to increase the number of links and joints in the linkages. After that, Transformation rule is applied to relocate the ground joint to form new topologies. The Detection rules recognize and extract the rigid structures in the graph. Finally, the Joint Replacement rule will recognize R-joints and replace them by P-joints when the position is accessible.

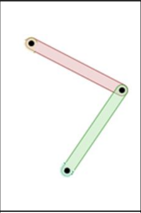
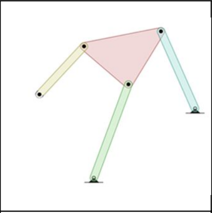
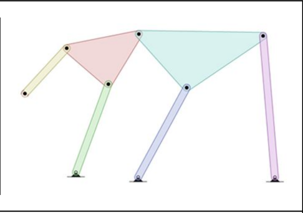
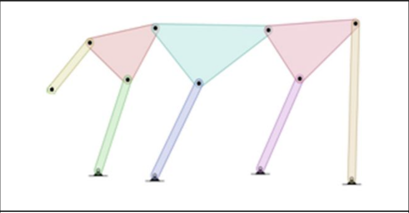
3.2.1 Generation Rules

One way to explore a more complicated linkage is to add more links and joints to a linkage and a new linkage is created. For example, the generation of some famous 6-bar linkages like Watt I Watt II or Stephenson I linkages can be created by adding a dyad between two different links of a 4-bar linkage. Hence, if more components which are like the dyad that can be added to the linkages are discovered, the complexity of the linkages can be increased by setting them as the additions to the linkages.

The function of Generation rules is to recognize the applicable locations at a linkage and to add the designed components to this location without changing the mobility of this linkage. Keeping 1-DOF for every linkage in this research is important. If linkages with different DOF are involved in the search process, designs of graph grammar rules will become more complicated. That's is because the change of the mobility for topology is depend on how many nodes and arcs are added. For each kind of DOF changes, we need to find out the all combinations of nodes and arcs to meet that change. So designing more components for the synthesis of multiple DOF linkage is not preferred in this study. In fact, the design space of higher DOF linkage could be discovered through degenerating the 1-DOF linkage space. So the first step to create these rules is to define what kind of components can be applied to the linkage and reserve its original DOF. The construction of these components is based on modified Gruebler's equation with new variables as below:

$$3(L-1+L_{rule}) - 2(J_1 + J_{1rule}) - (J_2 + J_{2rule}) = M \tag{2}$$

Table 1: Components for design the Generation rules.

				
	Dyad	Triad	Double-Triad	Triple-Triad
L_{rule}	2	4	6	8
J_{1rule}	3	6	9	12

In this equation, variables L_{rule}, J_{1rule} and J_{2rule} indicate the number of links, 1-DOF joints and 2-DOF joints that are added by a rule. When the mobility M is set to 1, combinations of these 3 variables can be found to balance this equation. Since RP-joints are addressed later, J_2 and J_{2rule} are set to 0. We focus on possible L_{rule} and J_{1rule} combinations to provide insight into the valid permutations of Generation rules. Some example of these permutations is depicted in Table 1.

After combinations are found, components with these numbers of joints and links can be designed. In order to reduce the amount of isomorphism during the synthesis process, only one

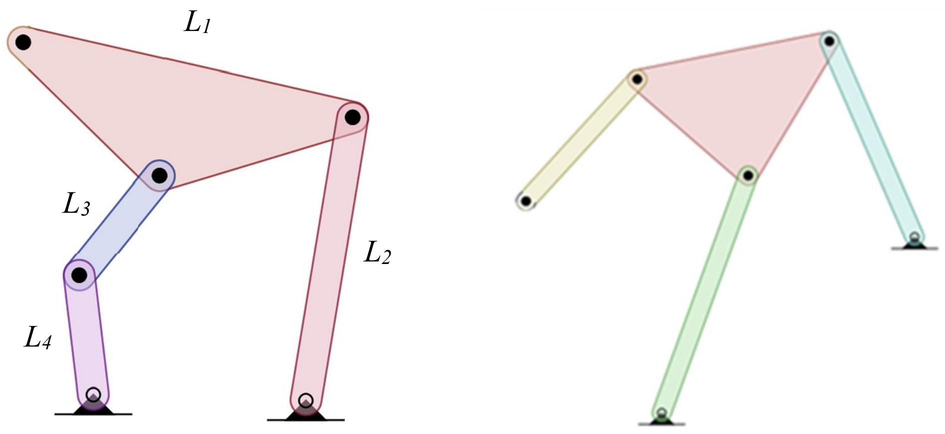


Figure 5: An example of applying a dyad rule to generate a 4-bar linkage with its representation in real linkage design.

component is used to represent each of these combinations. For the combination L_{rule}=2, J_{1rule} =3, only the dyad can satisfy it. But for the combination L_{rule} = 4, J_{1rule} = 6, there are more than one component can meet this requirement. In Figure 5, both of these components have 4 links and 6 joints. But the component on the left is constructed by attaching a dyad component L_3-L_4 to the other dyad component L_1-L_2. Since adding this component to a linkage is equivalent to adding two dyads in sequence, it is not considered as an appropriate component that can be used to make a unique topology. In other words, a component that is synthesized from dyad additions should not be an option for making additional Generation rules. In contrast, the component on the right that cannot be synthesis by dyad structure is chosen for the combination L_{rule} = 4, J_{1rule} = 6. Some examples are shown in Table. 1. These components are presented by graphs and are set in the RHS of the graph grammar rules as the additions that can apply to a graph.

Once the components are selected, we need to define the LHS of the rule. The function of which is to identify the locations in the linkage so that components could be added. For the dyad rule, it is recognized on any pair of links as part of the left-hand. For the more complex triadic rules, only one of the links and the ground link is recognized in the mechanism. The reason for this setup is to increase the number of ground joints after the application of triadic rules, and it can provide more recognizable locations for the Transformation rule to apply so more topologies can be synthesized. The application of Transformation rule is discussed in next section. An example for applying a single triad rule to a pendulum and create a Stephenson six bar inversion III linkage is shown in Figure 6. Compared to the LHS of the dyad rule in Figure 4, the node *n6* in the LHS of the triad rule has an extra label name "*ground*", which means one of the nodes in the LHS of the triad rule can only recognize the ground link in the host linkage. In this case, two joints in the triad attach to the ground with the others attached to the pendulum, and the new linkage is created.

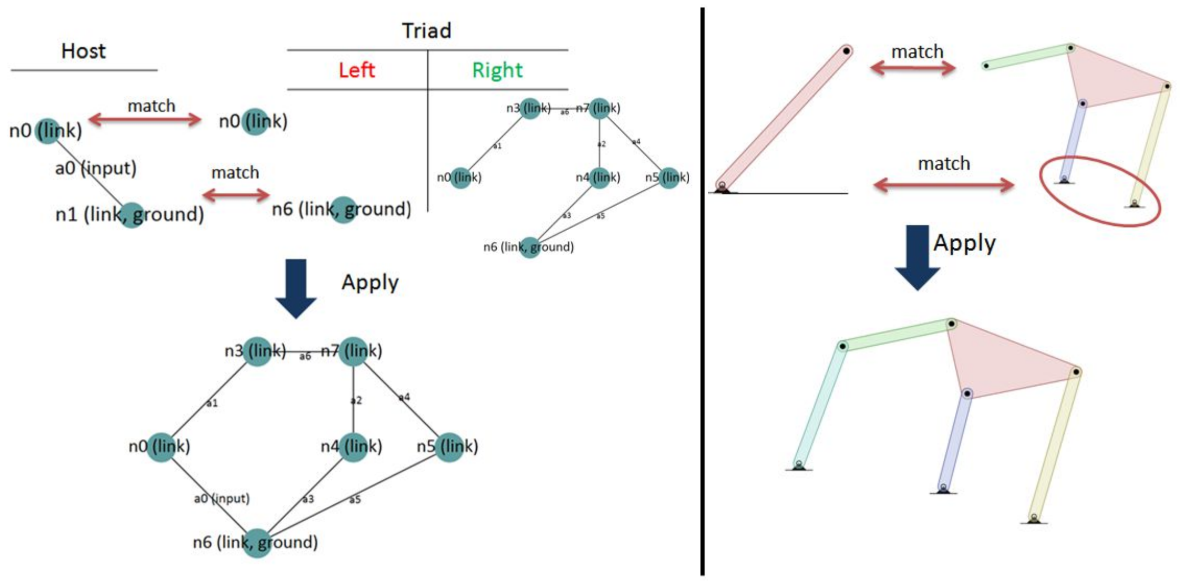


Figure 6: Applying a triad rule to generate a Stephenson six bar inversion III linkage with its representation in real linkage design.

For multiple triadic rules, the number of the ternary plates in the rule is needed to be increased for generating more complicated topologies. For example, the additions in Table. 1 are used for exploring linkages with 10 links due to one of these linkages can be made by directly adding the triple triad component to the input pendulum. When doing a 12-bar linkage search, a new component is needed which contains one more ternary plate and ground joint then Triple-Triad rule. Its creating process is shown in Figure7. A new component that has a ternary plate with

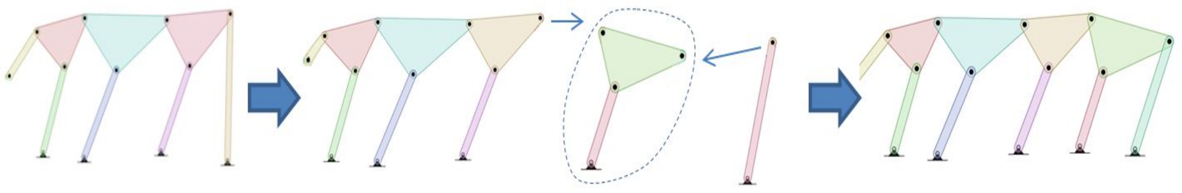


Figure 7: The process of generating a 4-triad component.

a binary link is inserted to the Triple-Triad component. This adding ternary plate process is repeated when higher level linkage generation is required.

Most of the topologies are created by applying these Generation rules and they simplify the generation constraints by adding components that are designed based on Gruebler's equation. The mobility of the new linkage will not be changed when applying Generation rules, but they limit the possible configurations that are created. To solve this, the next Transformation rule explores more linkage mechanisms to fully cover the space of possibilities.

3.2.2 Transformation Rule: Move Joint from Ground to Link

A new graph could be created not only by adding more components to the original graph, changing the combination of the nodes and the arc in the graph can also lead to a new configuration. For this reason, more results can be covered by using a Transformation rule while the Generation rules do not create all the possible topologies. When triadic rules are applied to the host, the number of ground joints in the linkage increases, and relocating these ground joints can result in finding new topologies. So the function of Transformation rule is to detach one of the non-input ground joints and connect it to a different link to form new mechanism. The Transformation rule is only valid when the number of ground joints is higher than two. Otherwise, an invalid linkage with a single ground joint could exist.

An application of this rule for generating a double butterfly linkage as shown in Figure 8. The host topology is 8-bar linkage that generated by adding a double triad to a pendulum *n0* and the ground. A collection *n5-a6-n6-a9-n1* and a node *n0* (the arc *a0* is not included) in the host graph are recognized by the component *n0-a1-n1-a0-n2* and *n3* in the LHS of the rule respectively. After this recognition process, this rule is applied to the host graph by removing the arc *a9* between

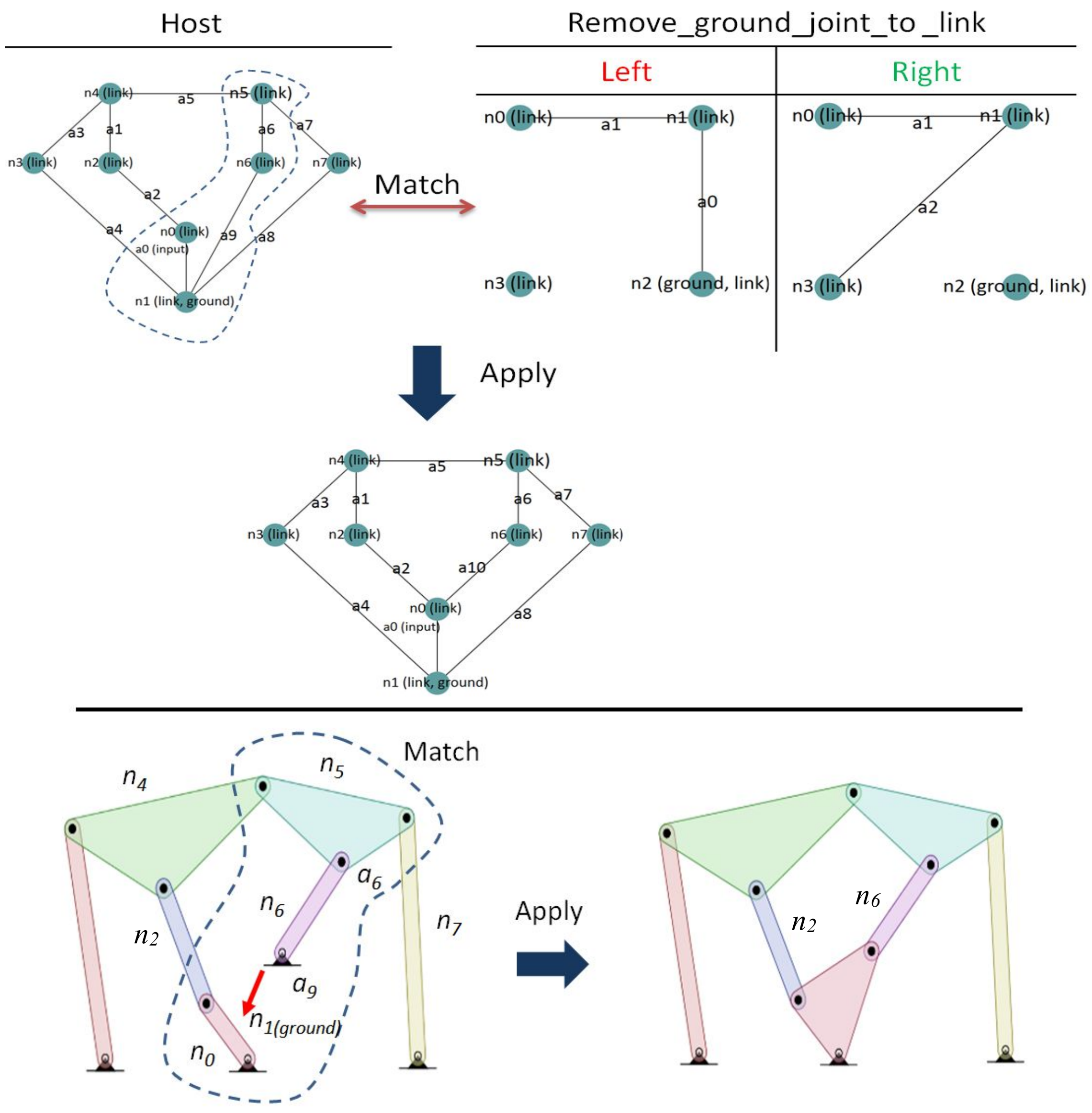


Figure 8: Applying a Transformation rule to generate a double butterfly linkage with its representation in real linkage design.

n6 and *n1* and adding a new arc *a10* between *n0* and *n6*, and then a new double butterfly linkage is produced. The node *n0* and arc *a1* that on LHS of this rule can avoid building hyper arcs (two or more arcs between two nodes) in the graph. Also, by using additional function during the recognition process, this rule can simply keep away from making rigid triangle structure.

One advantage of using Transformation rule is it can reduce the number of triadic rules significantly by changing the internal connection of the graph. When only using one triadic rule, the diversity of the topologies could be increase by connecting its binary links to different locations of a graph. The drawback of this method is it required more rules to capture all the combination of applicable locations so those binary links could be assigned differently. Instead, the Transformation rule can achieve the same diversity by changing the existing connect between the host graph and the new addition. For example in Figure 8, the double butterfly linkage could be generated by using only one double triad Generation rule while two of the binary link n2 and n6 are connected to the input pendulum n1 directly without using Transformation rule. But when we need to apply this double triad rule to other locations, more graph grammar rules need to be designed so those locations with different combinations of nodes and arcs could be recognized. The combinations of these designs could be a lot, especially when the component becomes more complicated, it is impossible to design all of these rules for each of the application. For this reason, the design of all of the triadic rules is to only connect one binary link to the host linkage, and the rest of them are connected to the ground. By using the single Transformation rule, relocating these ground joints can generate more new graphs without using extra graph grammar rules, and this is the reason why we maximize the number of the ground joints in the triadic rule design.

By using the transformation and Generation rules, the design space of R-joint mechanism topologies is able to be fully explored. In this space, single DOF for each of the topologies is guaranteed, but the rigid structure problem exist and it affects the rotatability of links in the linkage. So this problem needs to be resolved, and the invalid results should be excluded from the design space.

3.2.3 Taboo Detection Rules

The designs of generation and Transformation rules are highly based on Gruebler's equation. Since the Gruebler's equation only focus on the mobility of the whole linkage design, the relative mobility between each of the links in a linkage is neglected. So a rigid structure could appear after applying Transformation rule to a linkage. For example in Figure 9, the ground joint J_1 in the 8-bar linkage is relocated to the link L_1 after applying Transformation rule. This result in a new 8-bar linkage contains a 5-bar rigid structure. In this 5-bar rigid structure, all of the links are relatively static to each other. So this rigid structure moves as a whole plate and is equivalent to a coupler link in 4-bar linkage. Although this 8-bar linkage still remains 1-DOF, it is considered as a violation.

Rigid structures that can affect the partial mobility of the linkage should be captured. Correspondingly, topologies that contain these structures should be removed from the design space.

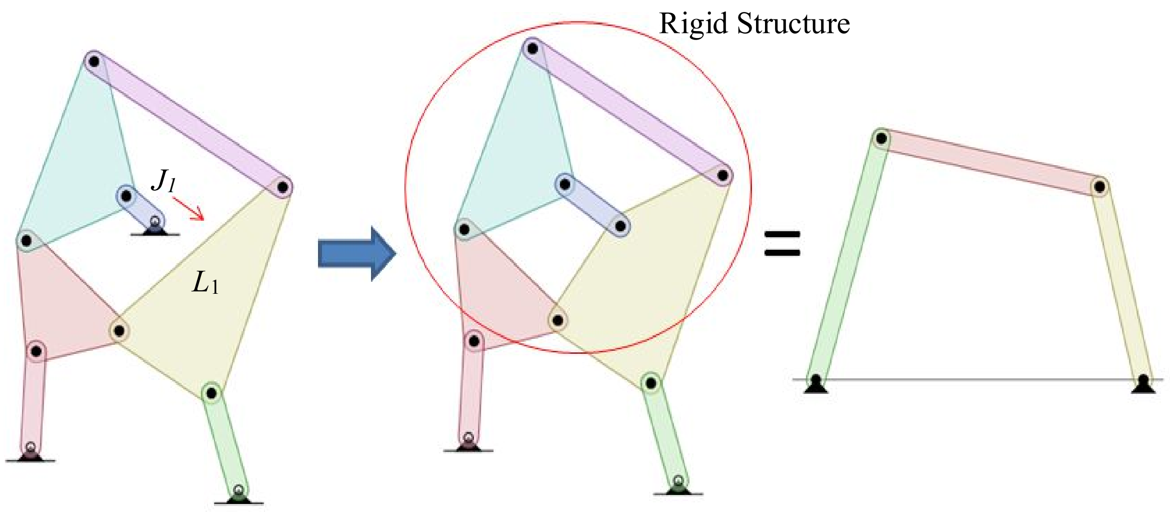


Figure 9: An 8 bar linkage contain a 5-bar rigid structure equivalents to a 4-bar linkage [18].

So this requires a universal method to detect and extract the rigid structure from any valid linkage. One of the approaches is to set up additional functions avoid creating simple rigid structures during rule application such as the additional function that in the Transformation rule can keep away from making a 3-bar triangle rigid structure. But when the number of links increases, the number of rigid structures grows exponentially. So finding all of the rigid structures and including them into the additional function is a tedious process.

When rigid structures appear in some linkages, a pattern can be found, and we can use this pattern to design a new graph grammar rule to capture these structures and exclude those valid linkages from the searching process. On the other hand, for the systematic enumeration of mechanisms, the structural characteristics of rigid structures are valuable to be sorted in order to ensure that future results do not suffer the same problems. When a rigid mechanism is found, it gives birth to a Taboo detection rule, which is automatically generated and check with subsequent mechanism graphs.

The pattern we discover in this thesis is described as below: For any 1-DOF R-joint linkages that have more than 4 links, a rigid structure could exist and it could function as a coupler link in a 4-bar linkage that connects to the input and output link. An example of this is shown in Figure 9 before. Due to this pattern, a rule can be used to obtain the rigid structure by identifying and removing a 3-binary-link chain that is constructed by input, ground and output links from the linkage to obtain a rigid structure. The result for this method is promised based on Gruebler's equation (1). For an N bar 1-DOF linkage with a rigid structure, when it contains a rigid structure that can match the description before, there must be additional structure to drive this rigid component and maintains the mobility back to 1-DOF. By introducing new variables to Gruebler's equation (1), the number of links and joints in the driver structure can be found and thus removed to find the rigid structure. The modified equation is as below:

$$3(L-1-L_{remove}) - 2(J_1-J_{remove}) = M \tag{3}$$

The new variables L_{remove} and J_{remove} are the numbers of the links and joints that can be removed. In order to reduce the DOF from 1 to 0, the combination of L_{remove} = 3 and J_{remove} =4 is chosen because it can reduce 1-DOF from the linkage. When L_{remove} = 3 and J_{remove} =4, the mobility in this equation is become *M-1*. A 3-binary-links chain with 4 joints is chosen to represent this combination. After removing this chain, the mobility of the remaining structure becomes 0 based on the equation (3), and a rigid structure is acquired.

A rule names *Remove_Three_Bar* in Figure 10 is constructed by the component we selected. The LHS of this rule is a chain that has 3 nodes and 4 arcs, and the RHS is empty, which means by applying this rule, a chain structure with 3 links and 4 joints will be removed from the linkage, and a rigid structure with *L*-3 number of links can be discovered during L bar linkage

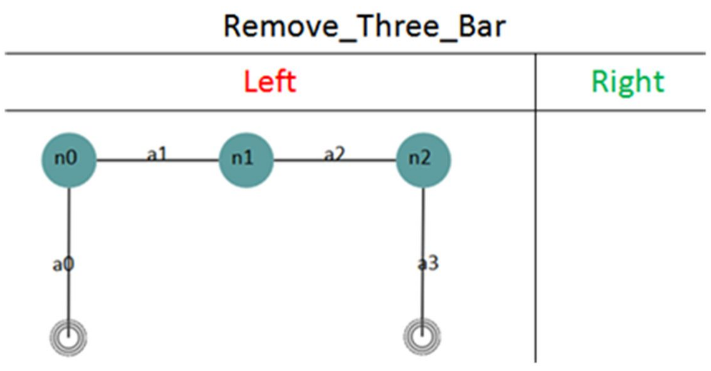


Figure 10: A *Remove_Three_Bar* rule.

generation. An example in Figure 11 shows how this rule is applied to an 8-bar linkage to obtain the rigid structure. The 8-bar linkage is generated after applying Transformation rule as it was discussed before, and since it contains a three binary link chain. This chain is deleted and a truss structure with 5 links is obtained.

The other method to prevent the newborn linkages having rigid structures is to conduct an examination by using the truss structures that have been discovered. The *Remove_Three_Bar* is only applicable to the 4-bar liked linkages to acquire the rigid structures. If a complex linkage contains a small rigid structure, like the 12-bar linkage with the 5-bar truss, this rule will not be capable to capture this truss structure due to this linkages does not act like a 4-bar linkage and there are not any 3 binary links chain within it. To solve this problem, the *Taboo_Structure* rules are used to recognize the smaller partial rigid structure in this linkage. For each of the *Taboo_Structure* rules, the LHS of it is set as the truss topology that was captured by *Remove_Three_Bar* before, and the RHS is blank since we only use the LHS to recognize rigid structure. Finally, all the *Taboo_Structure* rules are sorted inside a rule set called *Taboo_Ruleset*.

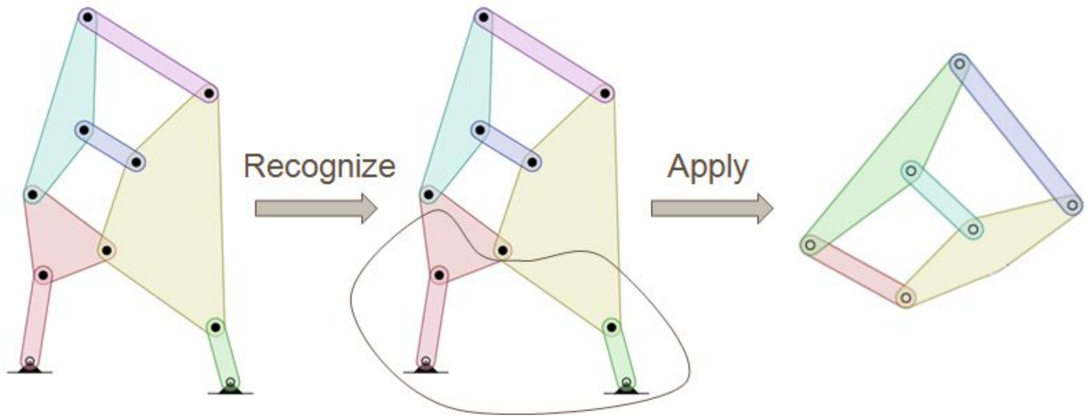


Figure 11: Applying a *Remove_Three_Bar* rule to identify a rigid structure.

In this case, a *taboo_ structure* rule with a 5-bar true in the LHS can recognize the partial rigid part in the linkage.

By using *Remove_Three_Bar* and *Taboo_Structure* these two kinds of rules, the problem of rigid sub-structure can be well avoided. When the search process goes further, more rigid trusses will be found and stored in the *Taboo_Ruleset*.

3.2.4 P-Joint Substitution Rule

Now the valid design space of R-joint topologies can be fully defined by the Generation, Transformation and Taboo detection rules that we discussed before. Since prismatic joints are also applied in real linkage design, introducing P-joints in mechanism topology synthesis can increase the diversity of the design space. The general approach to include P-joint linkage design into the search process is to apply a graph grammar rule to replace the revolute joint by prismatic joints, and a new topology with P-joint is created. However, the P-joint replacement process is not straightforward due to it could affect the rotatability of a link with R-joint on it. A prismatic joint is considered as a full joint, but its slot angle and the link for where it is placed to can affect the mobility and result in immovable link. For example, a four-bar linkage in Figure 12a that has 3 P-joints. Because those 3 slot are parallel to each other, the link 2 and link 3 can move along the slot

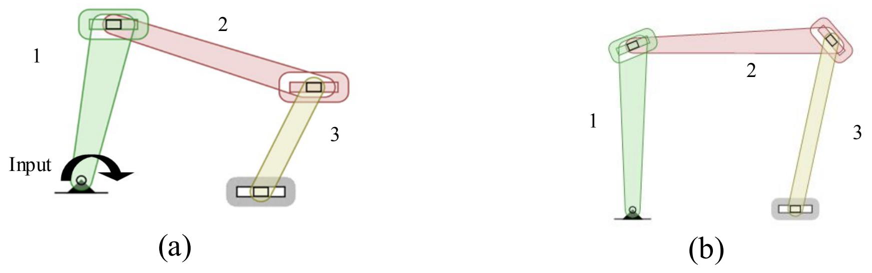


Figure 12: Two 4-bar P-joint linkage with different DOFs.

independently, and the mobility for this linkage is 2 instead of 1. In Figure 12b, these 3 P-joint now have the different slot angles, but the rotatability of link 1 about the revolute input joint is constrained by these P-joints in the linkage. Hence, we attempt to solve these problems and synthesize valid complex P-joint linkages. In this section, a P-joint substitution rule is used to recognize an arc that does not contain any joint type label and add the "p" label to this arc. After that, this arc is representing a P-joint in the linkage. This recognize-apply process is obvious, but in order to avoid the problems that were described before, some additional functions are required during the recognition-apply process. These additional functions are designed based on 3 constraints, and these constraints can ensure the P-joint substitution rule be able to identify the appropriate locations and place the P-joints correctly.

The first constraint is the slots of the sliding pair cannot be parallel. This problem was demonstrated by the example in Figure12a and it can be solved by assigning random angles to the P-joint slots. Because the topology cannot present the angle of a slot, this solution is applied to the models that are used for kinematic simulation.

The second constraint is two binary links with only P-joints cannot be connected together. Take the example in Figure 12b, the binary links L_2 and L_3 both have two P-joints. This result in L_2 and L_3 are the only movable links in this linkage and input link is fixed. Therefore, when replacing P-joint to a R-joint arc, its neighbor nodes and arcs needed to be checked beforehand to avoid having the PPP chain. Figure13 shows two binary chains and each of them has 2 P-joints. Suppose the arc *a0* is a potential location that a P-joint is going to be placed. Before that happen, an examination is needed to check if it is a valid location for this rule to apply. The nodes *n0* and *n1* are first noticed as 2-DOF nodes, which means they are representing binary links, and two P-

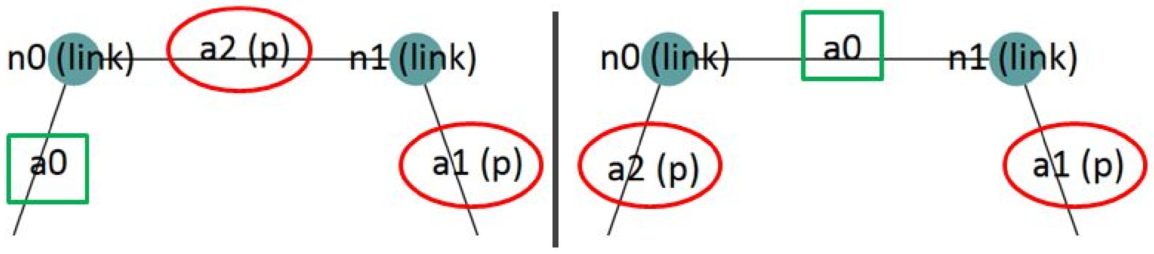


Figure 13: Two situations that P-joint cannot be placed to the a0.

joints (*a1* and *a2*) are detected. So placing P-joint at a0 will lead to a PPP chains and it is considered as a violation.

The third constraint is the number of R-joints in each kinematical loop should not be less than 2 [11], so the rotation of links with R-joints could be preserved. A linkage topology can be seen as a combination of different kinematical loops. When placing a P-joint to a recognized R-joint, all the loops that contain this same recognized joint in the linkage need to be examined. Starting from this R-joint, a best-first tree search method is used for searching the first loop that only contains 2 R-joints. If this loop is found, placing a P-joint to this R-joint will cause the linkage lack of rotation. For example, an 8-bar topology with 4 P-joint is shown in Figure14, and the arc *a7* is a potential location for adding a P-joint. Three loops in this topology are sharing the arc *a7* and the loop C that already has two P-joints is found first. So arc *a7* is not an option for placing P-joint. Alternatively, if no loop is returned, the location is valid for adding P-joint.

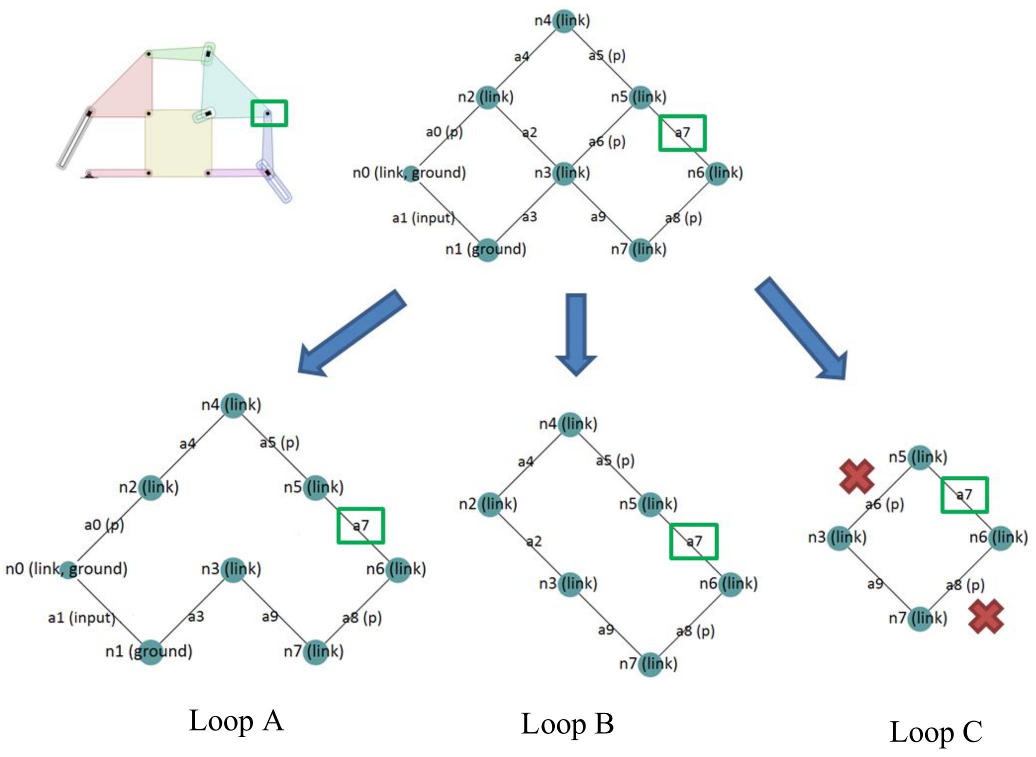


Figure 14: Three loops in 8-bar topology contain the same arc a7.

4 Tree of Topologies

Following the explanation of the Generation, Transformation, Detection and P-joint substitution rules, these rules can now be applied into the search process of R- and P-joint linkages. In this study, a search tree is used for exploring the design space of the topologies. The benefit of using a search tree is it directly relates to graph synthesis process. In the tree search process, each state of the tree can be presented by a graph. Once all of the graph grammar rules are designed, the new graphs will be generated as the next level in the tree.

Figure 15 shows how the graph grammar rules operate in a search tree and discovering the design space. This tree starts with a graph A which is set as an initial host graph or a seed graph in the searching process. After the seed is placed, graph grammar rules start to recognize the locations that they can be applied to. In this case, rule 1 and 2 are capable to be applied to the seed graph

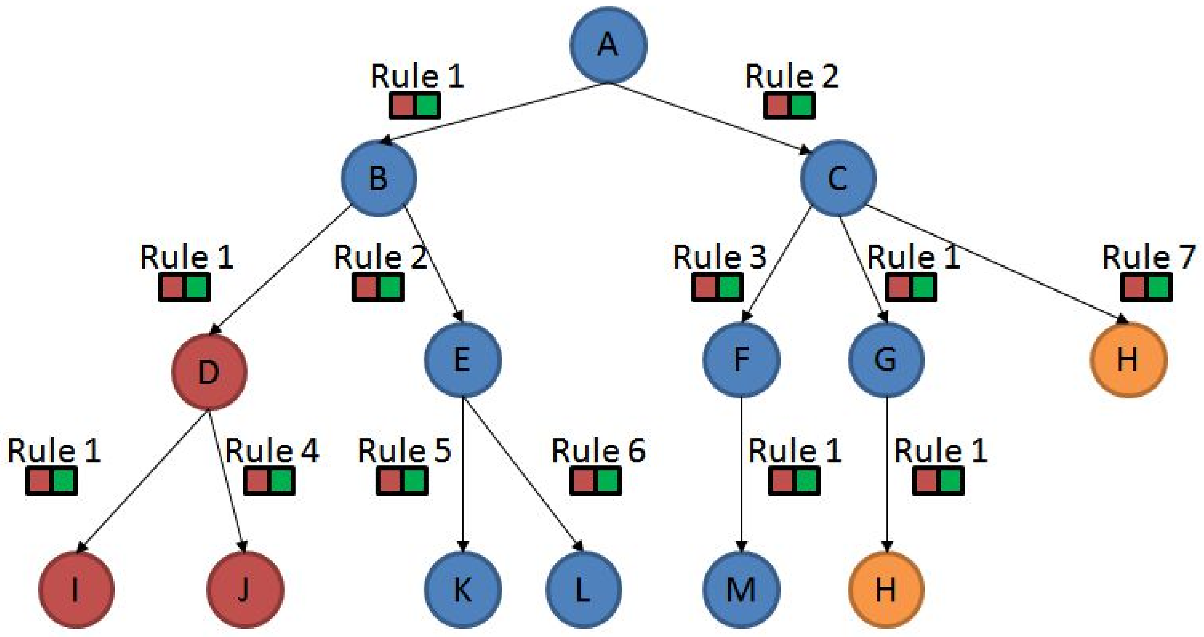


Figure 15: A tree of candidates is generated by applying different graph grammar rules.

and the new candidates graph B and graph C are created, and they form the second level of the tree. For the next level, each of the graphs at level 2 will be seen as a new host graph, and the recognize-apply process repeats again. This tree searching process finishes when no rule is recognized. So for the graphs that at the bottom of this tree, there are no rules that can be applied to them and this tree stops growing, and the whole design space is discovered.

Based on the expected result and the rules we built in this study, some invalid candidates need to be excluded from the tree searching process. In this study, the isomorphic topologies and topologies with rigid sub-structure are two types of unwanted results. Without removing these results, they will involve the search process and creating more undesired results. The isomorphism problem exists in the previous searching tree example. The two isomorphic candidates are represented by the some topology H, and they will be able to develop two identical sub-branches in the searching tree if more rules are applied to them. This problem should be avoided because generating isomorphic results is a redundant task, and it requires more time to finish the searching process. Consequently, only one of these topologies H should be kept in this tree. For the rigid sub-structure problem, the candidate D is also considered as a violation since it has a rigid structure within the topology. By setting this candidate as parent graph, the whole sub-branch that is generated by this candidate D contains all of the topologies that are with rigid structures. For this reason, the candidate D cannot be used for exploring and should be removed from the searching process once it is generated. To ensure each of the children in the tree is valid design, the problem of rigid structure and isomorphism need to be detected and solved.

The rigid structure in a graph can be captured by applying Taboo detection rules, as was mentioned in section 3.23. All of the new rigid structures serve as new rules to prevent candidates having these same structures. For the isomorphism problem, graphs that are considered as isomorphic will be removed by graph isomorphism check. The method of checking graph isomorphism will be discussed later. Since these two problems need to be resolved before storing the new candidate, addiction work is necessary during the synthesis process to ensure the validity of the new result.

4.1 Graph Isomorphism Exclusion

In this section, a methodology is established to examine graph isomorphism and avoid using the identical graph in further generation. When determining whether two graphs are isomorphic, we only focus on the connections between the nodes and arcs in the graphs. The local labels like "*ground*" and "*input*" in the nodes and arcs are not considered because they can be assigned after the topologies are designed. Figure 16 shows two linkage designs that isomorphic. Despite these two linkages have different numbers of ground joints and a different input links and they are regarded as two different mechanism designs, the graphs that represent the topology of these two structures are isomorphic. In this study, the isomorphic graphs are required to be detected and removed. Two procedures are constructed and be able to accomplished these requirements.

The first step is to compare the basic information between two graphs. This information includes number of nodes and arcs, the sequence of the node degree and the sequence of the secondary node degree. The number of nodes and arcs indicates how many nodes and arcs this topology contains. For each nodes in the graph, the degree of it is the number of the arcs are connected to it. For instance in Figure 17, the degree of node *n4* is 3 because there are three arcs *a2, a3* and *a6* attached to it. So the sequence of the node degree is obtained by sequencing all of the node degrees. Sequence of the secondary node degree is also needed because it can provide extra information for isomorphic check. For each of the nodes, this sequence is formed by

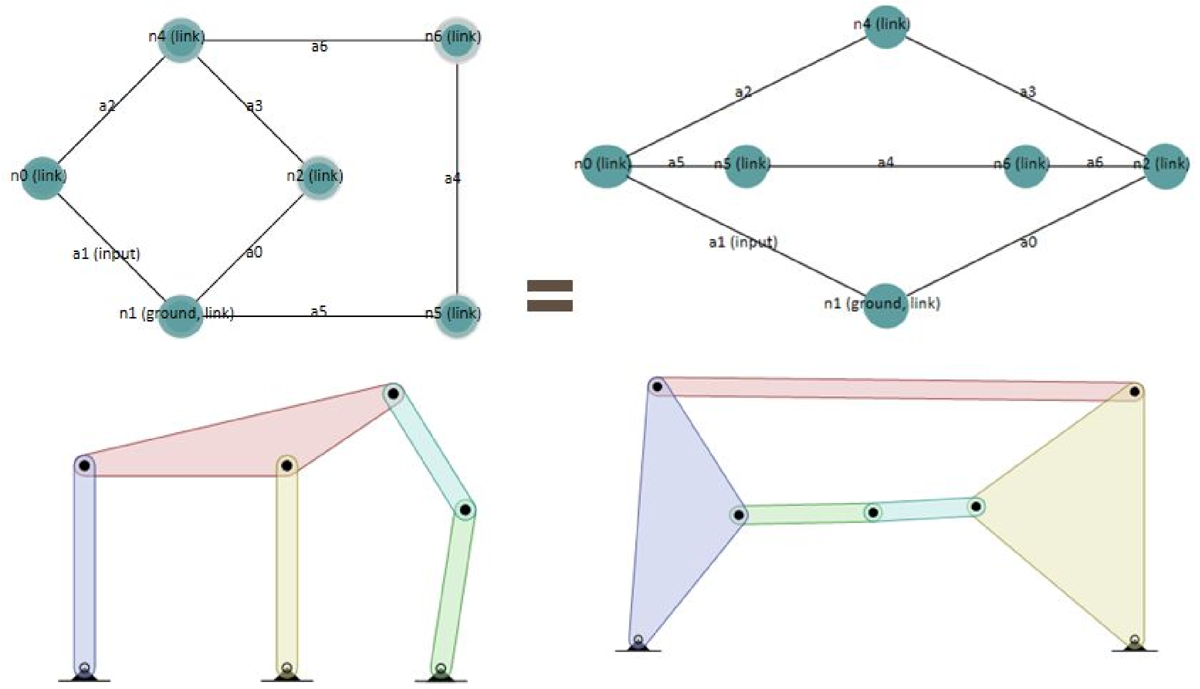
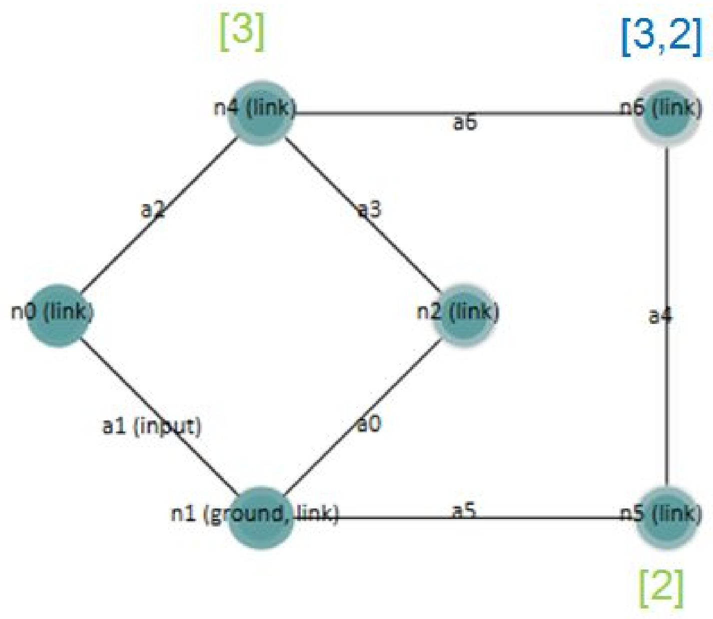


Figure 16: Two graphs that represent the linkages are isomorphic when the local labels are not considered.

indicating the degree of its neighbor nodes. In Figure17, the sequence of the secondary node degree for node *n6* is [3, 2] because the degree of its neighbor node *n4* is 3 and the degree of node *n5* is 2. All of these sequences will be sorted into a list and be compared to another graph during isomorphic examination. If two graphs have any different basic information, they are considered as two different graphs and can stay in the tree search process. By comparing this basic information, around 70 percent of the non-isomorphic graphs can be found.

For the graphs that share the same basic information, they still cannot be considered as identical results. So the recognize function is used as the second step for isomorphism checks. In this step, one of the graphs is set as a host graph, and the other is set as the LHS of a graph grammar rule. By using the recognize function, the rule will try to map the host graph. If this mapping



Number of Nodes
N=6

Number of Arcs
A=7

Sequence of Node degree
[3, 3, 2, 2, 2, 2]

Sequence of Secondary node degree
[3, 3]
[3, 3]
[3, 2]
[3, 2]
[2, 2, 2]
[2, 2, 2]

Figure 17: A 6-bar topology with the graph information.

succeed, these two graphs are identical because they can match each other, else they are non-isomorphic and will be sorted for further generation.

4.2 Procedure of Synthesis and Examination

The validity of a child graph is achieved by established the functions of isomorphism checking and taboo structure detection. Hence, a procedure of generating a child graph is required so these two functions can be executed during the process of generating children graphs. Figure 18 shows the whole procedure of how a child candidate with R-joint is generated from a parent graph with these additional functions. All the candidates are sorted in a queue in this process. The queue sorting method follows the first in first out (FIFO) criterion. This sorting method references

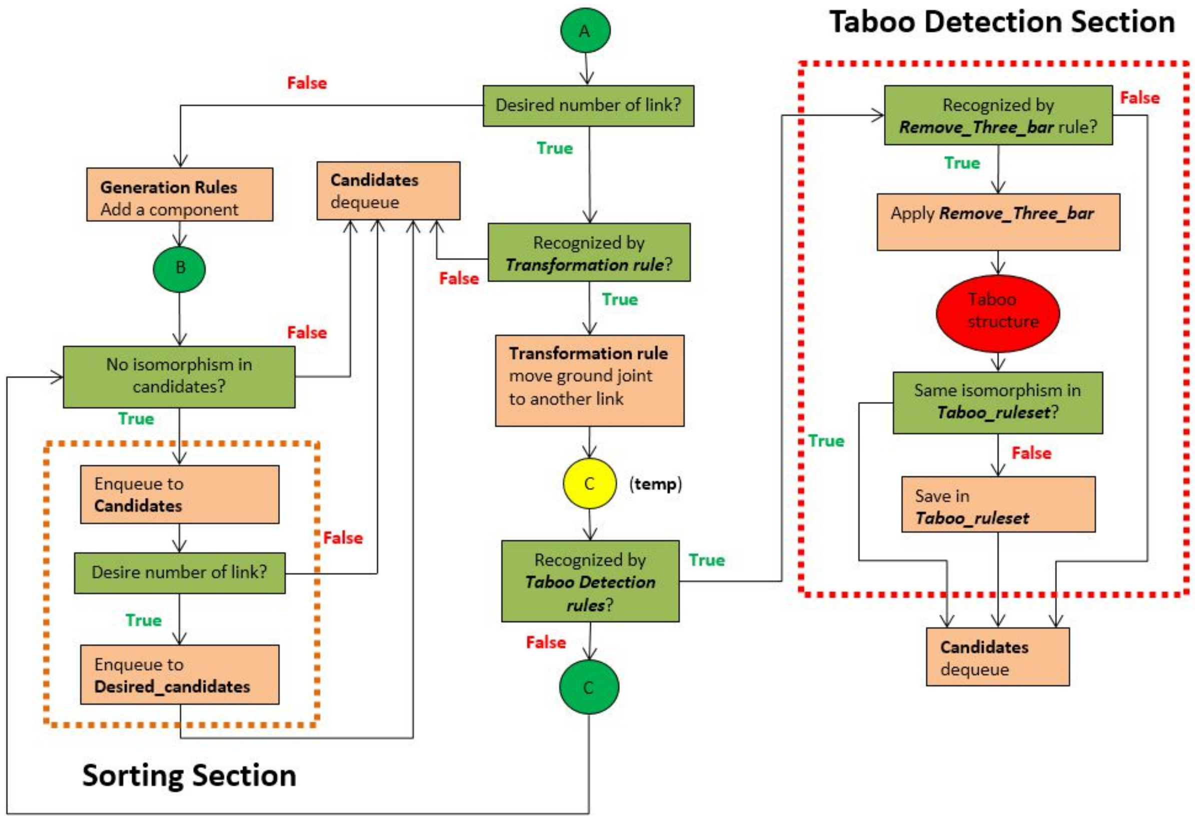


Figure 18: The procedure of generating R-joint topology B or C from the parent topology A.

the Breadth-First Tree Search process. Since we only focus on generating a single candidate, this tree search process will be discussed in next section.

At the beginning of the iteration, a candidate is dequeued from the *candidates* queue. Depend on which state of the tree this candidate is in, it can reference as a seed of the tree or a parent graph of a sub-branch. A Boolean statement is used in the initial step to check if the candidate A has the desired number of links, and it will decide which kind of graph grammar rules will be applied in the further step.

If the number of links in candidate A is less than the desired number, Generation rules will be applied to this candidate to increase the link number and the new child B with more links and joints will be created. After that, this child B needs to go through the isomorphic check and it will be sorted back into the *candidates* queue only if there no other isomorphism exist in there.

Otherwise, if the candidate A has the desired number of links, the process will go to the other direction. In this direction, first, if this candidate A only has two ground joints, then no rules are applicable to it since the *Transformation rule* require at least 3 ground joints to be recognized. So the search process along this branch is finished, and it is considered to be at the bottom of the tree and no children graph can be generated from it. Else, the Transformation rule will relocate one of these ground joints to a link and create a temporary candidate graph C. Since this graph may contain rigid structure after the application of *Transformation rule*, it will be checked by Taboo detection rules to determine whether it contains any rigid structure. If a rigid structure is found, this candidate is confirmed as an invalid result, and the rigid structure within it may be sorted in the *Taboo_ruleset* if it is a new truss structure. After that, this iteration is finished. On the contrary, the isomorphism of this graph C is checked. Like the candidate B, it will be sorted if it is the unique candidate in the whole tree.

The sorting section is after the isomorphism checking process, and any graph candidate that can enter this section is considered as a valid result. In this section, first, the valid candidates are enqueued to the *candidate* queue so it will be used as the parent graph in a new iteration. After that, the copy of it will be checked. If the number of links it has meets the design requirement, it will be sorted in the list called *Desired_candidates*, and the iteration restarts.

The functions of taboo structure detection and isomorphism checking are integrated into the procedure of a single topologies generation. Due to this, the validity of each of the newborn child topologies is guaranteed. When there is no more rule that can be recognized, this procedure stops and all R-joint topologies with the desired number of links are explored and sorted in the *Desired_candidates.*

4.3 Selection of Tree Search Method

With the procedure of examination and synthesis that were established above a tree search method now can be applied for fully exploring the valid design space of topologies. A suitable tree search method should be selected so it can explore the design space effectively. Because the design space needs to be fully discovered, some informed tree search methods like best-first tree search and A* tree search are not preferred because they only focus on finding a single optimal result in the tree. So in this section, the methods of breadth-first tree search (BFS), depth-first tree search (DFS) and random tree search are discussed.

The main difference between BFS and DFS is the orders of how the candidates are generated. In BFS, the exploration within one level of the tree should be completed before new candidates can be generated from this level. It is accomplished by the sorting all the candidates in a queue. As it was mentioned before, the queue sorting follows the first-in-first-out (FIFO) criterion. At first, all of the parent graphs that are at the same level are sorted in the queue. After the first parent is dequeued, new children that are generated by applying the recognized grammar rules to this parent and they are stored at the end of the queue. Later, the second parent is dequeued and this process repeats. When the queue does not contain any parent candidate, it means all of the candidates at the parent level have been visited and used for generating new children candidates. The children level will be completed once all of the children candidates are generated. Meanwhile, the queue only contains these newborn candidates and they sever as the new parents to start another exploration at the next level.

In DFS, the exploration is along a branch of the tree. Different from BFS, the DFS uses a stack for sorting candidates and the criterion of it is last-in-first-out (LIFO). The candidate that is last pushed into the stack will be the first one out of it. So in DFS, first it sorts all of the candidates that were created by the seed. After that, the candidate on the top of the stack will be popped out and become a parent candidate immediately to synthesize the children candidate graphs for the next level. These graphs will be pushed into the stack and the whole process repeat. When there is no applicable rule for a candidate once it is out of the stack, it is considered to be at the bottom of the tree, and the search process along this branch is finished. So another candidate that from previous level is popped out from the stack is used for searching along the other branch.

In random tree search, the searching process is randomly along a branch in the tree. Once a seed candidate is set in the tree, one of the invoked rules will be randomly selected and applied to it, and a new candidate is generated at the next level. Then a new random rule selection and application process happen again to this candidate. This process stops when it finds the desired candidate. Otherwise, it will continue until it reaches the bottom of the tree, and the whole process restarts from the initial seed state.

When the design space of mechanism topologies needs to be entirely discovered, the BFS and DFS methods are first considered because the fully exploration of the design space by using random tree search is not promised. These two methods are compared by using them to explore the space of 8- and 10-bar R-joint linkages. The times for completing the 8-bar linkages search are about the same. But a difference appears during 10-bar linkage searching. It is shown that for synthesizing the same amount of results, BFS uses much shorter time than DFS. The duration for running BFS is 38 seconds and for DFS is about 4 minutes. As the topologies become more complicated, the duration of finishing a tree search will increase significantly. We speculate the reason why BFS takes less time is because the isomorphism checking finishes faster during the search process. For isomorphism checking, all of the candidate topologies in the queue or stack need to be examined entirely. Due to the different sorting methods, isomorphic graphs in BFS may be discover earlier than which in DFS. For this reason, the BFS method is selected as it can generate the whole design space with less time, and it is more adapted to the rules we construct in this study.

For random tree search, when the more candidates are found, the possibility of generating undiscovered candidate becomes smaller because the random generation could create isomorphic solutions repeatedly for every restart. Although the random tree search is not suitable for exploring the whole space, the time that it spends on finding a single desired candidate is very fast. For this reason, a design exploration tool is built based on the random tree search. This tool can create a mechanism topology that satisfies the required of number of links, P-joints and RP-joints. The detail of this generator will be discussed in the result section.

4.4 Example of a Tree Topologies

After selecting the BFS as the tree search method, the exploration of mechanism topologies can start. Within the tree, each of the candidates is generated by the graph grammar rules and no rigid structures and isomorphic graphs exist in this tree. When the exploration is finished, the arrangement of the desired mechanisms can be presented by a tree of topologies. In this section, the procedure of generating 4, 6 and 8-bar linkages with R- and P-joints is shown in Figure 19. Five rules are used for this tree search, which are *Dyad, Triad, Double-Triad, Transformation* and *P-joint substitution* rules. They are represented by R1 to 5 respectively in the tree of topologies. A pendulum with 2 links 1 R-joint and 0 P-joint (#L=2, #R=1, #P=0) is placed at the first level as a seed of the tree because it is the most simple 1-DOF R-joint linkage. The seed is recognized by rules R1, R2, R3 and R5. After applying these rules to the seed, four children graphs are generated at the second level. These are the typical four-bar linkage (#L=4, #R=4, #P=0), a six-bar linkage (#L=6, #R=7, #P=0), an eight-bar linkage (#L=8, #R=10, #P=0) and a two-bar linkage with a single P-joint (#L=2, #R=0, #P=1), and the search process at second level is finished. A recognize and apply process repeats to generate new levels of candidates until no rules can be recognized, then the exploration stops and the tree will contain all the topologies.

In each of the iterations, the isomorphic candidates or candidates that contains rigid structure will be discarded. If the discovered rigid structure is a new taboo structure without a rigid sub-structure, it will be sorted in the *Taboo_ruleset*. Hence, this breadth-first search is also able for exploring the rigid structure topology. Once the exploration is finished, the space of the rigid structures can be obtained.

During the searching process, once a candidate contains a P-joint, only the *P-joint substitution* rules will be used for synthesis of its successors. The reason for this is it can reduce the searching space. In a fully discovered space, each of the R-joint linkages in the space can be used to generate P-joint linkages. If the additional generation and Transformation rules are applied to a P-joint linkage, there will be another isomorphic topology with same P-joint arcs exist in the tree. This results in spending more time to eliminate the redundant results. So the bottom of the tree contains results that have the maximum number of P-joints and no more P-joint can be added. Also, these results are originally generated from R-joints only linkage. Hence, when the tree search process stops, 4, 6 and 8-bar linkages with only R-joint or R- and P-joints can be obtained from the tree.

R1	Add Dyad
R2	Add Triad
R3	Add Double Triad
R4	Relocate Ground joints
R5	Add P-joint

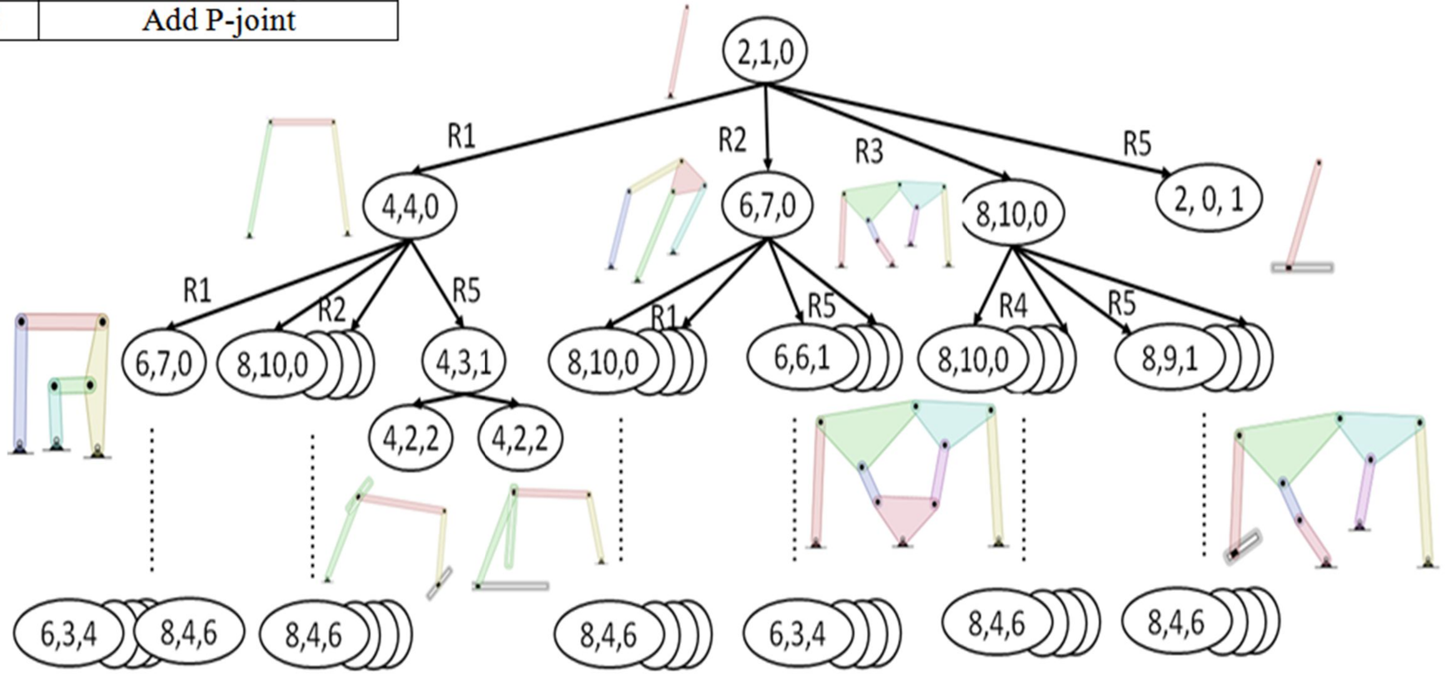


Figure 19: A tree of topologies that contains R- and P-joint linkages.

5 Exploration of Higher DOF R-joint Topologies

By now, the valid design space of R-joint 1-DOF mechanism topologies are fully explored by using the graph grammar rules to produce a BFS tree search. The graphical representations for all of these results are obtained. With these results, a further exploration of higher DOF topologies could be easily achieved by degenerating 1-DOF topologies. This process is also accomplished by graph syntheses, and a new graph grammar rule is designed to obtain higher DOF close chain topologies. Since the multiple inputs linkage mechanisms are also used extensively in different applications, we also capture these mechanism topologies in this study.

5.1 Degeneration Rule

Since the mobility of a linkage can be changed by increasing or reducing the number of components, the approach of degenerating 1-DOF topologies is the similar to obtaining the rigid structure. Recall the modified Gruebler's equation (3) in section 3.2.3. In order to increase 1-DOF, the combination of L_{remove} =1 and J_{remove}=2 is selected. This combination references a binary link with two revolute joints. The mobility of a linkage will increase by removing this component. Based on this, a graph grammar rule is designed to recognize the binary links and remove them from a topology, and a new topology with higher mobility is created.

The design of the Degeneration rule and its application is shown in Figure 20. On the LHS of this rule, the node *n2* represents a binary link that connects between two neighbor nodes *n0* and *n1*. In order to prevent degenerating to an open chain topology, each of the node *n0* and *n1* is set to connect to 3 arcs individually, which means the link that is recognized by *n0* or *n1* at least contain more than 3 joints and it is not a binary link. The reason for this setup is if the binary link is originally connected to another binary link and be taken out, the binary that it disconnected from

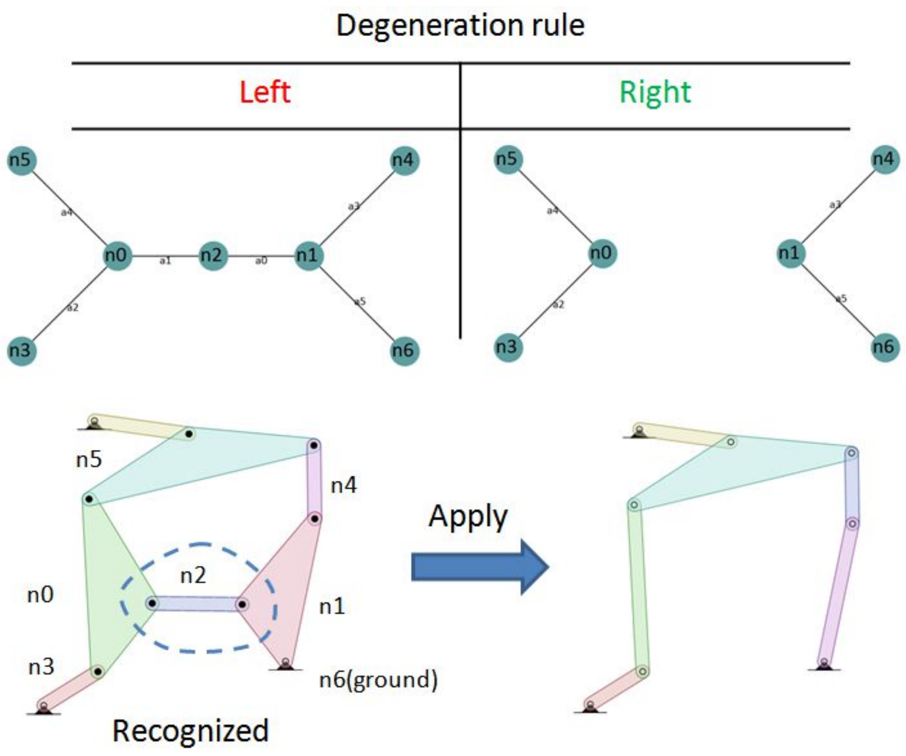


Figure 20: A Degeneration rule and its application of removing one binary link from an 8-bar linkage to create a 7-bar 2-DOF linkage.

will become a single pendulum in the topology, and an open chain is form. Also, the node *n2* is set to 2 degrees strict matched, so it can only recognize the 2 degree nodes, which is referred as the binary links. An 8-bar linkage in Figure 20 is used to demonstrate how this rule is applied. In this example, a binary link *n2* is connected to two ternary plates *n0* and *n1*. The LHS of this rule recognizes of the location *n0* to *n6* respectively. After it is applied to this topology, the link *n2* is removed and a 7 bar linkage with 2-DOF is created.

5.2 Degeneration Process

Unlike the method of exploring 1-DOF linkage, there is only one graph grammar rule is used for discovering the space of higher DOF linkages. So no tree search method is used to populate the results and no rigid structure problem needs to be considered since the degeneration process starts from the all of the existing valid 1-DOF topologies. This is the reason why we only focus on generating 1-DOF topologies first. Instead of make extra graph grammar rules to synthesize higher DOF linkages, they can be explored just by applying one rule to the obtained results.

Since applying this degeneration rule to a topology can increase 1-DOF, so the space of 2-DOF linkages can be created by applying this rule to all the 1-DOF topology. The isomorphism problem exists within these 2-DOF topologies. All of the newborn topologies are sorted into a list, and the redundant results are excluded. In the end, this list only contains valid 2-DOF topologies. Because this rule is universal to all of the topologies, so the higher DOF close chain topologies can be always obtained from the existing results.

6 Results

This synthesis method was applied to search all 1-DOF linkages with different number of links and joint types, and the results are compared with other researchers. As is done in the related work, this topological comparison often focuses only on the graph connectivity and no distinction is made as to what link is ground and what joint is the prime mover. Therefore, in the grammar rules and tree search performed here, the local labels ground and input will be removed. It is important, however, to consider the effects of the ground when populating the space for defining distinct kinematic behaviors. For example, the engineers are is likely aware that six-bar revolute joint mechanisms come in five varieties if ground is considered (Watt I, Watt II, and Stephenson I to Stephenson III) but just two if ground is not (Watt and Stephenson). However, it is possible with the rules to create the expanded ground-sensitive configurations as well. Additionally, the rigid structure topologies that with 5, 7, 9 and 11 links are captured. All of these topologies are accessible, and it is the first time that the topologies of 11 bar trusses structure become available.

The topologies with higher DOF can also be explored by degenerating the 1-DOF results. In the study, we are interested in the close chain topology designs, so a graph grammar rule is used to degenerate the results and avoid making open chain topologies. The number of topologies for each DOF matches other researches. However, the simulator we use in the study can only simulate 1-DOF linkage. So we can only provide the graph representation of these multiple DOF topologies in this study.

6.1 Mechanism with only Revolute Joints

The results for 6, 8, 10 and 12-bar 1-DOF linkages are in Table 2. For 6, 8, and 10-bar linkages, the results match other authors' findings [1] [2] [3], which are well established. For 12-bar linkage synthesis, 6856 topologies are found, and the amount is the same as Tuttle’s [1] and Lee & Yoon’s[3] studies. Each of these 12-bar linkages does not contain any rigid structure due to the examination of rigid structure. In Sunkari and Schmidt’s study[4], they infer the reason for why Hwang and Hwang having larger results is because the degeneracy testing algorithm they use does not remove all the infeasible solutions. This proves that our synthesis method correctly spans the design space for mechanisms with only revolute joints. The results for 14-bar linkage is different from Tuttle’s [1] and Lee & Yoon’s[3] studies. Also, the validity of each result from their studies cannot be tested because that is no available access to those linkages. In this study, the total number of 14-bar linkages was not obtained due to a prohibitively long search process, but it appears to be creating valid 14-bar linkages individually. It is important to note that all of our results are explicitly captured in computer files. Related works only publish 248 topology graphs for 6- to 10-bar and they are difficult to be accessible.

This synthesis method is also able to create linkages like 16- or 18-bar mechanisms. Again larger rigid structures may exist, but our detection of rigid structures will ensure that only feasible solutions are presented. While the entire space of solutions may not be explicitly created (e.g. each saved to a computer file), any number of valid solutions can be created.

Table 2: Results for 6, 8, 10- and 12-bar R-joints linkage compare with other authors’ works.

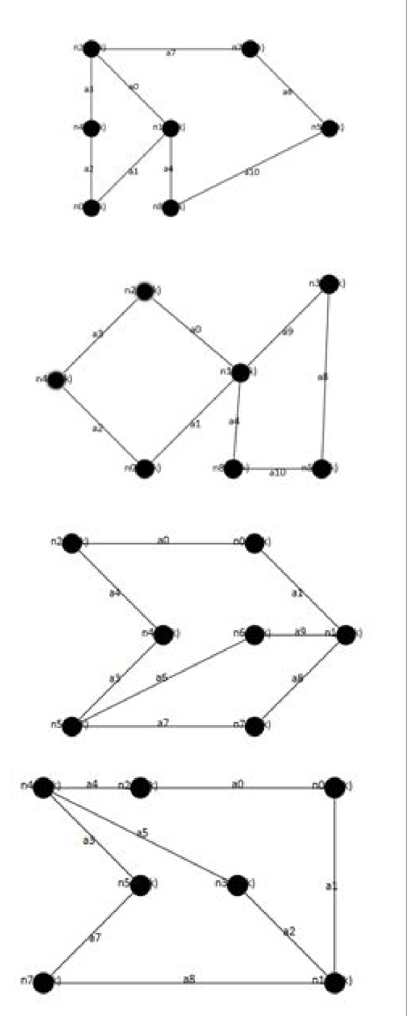
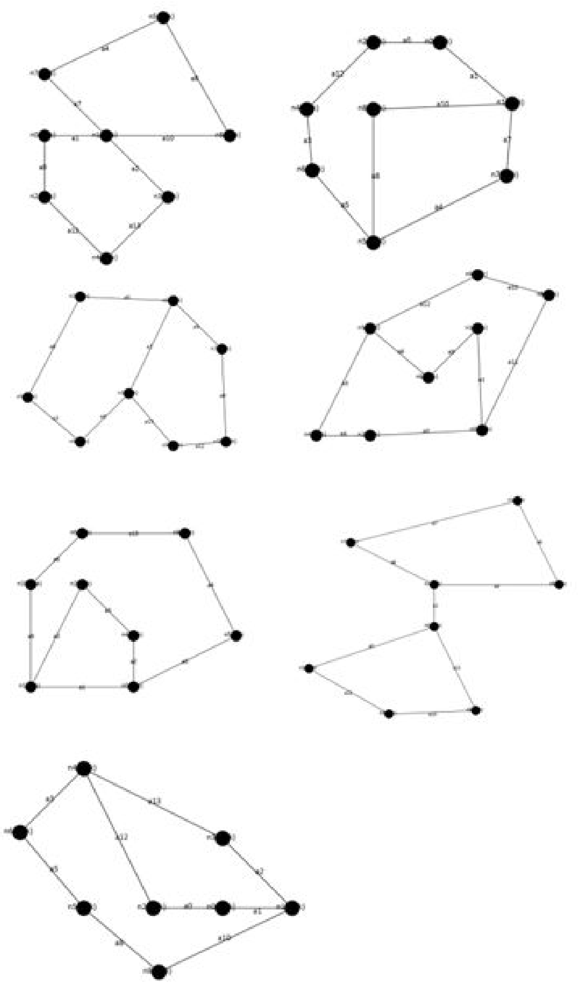
	Results	Tuttle	Lee & Yoon	Hwang & Hwang
6- bar	2	2	2	2
8-bar	16	16	16	16
10-bar	230	230	230	230
12-bar	6856	6856	6856	6862
14-bar		318123	275255	

The result for higher DOF linkage is in Table 3. For this time, the results we generate match Hwang & Hwang's study. Compared to Lee & Yoon and Tuttle study, our results are consistently higher. The topologies from these researchers are not available, so we cannot draw the conclusions on why these results are different. But we explicitly capture the total results of 2-DOF 7 bar linkage and 3-DOF 8-bar linkage with graph files. So Lee & Yoon and Tuttle are not correct because we obtained more topologies for these two results and they are shown in Table 4.

Table 3: Comparison between the results of higher DOF topologies.

	Results	Tuttle	Lee & Yoon	Hwang & Hwang
2-DOF				
7-bar	4	3	3	4
9-bar	40	35	35	40
11-bar	839	753	753	839
3-DOF				
8-bar	7	5	5	7
10-bar	98	74	74	98
4-DOF				
9-bar	6	10	10	6

Table 4: Results of 2-DOF 7 bar topologies and 3-DOF 8 bar topologies

2-DOF, 7-links	3-DOF, 8-links
	

6.2 Rigid Structures

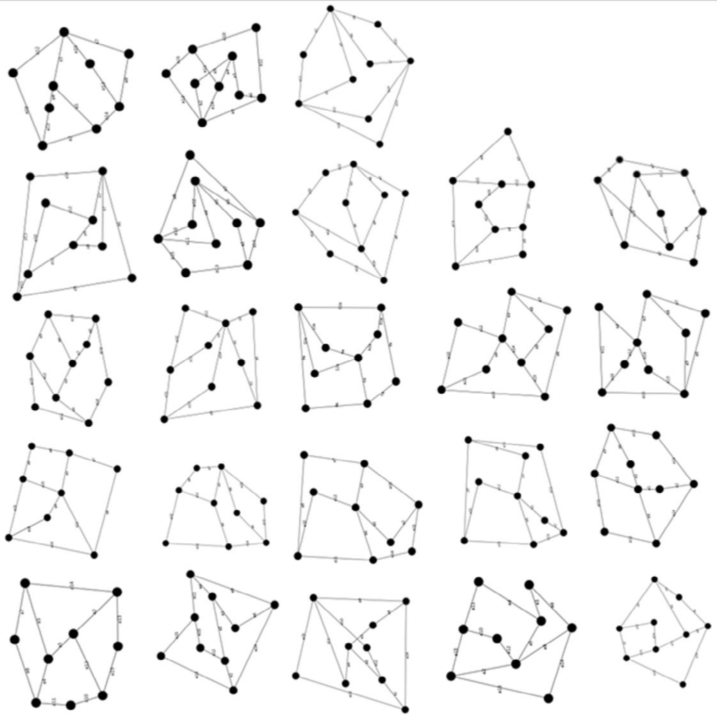
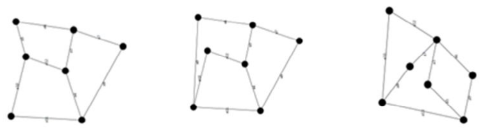
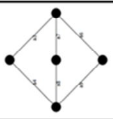
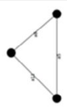
All of the rigid structures with 5, 7 and 9 links are obtained during R-joint linkages synthesis. This process is accomplished by the Taboo detection rules and the truss result are depend on the complete exploration of the design space of the 8- to 12-bars topologies. For 11 bar rigid structures, they can be gained through degenerating all of the 12-bar R-joint topologies. In this degeneration process, one of the binary links with 2 R-joints is taken out from the linkage. Based on the Gruebler's equation (3), this will reduce 1-DOF for the linkage, and the rigid structure topology with 11 links is formed. Table 5 shows the total number of each kind of rigid topologies and the graph representations of rigid structures with 5, 7 and 9 links are in Table 6.

The amount of the 5 to 9-link rigid structures matches Rojas's study [19]. For the 11 links results, we obtain more new truss structures than which have been published. In Yang and Yao's study[20], they state the total amount of 11-link topologies is 239. By using degenerating method, we create 562 11-link rigid structures. Each of these new results is checked by Taboo detection rules to ensure they do not contain any rigid sub-structures. We also provide the topology for each of these results. Before here, Rojas states that the truss structures are only available up to 9 links [19], but with this study, now each of the 11 links truss topologies is accessible as a graph that can be recognized by computer.

Table 5: Amounts of 3,5,7,9 and 11 links rigid structure topologies.

Number of links	3	5	7	9	11
Number of Topologies	1	1	3	28	562

Table 6: Topologies of 3, 5, 7 and 9 links rigid structures

9-links	
7-links	
5-links	
3-links	

6.3 Mechanisms with Revolute and Prismatic Joints

When replacing R-joints with P-joints in a linkage, the grammar rule is based on the last two constraints mentioned in section 3.2.4. The results for enumerating the size of the space are shown in Table 7. All of the 50 6-bar topologies involving combinations of R- and P-joints have been simulated in order to prove their 1-DOF validity. Furthermore, none of them has link that move independently of the prime mover. This result is different than Sardain's study [10]. The reason for that is because the P-joint substitution rule can be applied to the linkage that primary components and secondary components cannot be clearly defined. For example, the kinematic property of each link in a Stevenson-II 6-bar 4 P-joints linkage in Figure 21 depends on others. So Sardain's method is not suitable in this situation.

Beyond the 6-bar linkages, values are shown for 8, 10, and 12-bar R and P mechanisms. Unfortunately, at the time of writing, it was realized that errors occur in some of these generated mechanisms. Therefore, the reported numbers that are followed by an asterix (*) are upper-bounds instead of actual values. This is a result of a higher-order constraint that was not captured by the rule and has not been explicitly defined in the literature. This is illustrated by the topology in Figure 21(a), which is an 8-bar linkage with three consecutive P-joints. The positions of these 3 P-joints

Table **7:** Results for 6, 8, and 10-bar linkages

		2bars	4bars	6bars	8bars	10bars
Number of P-joints	1	1	1	6	84	2307
	2		2	16	345	12925
	3			19	780*	43883*
	4			9	1083*	97138*
	5				682*	
	6				140*	

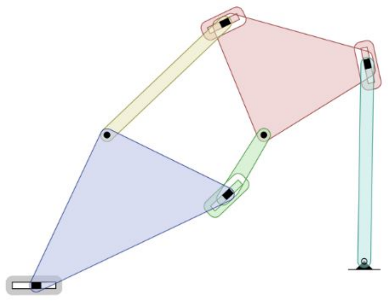


Figure 21: A Stevenson-II linkage with 4 P-joints [21].

do not violate the previous rules, but still the linkage has 2-DOF. The dyad (L1 and L2) in the linkage does not affect the movement of this linkage. After removing this dyad, a new 6-bar linkage is created, and a PPP dyad exists in this linkage which is considered as a violation in Figure 22b. The new challenge arises to include this as a constraint in the P-joint substitution rule to reduce these numbers to the actual set of feasible R and P mechanisms. Part of the difficulty is allowing valid PPP chains such as shown in Figure 22.

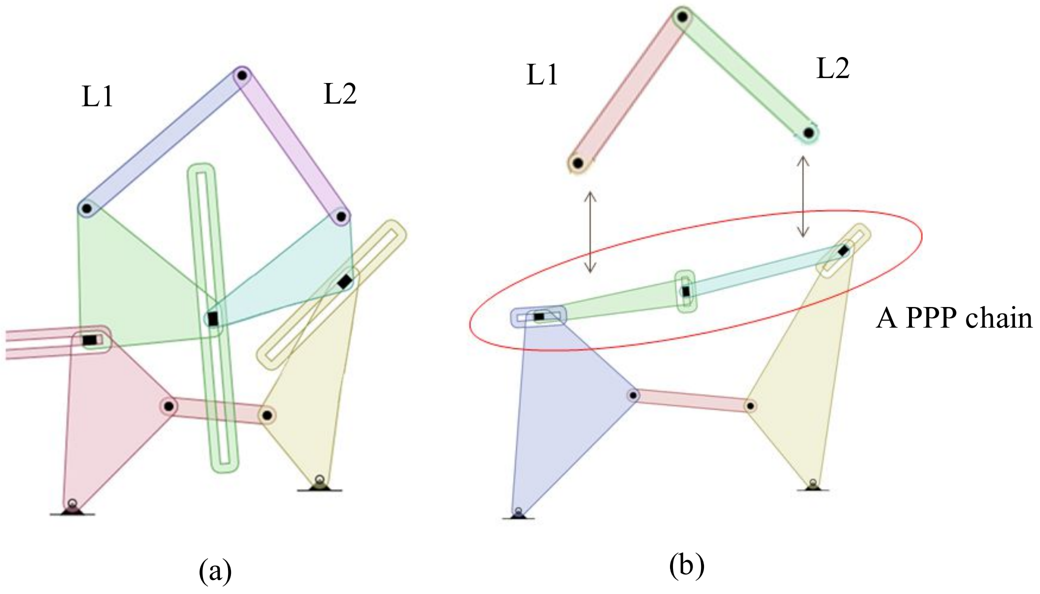


Figure 22: A dyad component attached to a PPP chain and this result is invalid.

6.4 Mechanism with Revolute and Pin-in-Slot Joints

After having the results with P-joints, we can introduce RP-joints to better represent real mechanisms. This is opposed to Tsai's approach where the RP-joint is a special case of collocating an R-joint at the same coordinates as a P-joint. However, this neglects the fact that the number of nodes will no longer correspond to the number of rigid links. Our goal is to include mechanisms like the scotch-yoke (with L = 3) and various other complex linkage with RP-joint designs.

The approach to generate an RP-joint in the graph is to search for binary links with one R-joint and one P-joint. The application of the rule deletes the binary link and replaces the joints with a single RP-joint. The rule is illustrated in Figure 23. Since we have some violations in P-joints results, we only show results for linkages with R and RP joints and the results we have are still valid. The reason for that is when generating certain number of RP joint, we need the select the linkage that has the same number of binary that contains R and P joints. So this can avoid selecting a linkage that contains PPP chain.

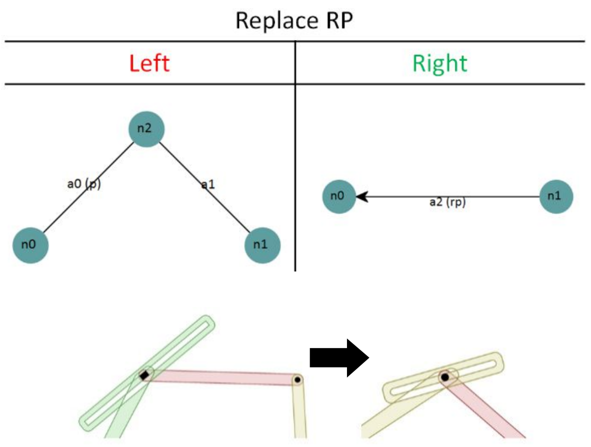
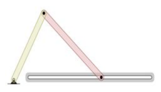
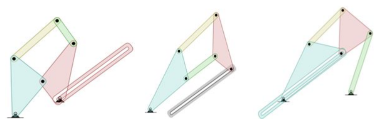
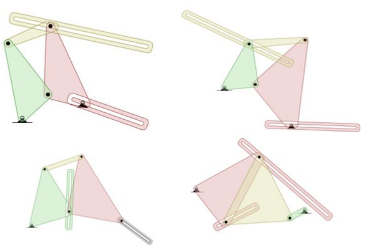
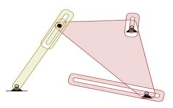
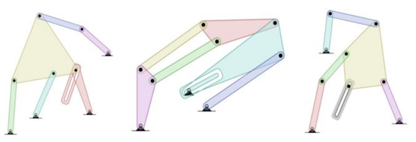
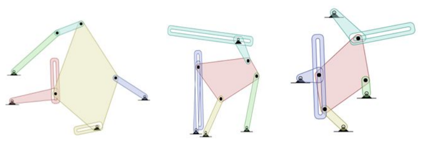
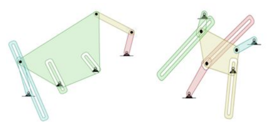
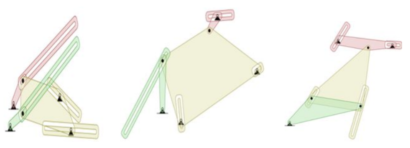


Figure 23: A *Replace_RP_Joint* rule with its application.

During isomorphism comparison, the direction of the arc representing an RP-joint is excluded since we are only concerned with the location of the RP-joint in the topology. The number of topologies for 3- to 7-bar linkage comprised R- and RP-joints is in Table 8. Since we use a one-way direction to present the order of the slot and pin design, the total number of R- and RP-joint topology designs could be calculated, which is the number of results we obtain multiplied by 2^n. The n is the number of RP-joints at the corresponded level. For example, we obtain 3 5-bar 1 RP-joint topologies without considering the direction of the RP-joint arc in this study. The total number of this topology designs can be calculated as $3*2^1$, which is equal to 6. This is because the RP-joint in each of these 3 topologies can be represented twice by using two different directional arrows separately.

The design space of the RP-joint topologies is first obtained in this study. With the representation of RP-joint, the real linkage design with pin-in-slot joint could be represented better. Also, the RP-joint topologies may provide alternative solutions in the linkage design since we are able to introduce RP-joint to the complex linkage syntheses.

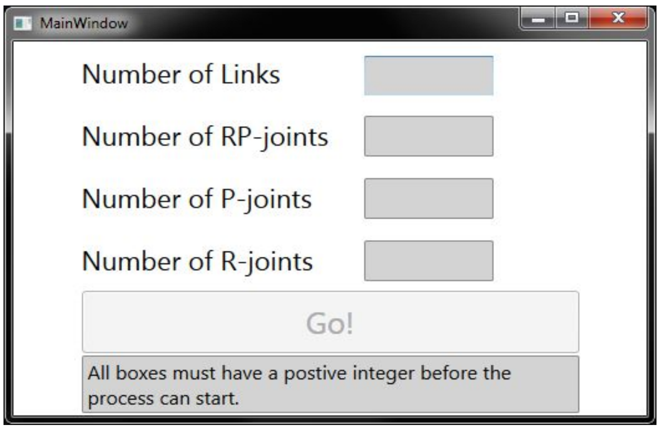
Table 8: Results for RP-joints linkages

	Total	Examples
Generated from 4-bar linkage		
3bar-1 RP	1	
Generated from 6 bar linkages		
5bar-1 RP	3	
4bar-2 RP	4	
3bar-3 RP	1	
Generated from 8 bar linkages		
7bar-1 RP	32	
6bar-2 RP	41	
5bar-3 RP	20	
4bar-4 RP	5	

6.5 Design Exploration Tool

With all the graph grammar rules and rigid structures we constructed, a design exploration tool is created to generate user specified linkage. Because the process of design and building the model of complex linkages is tedious, this tool provides a convenient way to generate the 1-DOF linkages with user's requirement. The Graphical User Interface (GUI) of this tool is shown in Figure 24. Based on the desired number of each component, this tool will synthesize a topology mechanism that can satisfy the requirement and put it into the kinematic simulator to simulate the movement of this topology.

The process of generating desired topology is accomplished by modified random tree search method. In this tree search, the graph grammar rules are applied in sequence base on their functions. First in this search, the generation and Transformation rules are randomly selected and applied to create the R-joint topology that has the desired number of links. This topology is checked by the Taboo detection rules to ensure its validity. After that, if a P-joint design is required, the P-joint substitution rule will randomly apply to the different locations of the topology until it

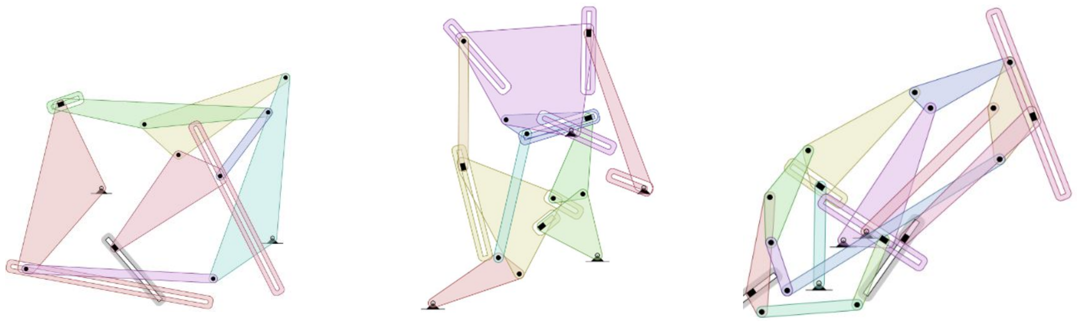


All boxes must have a postive integer before the process can start.

Figure 24: The Graphical User Interface (GUI) of the design exploration tool.

meets the requirement. If RP-joint is involved, we need to make some changes on the user input to ensure the result that it returns can meet the user's specification. Because this RP-joint substitution rule is to combine an R-joint and a P-joint together to create a Pin-in-Slot joint, meanwhile a binary link between these two joint is taken out. So the first step is to randomly generate a linkage with P-joints, and the number of P-joints should be the same to the desired number of RP-joint. In the same time, the link number of it should be equal to the user design number plus the number of PR-joint. So this change can provide enough recognizable locations for the RP-joint rule. And the result can be generated.

By using this design exploration tool, a random topology with desired number of link and joint types can be created quickly. This topology can be converted to a real design linkage and simulated in PMKS. Some of the results that are generated by this tool are shown in Figure 25. Usually, designing and modeling the linkage with this complexity is a time-consuming process. But the design exploration tool now can provide a simple way to achieve generating complex linkage mechanism that can satisfy the user requirement.



8bar, 2 P-joints, 2 RP-joints

(a)

10bar, 4 P-joints, 4 RP-joints

(b)

14bar, 5 P-joints

(c)

Figure 25: Three mechanisms are created by the design exploration tool .

7 Conclusions

This study establishes a new graph representation for linkages. The addition of RP-joints is introduced to the graph to represent a linkage that has pin-in-slot joint. So any planar mechanism linkage with R-, P- and RP-joints can be represented correctly by using this graph representation. This representation is easy to understand and it can capture the most of the information about the linkage. Such as the combination of the links and joints, the joint type of each of the joint and prime mover of the linkage.

With the graph representation, a valid design space for linkages comprised with R-, P- and RP-joints can be explored by graph synthesis. This synthesis process is accomplished by the graph grammar rules. These rules can provide a design language so each of the new designs can be generated by following these criteria. The function of a graph grammar rule is to modify a host graph to create a new topology. In this study, the Generation, Transformation and Detection grammar rules are constructed and are used to generate R-joints links. Generation rules can add components to a linkage without changing its mobility. When a linkage contains multiple ground joints, the Transformation rule can detach one of the ground joints and relocate it to another link. After that, rigid structures are be detected within the linkage by Detection rules. These rules are carefully constructed to make sure the results can meet the design requirement. A Breadth-First tree search method is used to explore the 1-DOF results by applying these grammar rules to synthesize new candidates in the tree. The correctness of this method is proven by having the same number of unique solutions to the results of other author's works. We also explore the topologies with higher DOF. By verifying the topology graphs, it shows the graph synthesis approach can cover more valid designs than other studies.

After R-joint linkages are constructed, P-joints can be added to the linkage by replacing R-joints. Because the P-joint can affect the rotatability of the linkage, this P-joint substitution rule is designed based on the 3 constraints to avoid rotatability being affected after introducing P-joint to the linkage. By this approach, we successfully obtain the valid design space of the 4 and 6-bar topology design with multiple P-joints. However, we found out these 3 constraints are not enough to avoid this problem after synthesizing the 8-bar results, and this problem is harder than expected. Although violations exist starting at 8-bars level, we define the upper bound of the P-joint topology design and the valid result is included. For the RP-joint generation, a binary link with R- and P-joint is replaced by a RP-joint. Since violation happens by applying P-joint, we only search for linkages that only have R- and RP-joint and it can prevent using violated P-joint results. The results for the enumerations of the space of RP-joints topology have been fully examined and it is an innovation presented in this paper.

In this study, a design exploration tool provides a convenient way to simplify the process of designing complex mechanism topologies with different types of joints. This tool is designed based on the modified random tree search method and the routine of graph synthesis process. The user only needs to specify the number of link and type of joints, and the random tree search process can return the desired topology, and its kinematic movement is simulated by the PMKS. Theoretically, this design tool can provide a 1-DOF topology with unlimited number of links. So it could reduce the time on designing complex mechanism and verify its validity.

Bibliography

[1] Tuttle.E.R, “Generation of planar kinematic chains,” *Mech. Mach. Theory*, vol. 31, no. 6, pp. 729–748, Aug. 1996.

[2] W.-M. Hwang and Y.-W. Hwang, “An algorithm for the detection of degenerate kinematic chains,” *Math. Comput. Model.*, vol. 15, no. 11, pp. 9–15, Jan. 1991.

[3] H.-J. Lee and Y.-S. Yoon, “Automatic Method for Enumeration of Complete Set of Kinematic Chains,” *JSME Int. journal. Ser. C, Dyn. Control. Robot. Des. Manuf.*, vol. 37, no. 4, pp. 812–818, Dec. 1994.

[4] R. P. Sunkari and L. C. Schmidt, “Structural synthesis of planar kinematic chains by adapting a Mckay-type algorithm,” *Mech. Mach. Theory*, vol. 41, no. 9, pp. 1021–1030, 2006.

[5] “Grinding Robot Arm,” *www.pushcorp.com*, 2015. [Online]. Available: http://www.pushcorp.com/Success Stories/Grinding-2.jpg .

[6] W.Huang and M.I.Campbell, “Planar Mechanism Kinematic Simulater:Complex path synthesis.” [Online]. Available: purl.org/pmks/MR13.

[7] L.-W. Tsai, *Mechanism Design: Enumeration of Kinematic Structures According to Function*. CRC Press, 2010.

[8] M. I. Campbell, “Planar Mechanism Kinematic Simulator,” *Oregon State University*, 2014. [Online]. Available: http://designengrlab.github.io/PMKS/.

[9] L. S. Woo, “Type Synthesis of Plane Linkages,” *J. Eng. Ind.*, vol. 89, no. 1, p. 159, Feb. 1967.

[10] P. Sardain, “Linkage synthesis: Topology selection fixed by dimensional constraints, study of an example,” *Mech. Mach. Theory*, vol. 32, no. 1, pp. 91–102, Jan. 1997.

[11] M. I. Campbell, “A Graph Grammar Methodology for Generative Systems,” *Methodology*, pp. 1–25, 2009.

[12] M. I. Campbell, R. Rai, and T. Kurtoglu, “A Stochastic Graph Grammar Algorithm for Interactive Search,” in *Volume 8: 14th Design for Manufacturing and the Life Cycle Conference; 6th Symposium on International Design and Design Education; 21st International Conference on Design Theory and Methodology, Parts A and B*, 2009, pp. 829–840.

[13] J. Patel and M. I. Campbell, “An Approach to Automate and Optimize Concept Generation of Sheet Metal Parts by Topological and Parametric Decoupling,” *J. Mech. Des.*, vol. 132, no. 5, p. 051001, 2010.

[14] A. Swantner and M. I. Campbell, “Automated Synthesis and Optimization of Gear Train Topologies,” in *Volume 5: 35th Design Automation Conference, Parts A and B*, 2009, pp. 13–22.

[15] M. I. Campbell, “GraphSynth.” 2011.

[16] W.Huang and M.I.Campbell, “Planar Mechanism Kinematic Simulater:3 Bar with 1 RP(1),” 2015. [Online]. Available: purl.org/pmks/MR1.

[17] W.Huang and M.I.Campbell, “Planar Mechanism Kinematic Simulater:3 Bar with 1 RP(2),” 2015. [Online]. Available: purl.org/pmks/MR2.

[18] W. Huang and M. I. Campbell, “Planar Mechanism Kinematic Simulater: 8 Bar with Rigid Structure,” 2015. [Online]. Available: purl.org/pmks/MR5.

[19] N. Rojas and F. Thomas, “On closed-form solutions to the position analysis of Baranov trusses,” *Mech. Mach. Theory*, vol. 50, pp. 179–196, Apr. 2012.

[20] T. Yang and F. Yao, “Topological characteristics and automatic generation of structural synthesis of planar mechanisms based on the ordered single-openedchains,” 1994, pp. 67–74.

[21] W.Huang and M.I.Campbell, “Planar Mechanism Kinematic Simulater:6 Bar with 4P.” [Online]. Available: purl.org/pmks/MR12.