The Dissertation Committee for Pradeep Radhakrishnan Certifies that this is the approved version of the following dissertation:

Automated Design of Planar Mechanisms

Committee:

Ashish Deshpande, Supervisor

Matthew I. Campbell, Co-Supervisor

Richard H. Crawford

S.V. Sreenivasan

Risto Miikkulainen

Automated Design of Planar Mechanisms

by

Pradeep Radhakrishnan, B.E.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2014

Dedication

To my family members

Acknowledgements

I would like to express my sincere gratitude to my advisor Dr. Matthew I. Campbell for his support, guidance and encouragement throughout this dissertation. I also would like to thank Dr. Ashish Deshpande, who not only agreed to fill-in for Dr. Campbell after his move to Oregon State University but also provided critical advice towards my goal during this time. I am also thankful to Dr. Richard H. Crawford, Dr. S.V.Sreenivasan and Dr.Risto Miikkulainen for agreeing to be a part of my dissertation committee and for providing valuable feedback during my defense.

During this time, I was a teaching assistant to Dr. Campbell, Dr. Don Berry and Dr. Michael Cullinan, who collectively provided a great experience and also supported me during critical phases by cutting me some slack from my teaching assignments. In addition, all my lab mates at the Automated Design Lab and the Manufacturing and Design Lab, other faculty and administrative staff in the department were of immense help during my doctoral program.

Finally, this dissertation came through because of the prayers and support of my family members. My friends in Austin were also of great help.

Automated Design of Planar Mechanisms

Pradeep Radhakrishnan, Ph.D.

The University of Texas at Austin, 2014

Supervisors: Ashish Deshpande, Matthew I. Campbell

The challenges in automating the design of planar mechanisms are tremendous especially in areas related to computational representation, kinematic analysis and synthesis of planar mechanisms. The challenge in computational representation relates to the development of a comprehensive methodology to completely define and manipulate the topologies of planar mechanisms while in kinematic analysis, the challenge is primarily in the development of generalized analysis routines to analyze different mechanism topologies. Combining the aforementioned challenges along with appropriate optimization algorithms to synthesize planar mechanisms for different user-defined applications presents the final challenge in the automated design of planar mechanisms. The methods presented in the literature demonstrate synthesis of standard four-bar and six-bar mechanisms with revolute and prismatic joints. But a detailed review of these methods point to the fact that they are not scalable when the topologies and the parameters of n-bar mechanisms are required to be simultaneously synthesized. Through this research, a comprehensive and scalable methodology for synthesizing different mechanism topologies and their parameters simultaneously is presented that overcomes the limitations in different challenge areas in the following ways. In representation, a graph-grammar based scheme for planar mechanisms is developed to completely describe the topology of a mechanism. Grammar rules are developed in conjunction with this

representation scheme to generate different mechanism topologies in a tree-search process. In analysis, a generic kinematic analysis routine is developed to automatically analyze one-degree of freedom mechanisms consisting of revolute and prismatic joints. Two implementations of kinematic analysis have been included. The first implementation involves the use of graphical methods for position and velocity analyses and the equation method for acceleration analysis for mechanisms with a four-bar loop. The second implementation involves the use of an optimization-based method that has been developed to handle position kinematics of indeterminate mechanisms while the velocity and acceleration analyses of such mechanisms are carried out by formulating appropriate linear equations. The representation and analysis schemes are integrated to parametrically synthesize different mechanism topologies using a hybrid implementation of Particle Swarm Optimization and Nelder-Mead simplex algorithm. The hybrid implementation is able to produce better results for the problems found in the literature using a four-bar mechanism with revolute joints as well as through other higher order mechanisms from the design space. The implementation has also been tested on three new challenge problems with satisfactory results subject to computational constraints. The difficulties in the search have been studied that indicates the reasons for the lack of solution repeatability. This dissertation concludes with a discussion of the results and future directions.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

The process of designing a planar mechanism can be broadly categorized into two stages. The first stage is identifying the type of application where the mechanism may trace a path (e.g.: conveyor mechanism), describe a motion (e.g.: opening and closing of a convertible roof-top in an automobile) or follow a function (e.g.: cam-follower mechanism for valve-opening in an engine). The second stage is synthesizing a mechanism to satisfy those design specifications. The preferred approach is to select a reference mechanism from handbooks such as [1] or textbooks such as [2–4]. The selection may also be augmented by the designer's experience in the related field. The selected concept is then modeled in a CAD package such as SolidWorks [5] Motion, Working Model [6], ADAMS [7], SAM [8] or WATT [9] and manually iterated to attain the solution. Packages such as SAM have built-in optimization routines based on gradient and evolutionary algorithms that can search the space for better solutions given the bounding box constraints for links and joints. This process of designing is time-consuming despite the existence of various mechanism atlases and references in handbooks as the magnitude of manual activity is high. The lengthy process discourages the designer from exploring many potential alternatives and leaves the user with only one or utmost two design concepts.

The vast amount of knowledge available in textbooks and handbooks can be harnessed into a database for mechanism concepts that can be used to automatically (i.e., computationally) generate planar mechanisms. Though there has been research on creating such repositories, there has not been much research into developing design rules that can be used to automatically synthesize planar mechanisms. Also, most of the research on automated synthesis has been restricted to solving four-bar and six-bar mechanisms for certain path tracing problems. With the exception of WATT, which can generate four-bar and six-bar mechanisms with revolute joints, there is no tool available currently that can simultaneously generate planar mechanism concepts and optimize them

for any user-defined application. The reasons for the non-availability of a complete tool for planar mechanism design may be attributed to the absence of a standardized repository of design rules, a generic kinematic analysis tool and generic and powerful optimization algorithms that can be employed for synthesis. This research seeks to fill these vacancies.

1.1 STATEMENT OF RESEARCH

A generalized methodology for automatically synthesizing planar mechanisms is developed by incorporating

a. a simplified and comprehensive knowledge representation scheme,

b. a generic kinematic analysis tool that can analyze any single-degree of freedom mechanism, and

c. generative search and optimization algorithms to simultaneously synthesize topology and parameters

such that multiple valid one-degree of freedom mechanisms are generated for any problem.

The objective of this research is to demonstrate the ability to computationally generate planar mechanism designs and optimize those mechanisms to satisfy user specifications. The planar mechanisms considered in this research are composed of revolute (R), and prismatic (P) joints. Through a simplified but comprehensive knowledge representation scheme, design rules are developed in this research that describe the design space of valid one-degree of freedom planar mechanisms. These designs are optimized using different algorithms to determine their compatibility for user specifications, the data for which is obtained from an integrated kinematic analysis routine. Those mechanisms that conform to specifications are presented to the user. The key element in this research is to develop a generic methodology for automated design of planar mechanisms. Making a process generic not only elevates the design complexity but also helps in understanding the

limitations of such processes and tools and their applicability to actual design practice. In addition, we feel that this approach will result in designs being generated rapidly to complement the manual approach that a designer normally adopts. The major topics of research in this dissertation are representation of knowledge using graph-grammar methodology, generalization of kinematic analysis and integration and development of algorithms for search and optimization of single and multiple design objectives.

1.2 ORGANIZATION OF DISSERTATION

The dissertation is organized as follows. Chapter 2 will present a brief review of the related work. This will be followed by an illustration of the plan of work and the problems that will be tackled in Chapter 3. The graph grammar based representation scheme developed for planar mechanisms and the grammar rules used in the generation process are presented in Chapter 4. The generated mechanisms are kinematically analyzed and Chapter 5 presents our generalized implementation of geometrical methods for position and velocity analyses and analytical equation method for acceleration analysis for determinate mechanisms. In the same chapter, an optimization based method for position analysis of indeterminate mechanisms is presented. Chapter 6 will present details on the optimization algorithm used in this research as well as the overall implementation pseudo code. The results obtained using our implementation is presented in Chapter 7 followed by a discussion on the same in Chapter 8. Concluding remarks will be presented in Chapter 9 where future activities are included.

Chapter 2: Related Work

The related work on the three aspects of automated design namely knowledge representation, kinematic analysis and optimization will be presented in the same order in this chapter.

2.1 KNOWLEDGE REPRESENTATION

The synthesis of planar mechanisms was aided largely due to the formal structure proposed by Freudenstein and Maki [10], who classified different kinematic entities based on structure and function that also included graph representation of different kinematic structures. Their work popularized the application of graph theory in this domain and hence, this section will focus only on work related to graph theory. This idea was further developed by Tsai (as explained in his book in [11] published as a collection of his work) and others as the graph based Systematic method where the nodes in a graph represent the links in the mechanism and the edges (arcs) represent the type of joint existing between the two links. This representation is very popular among researchers in the automated type synthesis of planar mechanisms. The graph representation is then transformed to an adjacency matrix formulation for computational purposes. A few papers that use the systematic method for enumerating planar mechanisms are listed in [12–14]. Mruthyunjaya's review paper on kinematic structures [15] also provides a detailed overview of the other papers that make use of this technique. The graph representation was further developed by Sohn and Freudenstein [16] where they used dual graphs to represent planar mechanisms for automated generation of one-, two- and three-degree of freedom mechanisms. In all these works, the focus has primarily been on representing planar mechanisms consisting of revolute joints, though there have research such as [12], [17] that have attempted to incorporate prismatic joint types. The graph representation based on the systematic method deals only with structural outline and does

not contain any information regarding input and ground links, which are usually bookmarked during later stages in design.

The graph representation is then used to generate the planar mechanisms that are part of the space of valid designs. Though there are several techniques available in the literature, only a few commonly used methods will be detailed here. Mayourian and Freudenstein [18] employed the restrictions in graph representation and actual design to generate a set of mechanisms (2D and 3D) with up to six links, that can serve as the basis for conceptualizing mechanisms and automating their synthesis for different applications. Another technique to identify the space of mechanisms is through number synthesis. In this technique, the Gruebler's Criterion is used, which determines the degree of freedom (M) of a planar mechanism through the equation $M=3(n-1)-2*f_1-f_2$, where n denotes the number of links, f_1 is the number of one-degree of freedom joints and f_2 is the number of two-degree of freedom joints. Depending on the required degree of freedom, the values of n, f_1 and f_2 can be varied to locate a mechanism. Once the number of links and joints are obtained, the assembly configuration of the mechanisms is determined and there may be multiple possibilities for the same configuration. In order to enumerate all possible configurations, exhaustive search techniques are employed. Some of these configurations are available in textbooks ([3,4]) and also in publications such as [15,19] where the number of unique one-degree of freedom mechanisms with revolute joints possible for mechanisms up to 14-links are listed. Enumerating mechanisms with more than one-degree of freedom is complex and is therefore not widely reported in the literature and hence not considered in this dissertation.

There are also techniques based on the addition of Assur group members (as illustrated in [20,21]) to generate multi-link planar mechanisms, where different structural elements are attached to a base mechanism in such a way that the degree of freedom of the planar mechanism remains unaffected. This technique is also employed only on planar mechanisms with revolute joints to generate mechanisms with one-, two-, and three-degrees of freedom. In all the techniques mentioned above, there are

possibilities for the presence of structurally equivalent candidates (isomorphic) and there are techniques illustrated within those publications as to how isomorphism is detected and such candidates eliminated. Another technique for generating planar mechanisms with fewer isomorphic candidates is presented by Rao in [22,23].

2.2 KINEMATIC ANALYSIS

Automated synthesis of planar mechanisms requires a reliable kinematic analysis tool, which may be based on graphical techniques or the analytical loop equation solution procedure as illustrated in various texts on kinematics. Some of the research projects on automated synthesis such as [24–26] base their analysis on the analytical loop equation method and Newton-Raphson solution technique [27]. There are other projects that propose reduction of complex mechanisms into Assur groups and then use the analytical loop equation technique for obtaining solutions. A careful examination of the different research projects shows that the focus is limited to standard four- or six-bar mechanisms with revolute joints and 1-degree of freedom since kinematic solutions are already available for such mechanisms. One of the reasons for not exploring mechanisms with more number of links (like eight or ten-bars) and joints such as prismatic and pin-in-slot is due to the absence of analysis tools that can handle generic topologies in a reliable manner. Though there are commercial tools such as ADAMS, Working Model, etc. available, they do not include any application-programming interface (API) that can be used in conjunction with the graph representation so that as the topologies are generated, they can be automatically analyzed in order to achieve automation in actual sense.

In addition to the non-existence of a generic tool for kinematic analysis, the graphical and analytical techniques available in the literature have certain limitations in their solution procedure. The graphical techniques such as the instantaneous center of rotation method for velocity (based on Kennedy-Aronholdt theorem [2]) and the dyadic decomposition technique for position analyses require a four-bar loop (or a dyadic configuration) for determining solutions. Due to this requirement, mechanisms such as

the double butterfly linkage [28], also known as an indeterminate mechanism, cannot be analyzed using the graphical technique. The mechanism also cannot be analyzed by applying the Newton-Raphson method to the analytical loop equations since that method does not generate reliable solutions for mechanisms with higher-order loops as there are higher numbers of unknowns in as many non-linear equations. This constraint in existing techniques is rarely highlighted in standard textbooks on kinematics [29]. In order to solve such mechanisms, Sreenivasan and Waldron [29] and Sommese et al. [30] provide numerical solutions for the loop equations using the polynomial continuation method (more details about polynomial continuation in Morgan [31]). There is an alternative method suggested by Wampler [32] which is, in turn, based on the method suggested by Nielson and Roth [33] where the Dixon determinant [34] is used but differs in the implementation. The difference in the approach of Wampler [32] and Nielson and Roth [33] is that equations are formulated directly in the complex plane in the Wampler approach as against the sine and cosine formulation in the Nielson and Roth approach. There are also methods based on elimination techniques for solving analytical position loop equations of planar mechanisms.

On the graphical methods for position analysis, there is a geometric iterative method [35] that claims to offer an alternate approach to solving the position problem. This method begins with the orientation of the input link, followed by a random positioning of one of the links connected to the input link, following which dyadic decomposition of the remaining links is carried out based on a set of rules listed in the article. Once all the pivots are assigned, correction (iterative process) is initiated and continued till the appropriate convergence is achieved. On the graphical velocity analysis, Foster and Pennock [36] suggest a graphical method to solve for two arbitrary secondary instant centers in a double butterfly linkage, which allows for determining other instant centers using the Kennedy-Aronholdt theorem. This method is based on an iterative technique described by Klein [28]. There are also other techniques for kinematic analysis proposed by Gea et al. [37] and Chen et al. [38] that are based on the minimum potential

energy and constraint superposition principles respectively. There is also another optimization-based technique proposed by Porta et al. [39] for linkage analysis. Though these new methods are available for solving the position and instant centers, there are no formal generalized implementations of these techniques available, thus making it difficult to analyze the reliability of these methods or their applicability for a wider class of problems.

2.3 OPTIMIZATION

Optimization has been used for many decades for dimensional synthesis of planar mechanisms. To effectively use optimization, there are several inputs required. The inputs define the problem type (i.e., path, motion or function) and the related design specifications such as the topology of the mechanism (four-bar or six-bar), the location of inputs and ground, the output joint or link, the overall size of the mechanism (bounding box) and so on. In addition, an objective function is formulated based on the input design specifications, which is minimized by the algorithm. Cossalter et al. [25] provide a brief review of the related work in this area (such as the least squares approach, penalty functions, selective precision synthesis and stochastic formulations) in addition to their work on using a quasi-Newton non-derivative optimization approach for synthesizing planar mechanisms. In the paper, the authors explain the formulation of an objective function, which is based on the sum of the squares of distance between the desired and actual points, and the use of a weighted scheme to distinguish between path, function and motion problems. The authors demonstrate their method and its efficiency for different problems using four and six-bar mechanisms. Alizade et al. [40]were one of the first to demonstrate the use of penalty functions along with inequality constraints for optimizing function-generating four-bar mechanisms. Sancibrian et al. [41] proposed an alternate formulation of the objective function consisting of kinematic, synthesis and assembly constraints that aids in analytically taking the derivative for use in a search algorithm.

The authors demonstrate their formulation using four- and six-bar mechanisms (Stephenson-III and Watt II) for path, motion and function type mechanisms synthesis.

There has also been a lot of research employing evolutionary algorithms for the synthesis of planar mechanisms. Cabrera et al. [24] proposed a genetic algorithm based synthesis of planar mechanisms where in benchmark path problems are synthesized using different planar mechanisms consisting of revolute joints. In another work Cabrera [42] proposed a multi-objective framework using a new algorithm whose basis is genetic algorithm and demonstrated the algorithm's effectiveness in the design of robotic hand grippers consisting of revolute and prismatic joints. Sedlaczek et al. [17] too demonstrated the synthesis of 1-degree of freedom planar mechanisms with revolute and prismatic joints for path-time problems using a genetic algorithm formulation. The authors presented a comprehensive structure that also included knowledge representation and kinematic analysis and were able to generate solutions in times varying from 17 minutes to 23 hours using genetic algorithms. While the previous works were based off genetic algorithm, Archarya and Mandal [43] estimated the performance of different evolutionary algorithms namely genetic algorithms, Particle Swarm Optimization and differential evolution for synthesis of planar mechanisms. Basing their evaluation criterion on the least error between desired and actual paths, the authors found that differential evolution algorithm produces the best result among the algorithms tested. Some of the other works on the synthesis of planar mechanisms are listed in [44–49].

While most papers adopt similar objective function formulations and problems, the major constraints as mentioned by Cossalter et al. [25] are related to kinematic analysis and the actual mechanical assembly of those mechanisms. Also, the methods developed thus far are not generic to any kind of synthesis problem [41] as the same benchmark problems or simpler mechanisms have only been evaluated in most cases.

2.4 CONCLUSION

The review of literature proves that there is still significant room for improvement in all the areas that can be used to create a totally automated conceptual design tool for planar mechanisms.

Chapter 3: Methodology

The methodology followed in this dissertation is summarized in Figure 3-1 where the limitations and challenges in current literature will be explored further in two phases. The first phase ((a) in Figure 3-1) focuses on the development of a comprehensive representation scheme as well as the generalization of kinematic analysis routines. Under representation, a rich graph-grammar based scheme will be presented using which building block rules will also be developed that can be used to generate planar mechanisms. The goals in knowledge representation are: 1. Develop a generic planar mechanism representation scheme that can be used to represent different joints and link types within the same scheme 2. Use the scheme to generate the largest set of valid designs, in this case 1-degree of freedom planar mechanisms with the least number of invalid designs using minimum number of rules. At this juncture, revolute (R), prismatic (P) and pin-in-slot (R-P) type joints will be considered with the possible extension to gear representations. The kinematic analysis routine involves development of generalized analysis schemes based on the methods available in the literature. In addition, alternate methods for solving kinematics of indeterminate mechanisms will be explored. The goal here is to create a generic and robust kinematic analysis tool that is also computationally efficient to be used in the automated conceptual design of planar mechanisms.

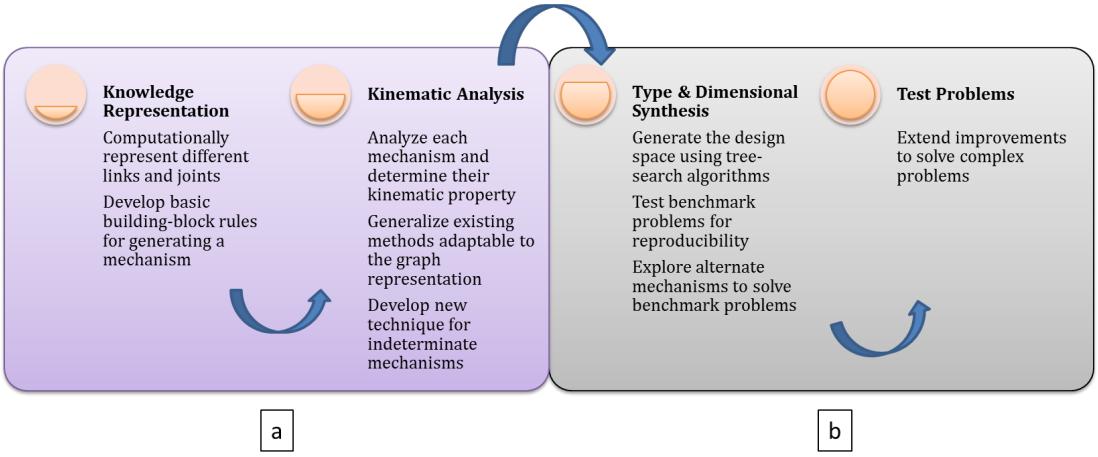


Figure 3-1: (a) Phase I: Generalizing Knowledge Representation and Kinematic Analysis (b) Phase II: Type Synthesis and Dimensional Synthesis and Test Problems

The second phase ((b) in Figure 3-1) will focus on type and dimensional syntheses. During this phase, the grammar rules developed in the first phase are combined to generate different planar mechanisms with an exhaustive tree-search algorithm. The focus is to test whether the grammar rules are able to generate all the one-degree of freedom mechanisms as reported in the literature [15,19] for revolute joints. In addition, the set of valid planar mechanisms consisting of prismatic and pin-in-slot joints is also enumerated. These mechanisms will then be synthesized for different problems in the literature (referred to as benchmark problems in this report) using appropriate optimization algorithms during which the kinematic analysis routines developed in the first phase will be used to guide the optimization process. Four bar mechanisms with revolute joints will be synthesized to solve the benchmark problems (listed in Table 3-1). In addition, the design space will be explored to solve benchmark problems with higher order mechanisms. Once the benchmark problems have been solved and the capability of our implementation proved, three additional challenge problems are tested using our approach for further evaluation and discussion. The second phase amalgamates our

research in knowledge representation, kinematic analysis and search and optimization into a complete tool.

The ultimate goal is to develop an automated design tool that follows the flow chart presented in Figure 3-2. As shown in the figure, the design tool requires specification of the problem in terms of either the path to be traced or the motion to be followed. After this, using grammar rules, the set of all possible mechanisms are generated (based on (a) in Figure 3-1). From this input (also called the design space), each mechanism is synthesized for obtaining the specifications set by the user using appropriate optimization algorithms. At this stage, the goal is to synthesize as many mechanisms as possible that can satisfy the requirements set by the user. The mechanisms presented to the user may be different variations of the same mechanism as in different link lengths in a four-bar mechanism or completely different mechanism topologies such as a four-bar mechanism with sliding members and a six-bar mechanism with revolute and prismatic joints. This research is carried out using GraphSynth software [50] developed by Campbell [51].

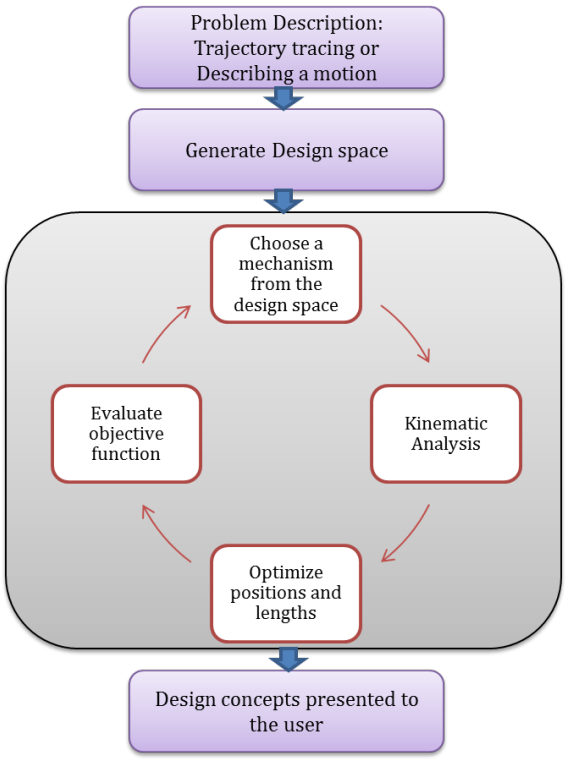


Figure 3-2: Overall flow depicting the automated design of planar mechanisms

3.1 BENCHMARK PROBLEMS SOLVED IN THIS RESEARCH

The benchmark problems used to test our implementation are listed in Table 3-1 where each problem is described in terms of the coordinates of the path traced by the mechanism. Two types of problems are considered namely path and path-time where the path is dependent on the input crank angle. The table also lists the source of each problem along with the least error obtained in the literature for that problem (Note that the least error is not necessarily obtained in the original paper). The error displayed in the rightmost column is defined in terms of the sum of squares of the distances between the synthesized set of coordinates and the original defined by the user unless otherwise noted (in parenthesis for a couple of problems).

Table 3-1: Benchmark problems for path synthesis

Problem Description	Source	Least Error
(20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45)	Cabrera et al. [52,53]	0.0002
(20, 10), (17.66, 15.142), (11.736, 17.878), (5, 16.928), (0.60307, 12.736), (0.60307, 7.2638), (5, 3.0718), (11.736, 2.1215), (17.66, 4.8577), (20, 10)	Cabrera et al. [52,53]	0.0047
(-24, 40), (-30, 41), (-34, 40), (-38, 36), (-36, 30), (-28, 29), (-21, 31), (-17, 32), (-8, 34), (3, 37), (10, 41), (17, 41), (26, 39), (28, 33), (29, 26), (26, 23), (17, 23), (11, 24), (6, 27), (0, 31)	Hongying et al. [54]	0.906 (average distance error)
(-27,1), (-21.857, -3.214), (-16.7, -7.428), (-6.428, -15.857), (-1.285, -20.071), (3.857, -24.285), (9, -28.5), (15, -29.9), (20, -30), (27.2, -25), (29.2, -20), (28, -10), (22.7, 2), (15, 10.6), (5, 16.5), (-10, 19.6), (-22, 17), (-28, 11), (-29, 5)	Hongying et al. [54]	0.4154 (average distance error)
(5, 0), (4.9240, 0.8682), (4.6985, 1.7101), (4.3301, 2.500), (3.8302, 3.2139), (3.2129, 3.8302), (2.5, 4.3301), (1.7101, 4.6985), (0.8682, 4.9240), (0, 5), (-0.8682, 4.9240), (-1.7101, 4.6985), (-2.5, 4.3301)	Matekar and Gogate [55]	0.0154
Path: (0, 0), (1.9098, 5.8779), (6.9098, 9.5106), (13.09, 9.5106), (18.09, 5.877), (20, 0) Time: (π / 6, π / 3, π / 2, 2 π / 3, 5 π / 6, π)	Acharyya and Mandal [56]	1.2162
Path: (0.5, 1.1), (0.4, 1.1), (0.3, 1.1), (0.2, 1.0), (0.1, 0.9), (0.005, 0.75), (0.02, 0.6), (0.0, 0.5), (0.0, 0.4), (0.03, 0.3), (0.1, 0.25), (0.15, 0.2), (0.2, 0.3), (0.3, 0.4), (0.4, 0.5), (0.5, 0.7), (0.6, 0.9), (0.6, 1.0) Time (in °): (0, 21, 42, 63, 84, 105, 126, 147, 168, 189, 210, 231, 252, 273, 294, 315, 336, 357)	Kunjur and Krishnamoorthy [57]	0.0196
x(t)=3 cos(t), y(t)=2 sin(t), t is time	Sedlaczek et al. [17]	0.1298
x(t)=-cos(t)*(0.5+cos(t)), y(t)=- sin(t)(0.5_cos(t)), t is time		8.055 E-5

Table 3-1 continued.

Problem Description	Source	Least Error
x(t)=0.5*(2*sin(t)-sin(2t)), y(t)=0.5 *(2*cos(t)+cos(2t)), t is time	Sedlaczek et al. [17]	1.139

3.2 CHALLENGE PROBLEMS SOLVED IN THIS RESEARCH

Once the benchmark problems have been solved and alternate mechanisms explored for those applications, the final step is to extend the algorithms to solve three challenge problems. These problems are representative of the complexity in an actual design setting and will test the algorithms and implementations that have been devised for benchmark problems. The first problem (shown in Figure 3-3) from [3] is a path synthesis problem that is part of a conveyor system. As shown in the figure, the mechanism used for this problem is a four-bar mechanism (O_2APBO_4) to which an Assur group (consisting of links 5,6,7,8) is connected. The specifications for the problem such as the bounding box for housing the mechanism and the path details are given in Table 3-2. The four-bar mechanism in the figure below is one of the expected outcomes while alternate mechanisms will also be explored from the design space of valid mechanisms.

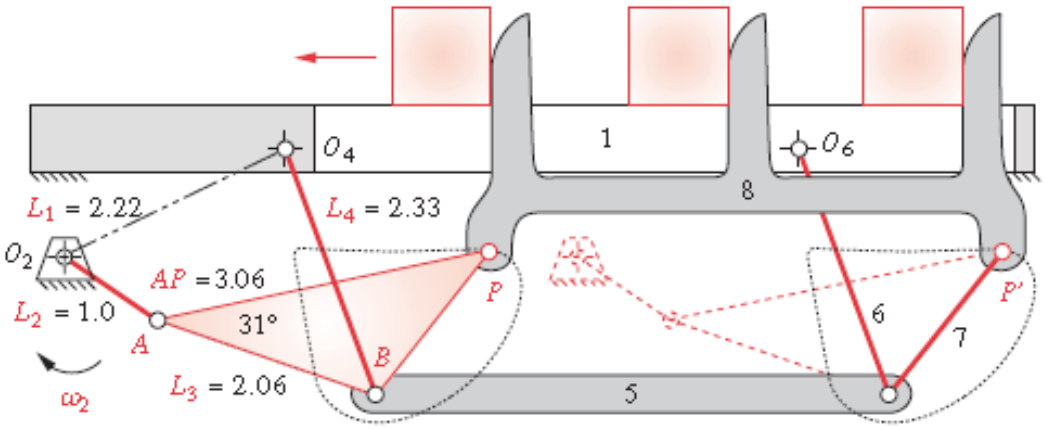


Figure 3-3: Challenge problem #1: Conveyor Mechanism from Norton [3]

Table 3-2: Coordinates of the pivot “P” from Figure 3-3.

S. No.	X	Y	S. No.	X	Y
1	4.0223	-0.479	19	2.0598	0.0328
2	3.9031	-0.7466	20	2.0655	0.0692
3	3.6805	-1.0186	21	2.0898	0.0887
4	3.3864	-1.2353	22	2.1357	0.0942
5	3.0816	-1.3577	23	2.2051	0.0892
6	2.8189	-1.3853	24	2.2985	0.078
7	2.6197	-1.3409	25	2.4151	0.0646
8	2.48	-1.2495	26	2.5526	0.0527
9	2.3858	-1.1295	27	2.7078	0.0451
10	2.3219	-0.993	28	2.8767	0.0432
11	2.2763	-0.8486	29	3.0545	0.0469
12	2.2402	-0.7022	30	3.2362	0.0541
13	2.2079	-0.5591	31	3.4164	0.0603
14	2.1766	-0.4234	32	3.5896	0.0586
15	2.1454	-0.2991	33	3.7491	0.0391
16	2.1153	-0.1893	34	3.8867	-0.0108
17	2.0887	-0.0965	35	3.9906	-0.106
18	2.0689	-0.0222	36	4.0436	-0.2603
Bounding Box: (Max Width: 15, Max Height: 15)					

The second challenge problem is a bio-mimicking problem where the motion of a coconut crab’s legs ((reference video is given in [58]) is replicated using a planar mechanism that traces the trajectories of different joints. A snapshot of the coconut crab is shown in Figure 3-4 where the joints considered for mimicking are highlighted in the figure using arrows in red. The trajectories traced by the joints of the rear leg of the coconut crab are listed in Table 3-3. This problem unlike traditional path and path-time problems requires identification of mechanisms where the joints are lined up as in the coconut crab in order to exactly the mimic the particular leg in the animal.

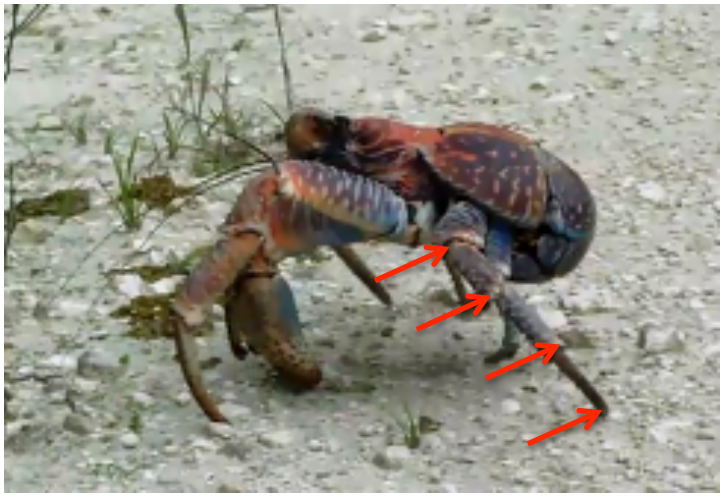


Figure 3-4: A snapshot of the coconut crab

Table 3-3: Coordinates of different joints in terms of absolute reference

Joint 1		Joint 2		Joint 3		Joint 4	
X	Y	X	Y	X	Y	X	Y
155.25	-77.25	156	-89.25	155.25	-97.5	149.25	-100.5
151.5	-84	153.75	-94.5	157.5	-102	153	-109.5
139.5	-86.25	140.25	-95.25	140.25	-103.5	132.75	-106.5
127.5	-88.5	131.25	-98.25	132	-105.75	124.5	-110.25
138	-85.5	138.75	-95.25	141.75	-104.25	134.25	-108.75

The third problem is to develop a mechanism to trace the trajectory shown in Figure 3-5, which is the logo of the University of Texas at Austin. The coordinates for the curve are listed in Table 3-4. This trajectory is complex and may be traced by either a single mechanism or using multiple planar mechanisms. The trajectories in challenge problems 2 and 3 are examples of problems where multi-objective optimization scenarios are explored. These example problems demonstrate the level of complexity that can be handled through algorithms in an automated design scenario and the eventual goal is to prove that a tool for automated synthesis of planar mechanisms is capable of generating useful design suggestions.

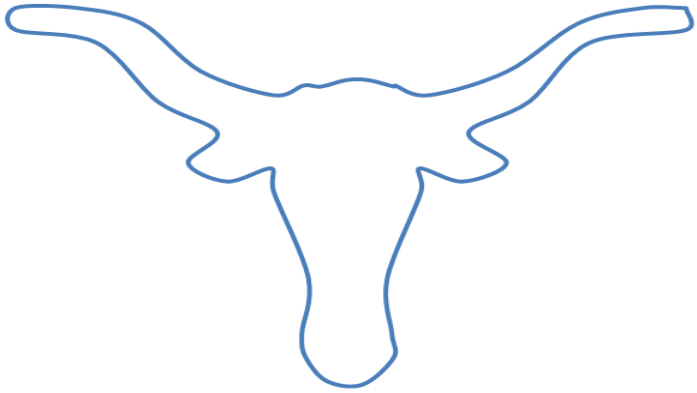


Figure 3-5: Logo of the University of Texas at Austin (also called “Longhorn”)

Table 3-4: Coordinates of the “Longhorn”

S. No.	X	Y	S. No.	X	Y
1	10	10	21	523	152
2	55	10	22	483	169
3	121	24	23	437	157
4	194	68	24	435	178
5	275	90	25	399	258
6	310	81	26	406	309
7	311	82	27	402	330
8	338	76	28	380	352
9	360	76	29	343	355
10	386	82	30	311	330
11	404	81	31	313	309
12	435	90	32	318	258
13	510	68	33	283	178
14	571	24	34	285	157
15	637	10	35	240	169
16	701	10	36	195	152
17	701	30	37	234	124
18	617	42	38	172	96
19	554	96	39	110	42
20	493	124	40	10	30
Bounding Box: (Max Width: 750, Max Height: 750)					

3.3 CONCLUSION

A detailed research plan is presented in this chapter that focuses on creating a graph-grammar based representation and rules system, developing a generic kinematic analysis tool and implementing an optimization algorithm that aids in the generation of different mechanism designs. The benchmark problems and the challenge problems that are used to test the proposed implementation are also listed.

Chapter 4: Representation

Computational tools to automatically synthesize planar mechanisms are explored as a way to overcome the difficulties and complexities of creating mechanisms manually. A powerful yet simple approach is to employ the concept of generative grammars. The review presented in Chapter 2 presents the limitations in existing representation schemes. The graph-based approach presented here builds on the traditional approaches but implements a novel representation scheme. The method represents links and pivots using nodes and the relationship between them using arcs. Labels and variables are used so that the graphs can be used for generation and evaluation of planar mechanisms. The presented scheme is generic and is able to represent different joints and link types. Based on this representation, grammar rules are developed so that topologies can be generated on the fly using a tree-search process starting from an initial seed graph. This chapter presents a detailed overview of the representation scheme and the grammar rules used in the overall search process. The next section 4.1 will explain need for a better representation scheme followed by section 4.2 where details on the graph-grammar based representation for planar mechanisms are presented. The grammar rules that are used to generate different planar mechanisms are explained in section 4.3, which will be followed by the generation of mechanism topologies using a search process in section 4.4. Section 4.5 will discuss the limitations and issues such as isomorphism and confluence followed by concluding remarks in section 4.6.

4.1 Need for Better Representation

Graph based schemes have been popular in representing planar mechanisms for synthesis and enumeration of topologies. The traditional graph based schemes, as explained in the related work (refer Chapter 2, section 2.1), are extensive but do not consist of all relevant information such as information on grounded joints, inputs, the output joint, etc. that completely describes a mechanism. Instead, they require tedious

bookmarking during runtime. The lack of a succinct but rich representation scheme is one of the reasons that research in the automated design of planar mechanisms is restricted to a few topologies such as four-bar and six-bar mechanisms. This is without considering the limitations in kinematic analysis.

The graph-grammar approach on the other hand helps in formulating a more generalized scheme through the use of descriptive labels that is currently unavailable in the existing approaches. This representation along with grammar rules help in formulating a generative design scheme that is akin to the natural design process than what is possible in the traditional graph approaches. That is, in traditional graph based formulations, the designer or the user would not be able to relate to the designs being generated since they do not contain any information such as grounded links and the type of joints until the post processing stage while in the approach presented here, the grammar completely defines the topology at every stage in the topology generation process. Not only this, the ability to present a descriptive graph will be of great advantage to the design community rather than a just a node-edge representation. In addition, a comprehensive information-rich representation scheme helps in the formulating better design automation approaches as will be shown in this dissertation. The representation scheme presented here has been developed through the use of GraphSynth [50], which is a graph-grammar manipulation tool developed by Prof. Matthew I Campbell.

4.2 BASIC REPRESENTATION

Two different representation schemes and grammar rules have been developed during this dissertation. The initial representation scheme and set of grammar rules are detailed in [59]. Though the underlying principle behind the representation scheme presented here remains the same, there are several changes in the grammar (referred to as labels) usage in order to increase the degree of generalization. Also, since a few constraints in kinematic analysis (will be explained in Chapter 5) and parallel computation were encountered, the grammar rules have certain changes from the first set

illustrated in [59]. Hence, only the latest version of the representation scheme and grammar rules will be described in this chapter.

The improved representation scheme developed is illustrated by means of an example four-bar mechanism shown in Figure 4-1. The graph-grammar representation for this four-bar mechanism is shown in Figure 4-2.

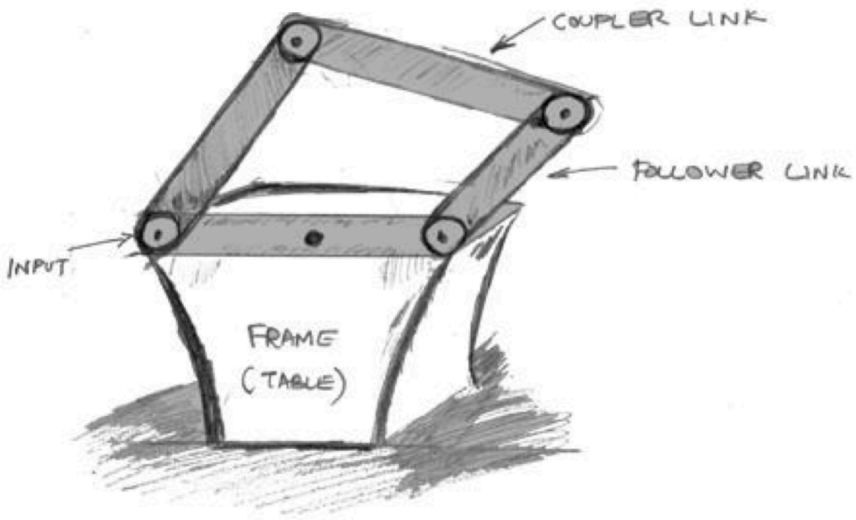


Figure 4-1: An illustration of a four bar mechanism

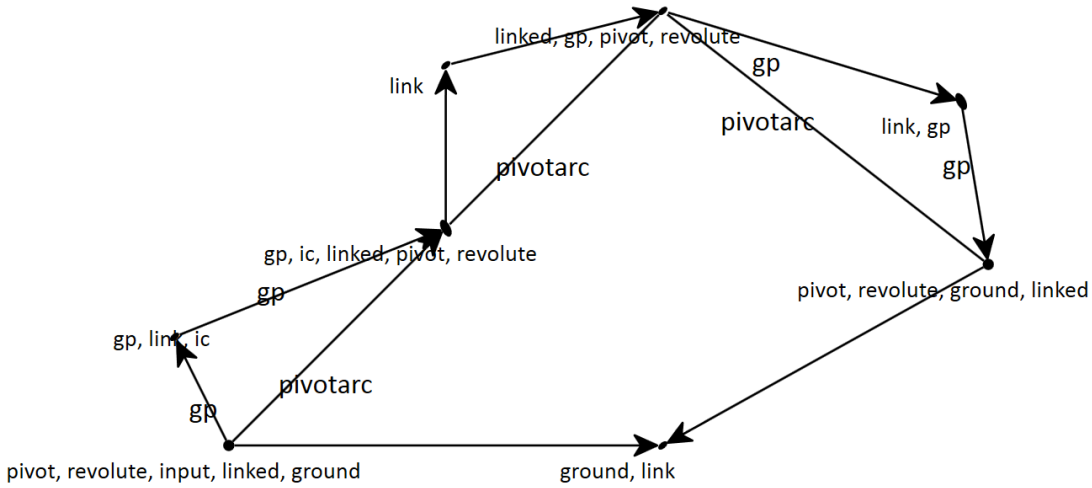


Figure 4-2: Graph-grammar representation of the four-bar mechanism shown in Figure 4-1

As shown in Figure 4-2 and comparing that to Figure 4-1, links and pivots are represented using nodes. The nodes are identified by small black dots in Figure 4-2. Arcs connect pivot and link nodes to create the mechanism. Every node is identified by a name but in the figure above the node names are not shown and only the labels are shown. Node names are used only as placeholders whereas labels are indicative of the function of the node. In Figure 4-2, there are different labels (listed in Table 4-1) associated with every node. On a closer observation of the graph and labels, it can be seen that the node with "ground" and "link" labels is used to represent the ground link or the global frame (refer Figure 4-1) in the planar mechanism. There are two pivots attached to the ground link. This information (as to which pivots are attached to the ground) can be ascertained using "pivot" and "ground" labels. The pivot on the left side of the graph has a label called "input" that indicates that the input is connected at this joint location. The "revolute" label is used to indicate the presence of revolute joints (R) at the concerned pivot nodes. You may notice that this graph has only revolute joints due to the presence of "revolute" label at each of the "pivot" nodes (also corresponds to Figure 4-1). The

“input” pivot is connected to another pivot with labels “gp,ic,linked,pivot,revolute” through a link with labels “gp,link”. This joint corresponds to the joint between the input link and the coupler link in Figure 4-1. The node with “link” label is used to represent a link and additional labels such “gp” are used to indicate that there is grounded pivot at one of the ends of that link. This information is useful during the formulation of grammar rules. For instance, labels such as “gp” and “ic” are part of both link and pivot nodes. If they are part of the link node, then there is a grounded joint or an input or both connected to one end of that concerned link. The same meaning carries over if these labels are part of a pivot node (“gp” is short form for grounded pivot and “ic” for input connected). Note that the “link” nodes contain labels such as “gp” and “ic” while the arcs that connect such nodes contain the “gp” label.

The node with labels “gp,ic,linked,pivot,revolute” is connected to another pivot node with labels “linked,gp,pivot,revolute” through a link node with label “link”. This corresponds to the joint between the coupler link and the follower link in Figure 4-1. The label “linked” is used to indicate whether a particular pivot is connected to another link or not. The labels “gp”, “linked” and “ic” are used to formulate better grammar rules and reduce the search space so that a concise set of mechanism topologies are produced. An arc with label “pivotarc” connects the pivots. This is helpful during kinematic analysis to calculate the distance between two pivots. The arrowhead on each arc is used to reduce the list of applicable options during the generation process in order to reduce confluent recognition options in GraphSynth. Table 4-1 below gives a summary of the list of the labels used to define a generic four-bar mechanism. This list is applicable for all mechanisms that have joints with similar characteristics.

Table 4-1: Details of the graph-grammar representation used in a four-bar mechanism with revolute joints

Link / Pivot (ref Figure 4-2)	Grammar Representation (Node/Arc)	Labels Used
Ground Link	Node	ground,link
Input Link	Node	gp,link, ic
	Arcs	gp
Coupler Link	Node	link
Follower Link	Node	gp,link
	Arcs	gp,link
Input Joint	Node	pivot,revolute,input,linked,ground
Joint between Input and Coupler links	Node	pivot,revolute,linked,gp,ic
Joint between Coupler and Input Links	Node	pivot,revolute,linked,gp
Ground Joint (between Follower and Ground links)	Node	pivot,revolute,linked,ground
Arcs between Pivots	Arcs	pivotarc

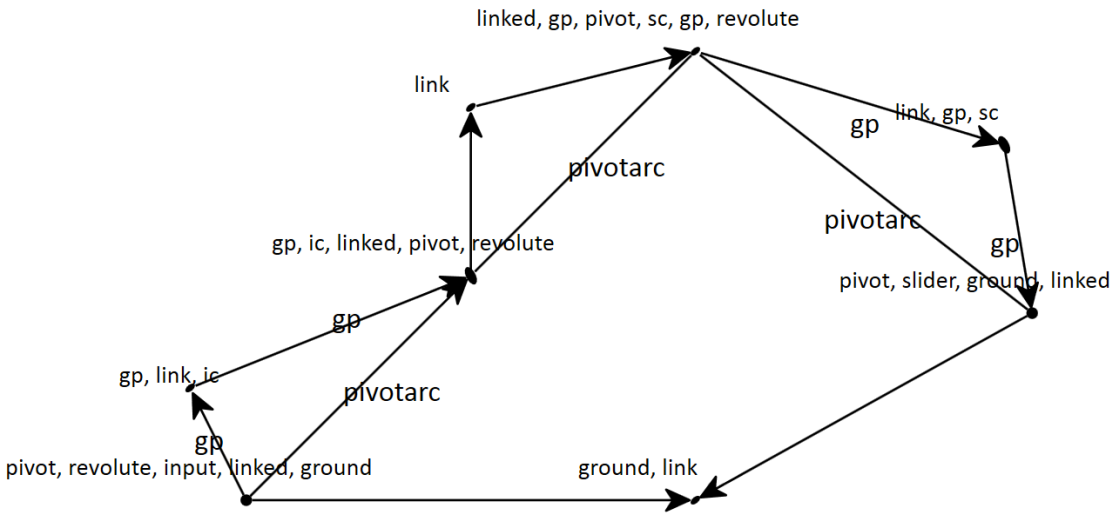


Figure 4-3: Graph-grammar representation for a slider-crank mechanism

Figure 4-3 (shown above) presents the graph-grammar representation for a slider crank mechanism. It can be seen from the figure that the representation is similar to that of the four-bar shown in Figure 4-2 except that there are changes in two pivot nodes and one link node. The pivot node that is connected to the ground link (with labels "ground" and "link" on the right side) has the label "slider" instead of "revolute" and the corresponding follower link has an additional label "sc" indicating the presence of "slider" at one of its end. The pivot between the coupler and the follower link also has an additional label "sc" indicating that the presence of a slider at the other end of the link with label "sc". In this representation scheme, the input is set to a revolute joint since the kinematic analysis is robust for such mechanisms. But it is also possible to have the input to be a sliding joint (P). The sliding angle is represented using the "variable" feature for nodes in the GraphSynth tool. More details about the software are available in [51]. The representation scheme described here can be expanded to include other elements such as gears and cams, thereby generating a diverse set of topologies within the same framework.

4.3 GRAMMAR RULES

Using the representation scheme, grammar rules are developed to generate different mechanism topologies. The rules have been developed by reviewing earlier iterations [59] and through experiences in other graph-grammar based research. A building block methodology is adopted in formulation of these rules where new mechanisms are created by adding links and pivots to existing graphs (topologies). This aspect will be evident explanation of rules in the following sections.

4.3.1 Grammar Rule Formulation and Identification

Grammar rules are integral to the design generation process and GraphSynth is used to develop and test these rules. A typical search process that will be followed in this dissertation is shown in Figure 4-4, where the process begins with a seed graph and

grammar rules will be successively applied to create candidate graphs or just called "candidates" at every level. This step corresponds to the "Design Space Generation" module in Figure 3-2. The "candidates" at every level in the tree are potentially different mechanism topologies. Though there are topologies with higher degrees of freedom in the search tree, only one-degree of freedom are extracted for synthesis purposes. The secondary candidates in turn form seeds at the respective levels to generate candidates further down in the tree. Due to this process flow, the seed is an important parameter since it influences the character of the rules being formulated and also the degree of generalization for the entire process. The seed used in this work is shown is Figure 4-5.

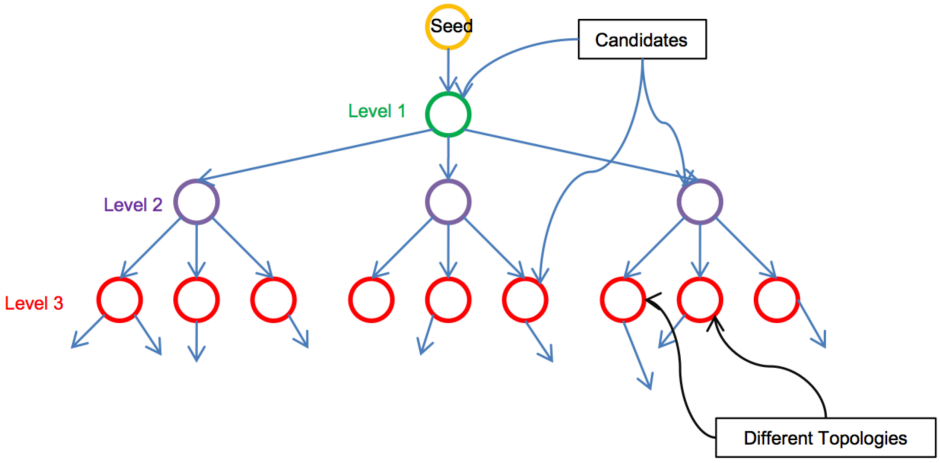


Figure 4-4: An illustration of the tree-search process using seed and grammar rules

Figure 4-5: The starting seed graph used in the tree-search process

The above figure shows that the seed has two nodes; one is a ground link defined by labels “ground, link” and the other is a revolute joint and an input defined by labels “input pivot revolute, ground”. This indicates that the process begins with the knowledge of the type of input and the reference frame. The position of the joints and links are adjusted during optimization and are not controlled during the topology generation process.

Using this seed as the base, all the grammar rules are formulated. The graph-grammar rules are organized into different sets based on their functions and there are four different grammar rules used in this research. The first grammar rule set consists of eight rules. The grammar rules pertaining to the first rule set are shown below in various tables (Tables 4-2 to 4-9). Shown in Table 4-2 is a rule that attaches a link to the pivot node in the input seed. The rule recognizes that the seed graph does not contain any global label (in this case “1”) and after applying the rule assigns a label of “1” to the seed graph. This rule is applied only once as there is a negating label entry “1” under rule properties that prevents this rule being used again on a candidate graph that has the said label. The node on the left side of the rule consists of a “Negate Labels” entry for the label called “linked”. This means that if the seed consists of a node with the label called “linked”, then the rule is not recognized on that graph. This is done to prevent attaching a link to that pivot if it is already connected to another link. Though there can be several links connected at any pivot, for the sake of simplicity and correctness of rules, we are restricting that number to just two links at every joint.

Table 4-2: Rule to connect the input pivot with a link

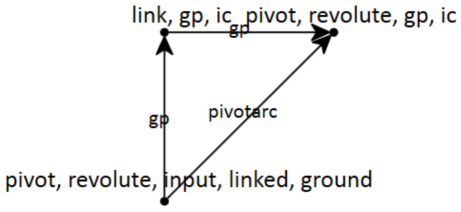
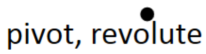
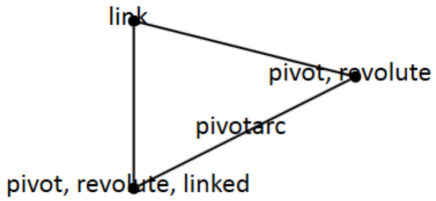
Left Hand side of rule	Right Hand side of rule
pivot, revolute, input, ground	

Table 4-2 continued.

Negate Labels: linked	
Rule Properties: L Negating Labels: 1, 2 R Labels: 1	

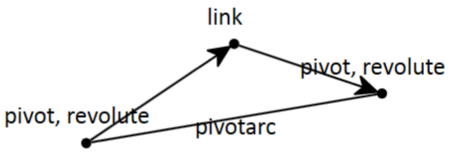
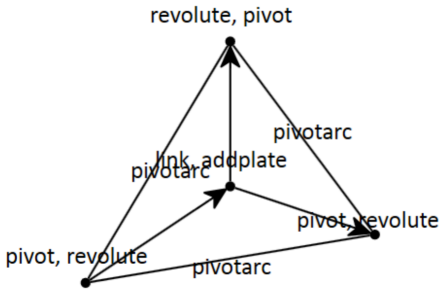
The second rule in this set is shown in Table 4-3. Here, the rule is attaching a link and a pivot to another pivot node that does not contain any of the "Negate Labels" specified in the table. That is, if the particular node in consideration consists of any of those labels under "Negate Labels" (Note: the labels in graph indicate the minimum number of labels that should be part of the node), then that node will not be recognized. After the rule is applied, the new pivot that is created does not contain the label "linked", indicating the availability of that pivot for further manipulation by rules. At the same time, the original pivot where the link is attached has attained the "linked" label. Also, this rule works only on those graphs that contain label "1".

Table 4-3: Rule to add a link to an existing pivot

Left Hand side of rule	Right Hand side of rule
	
Negate Labels: linked, input, ground, slider, sc	
Rule Properties: L Labels: 1 L Negating Labels: 2 R Labels: 1	

The third rule is shown below in Table 4-4 that creates a ternary link from a binary link. In this study, we are limiting the type of links to binary and ternary, though it is very easy to develop grammar-rules to create other link types such as quaternary and pentagonal links.

Table 4-4: Rule to convert a binary link to a ternary link

Left Hand side of rule	Right Hand side of rule
	
Negate Labels on pivot nodes: slider Negate Labels on link node: ground, addplate, sc	
Rule Properties: L Labels: 1 L Negating Labels: 2 R Labels: 1	

The rule shown in Table 4-5 adds a “link” node between two “pivot” nodes that are already part of different links and do not contain “linked” label on the “pivot” nodes. This way two unconnected pivots are joined by a link. Also note that there should not be any prior connection between these two “pivot” nodes in consideration. Though the “linked” label specified under “Negate Labels” for this rule should take care of that situation, the other labels on the node as well as the label that “Must NOT Exist” on the arc are provided just as a safety net. Table 4-6 displays a rule that identifies the presence of a four-bar loop around the input pivot. This rule is required to add sliding joints to the mechanism. The reason for this is that the generalized kinematic analysis for

indeterminate mechanisms with sliding members has not been developed (will be explained further in Chapter 5) and hence the restriction in adding sliding members to those mechanisms with input four-bar loop. Also, for simplicity sake, the sliding members are restricted to align alongside the frame (i.e., grounded sliding members). This rule assigns a "fourbar" label to the overall graph (and not to the concerned node).

Table 4-5: Rule to connect two pivots with a link

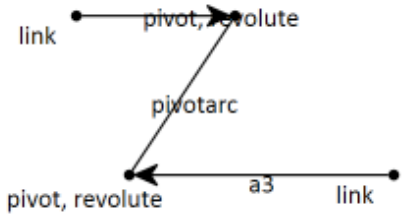
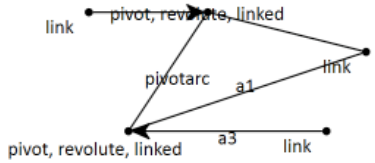
Left Hand side of rule	Right Hand side of rule
	
Negate Labels on "link" nodes: ground Negate Labels on "pivot" nodes: linked, slider, sc, notcon "Must NOT Exist" on arc with label "pivotarc"	
Rule Properties: L Labels: 1 L Negating Labels: 2 R Labels: 1	

Table 4-7 shows a rule that replaces the grounded revolute joint in a four-bar mechanism with a sliding member (prismatic "P" joint). Note the associated label changes at the pivot and link nodes. The seventh rule adds two links that represent a sliding member and is shown in Table 4-8. Table 4-9 adds a grounded revolute joint to a pivot that does not contain "linked" label.

Table 4-6: Rule to identify a four-bar input loop within a mechanism

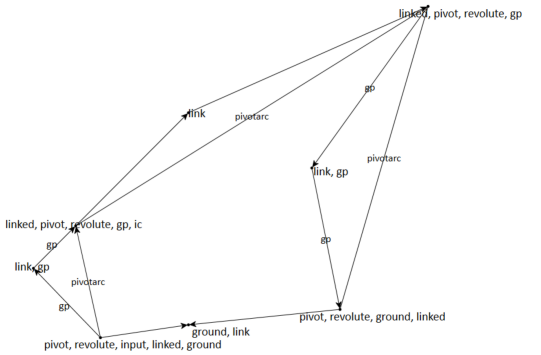
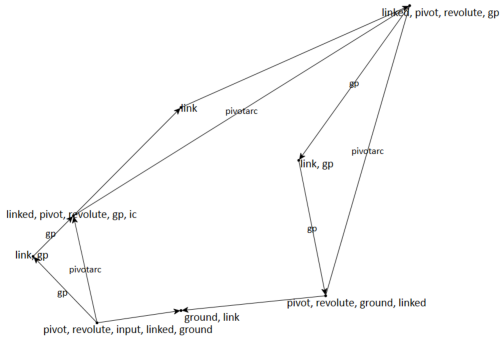
Left Hand side of rule	Right Hand side of rule
	
Rule Properties: L Labels: 1 L Negating Labels: 2, fourbar R Labels: 1, fourbar	

Table 4-7: Rule to replace revolute joints with sliding joints

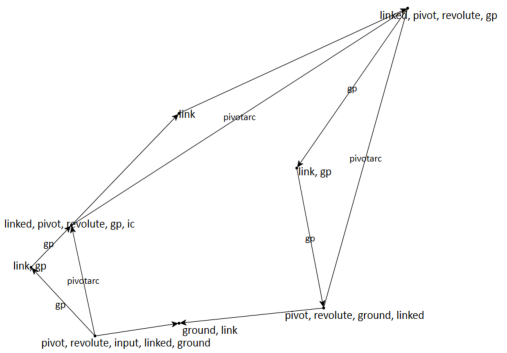
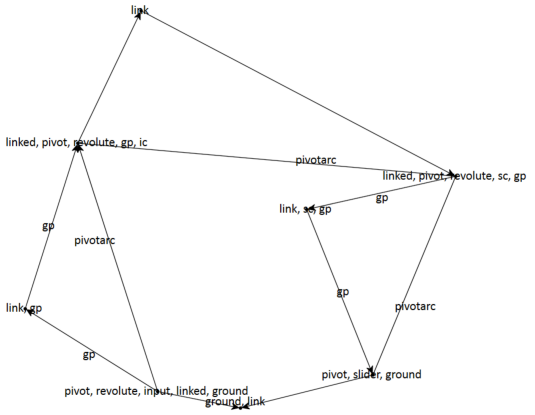
Left Hand side of rule	Right Hand side of rule
	
Rule Properties: L Labels: 1, fourbar L Negating Labels: 2 R Labels: 1, fourbar	

Table 4-8: Rule to add a sliding joint to a pivot

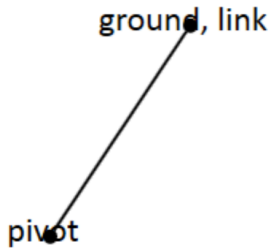
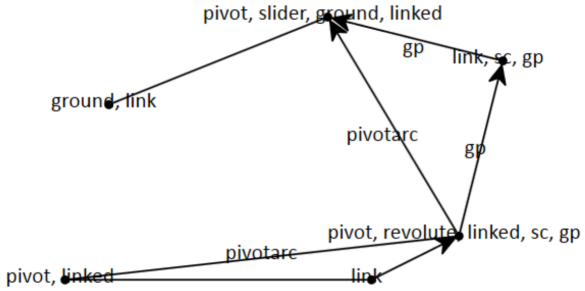
Left Hand side of rule	Right Hand side of rule
	
Negate Labels on “pivot” node: input, ground, sc, slider, linked “Must NOT Exist” on arc between the “ground, link” node and “pivot” node	
Rule Properties: L Labels: 1, fourbar L Negating Labels: 2 R Labels: 1, fourbar	

Table 4-9: Rule to connect a pivot to the ground with a link

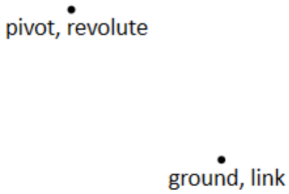
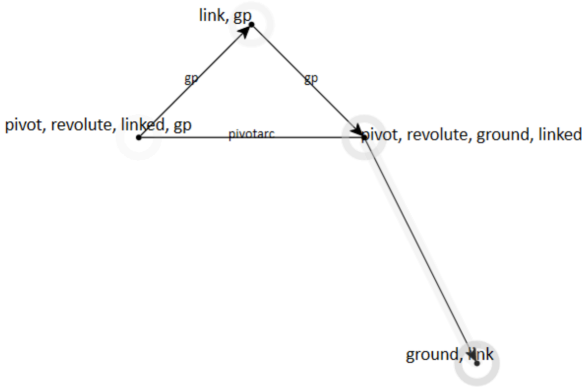
Left Hand side of rule	Right Hand side of rule
	

Table 4-9 continued.

Negate Labels on “pivot” node: linked, ground, gp, ic, notokay Negate Label on “ground,link” node: ic, gp	
Rule Properties: L Labels: 1 L Negating Labels: 2 R Labels: 1	

The eight rules belonging to the first rule set can be summarized as shown in Table 4-10 below.

Table 4-10: Summary of the functions of each rule in rule-set #1

Rule No	Function
1	Add a link to the input joint
2	Add a link and a pivot to another joint
3	Create a ternary link from a binary link
4	Connect two pivots
5	Identify a four-bar input loop
6	Replace revolute joint with a sliding joint
7	Add a sliding member to a pivot
8	Connect a ground pivot through a link with another joint

Though all the grammar rules have been extensively tested, there were some mechanism topologies that consisted of a truss structure resulting in a 0-degree of freedom. In order to avoid generating a mechanism with a truss, another set of rules was created to detect such cases, remove those invalid connections in the graph and then add a “notokay” label as in rule 1 in Table 4-11 or “notcon” label as in rules 2, 3 and 4 in Table 4-11 to the concerned nodes so that when the first rule set is reapplied, topologies with trusses are

not regenerated. There are four rules that form part of the second rule set and are shown below in Table 4-11.

Table 4-11: Rules in rule set #2

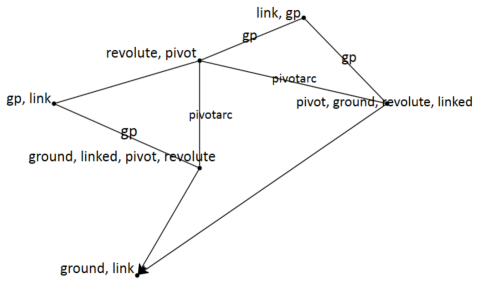
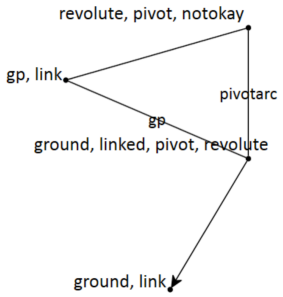
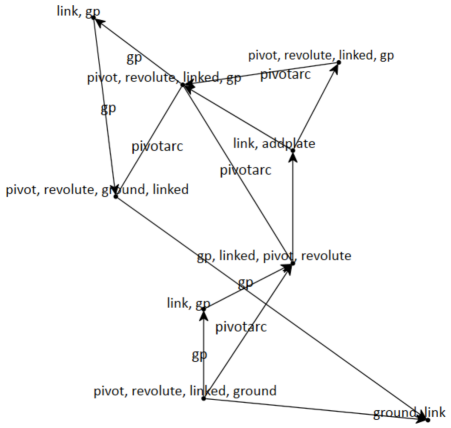
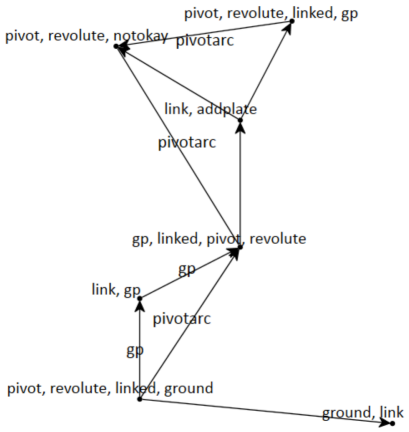
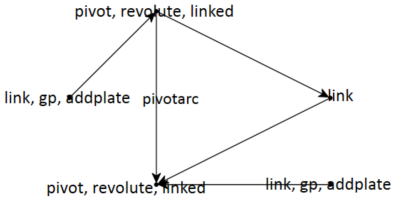
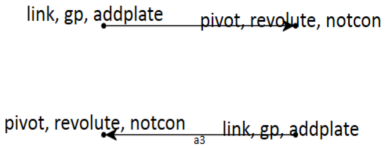
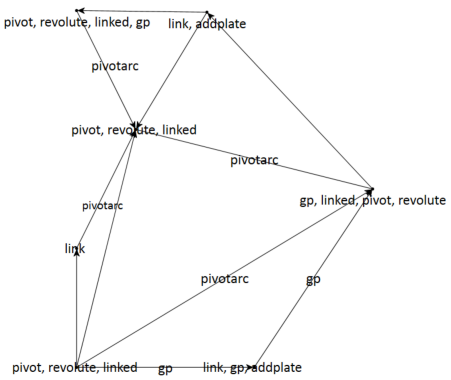
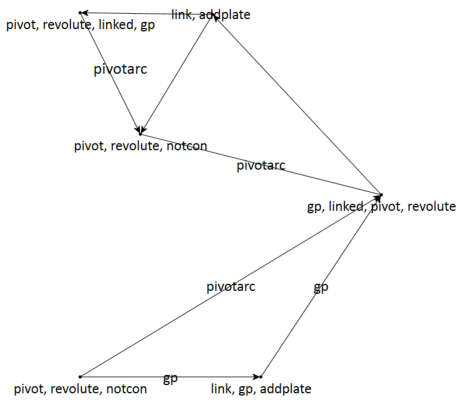
Rule No	Left Hand side of rule	Right Hand side of rule
1		
2		
3		

Table 4-11 continued.

Rule No	Left Hand side of rule	Right Hand side of rule
4		

Despite the second rule set, there is a class of topologies that returns a one-degree of freedom based on Greubler's equation despite the presence of a truss as shown in Figure 4-6. The figure returns a degree of freedom equal to 1 due to the fact there is a ternary link connected at joint H while the rest of the structure is a truss. This is not detected by rule set 2 since link (D-F) is connected only at the last stage in the generation process after which this topology is retrieved for further synthesis. Therefore, the strategy adopted to avoid this candidate or similar candidates being generated is to detect the presence of a link where one or more joints are not connected. That is, if the Gruebler's criterion returns a value of 1 but the graph consists of a binary link with a pivot without "linked" label or a ternary link with two pivots that do not contain the "linked" label, then the situation similar to Figure 4-6 is encountered and the concerned candidate graph is removed from further consideration. This ensures that only valid one-degree of freedom joints are presented for further synthesis.

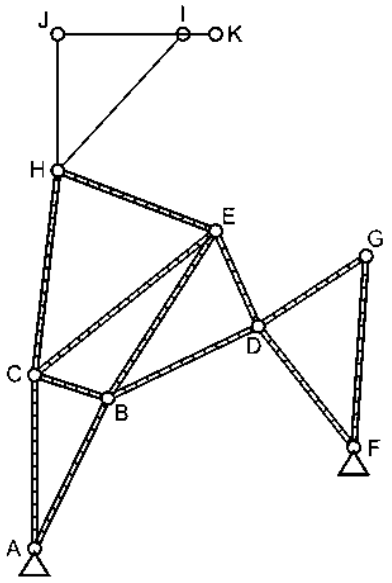
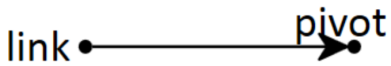
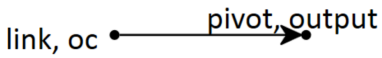
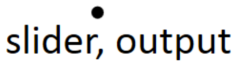


Figure 4-6 A mechanism with 1-degree of freedom when calculated using Gruebler's equation but consists of a truss as indicated by the hashed representation

The third rule set shown in Table 4-12 assigns an "output" label to one of the "pivot" nodes. This label is used to inform the optimization routine that this pivot is required to trace the desired path specified by the user. "output" labels are not assigned to joints that are grounded or to those joints part of the "input" binary link node. There are two rules here, the first rule assigns "output" label to a node representing a revolute joint and the second rule assigns "output" label to the sliding joint. The associated properties are listed under each rule in Table 4-12.

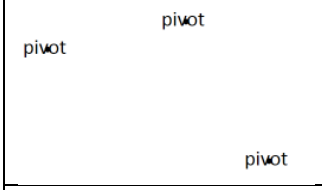
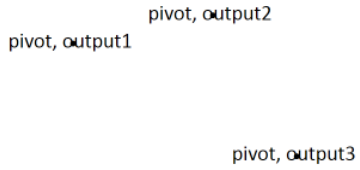
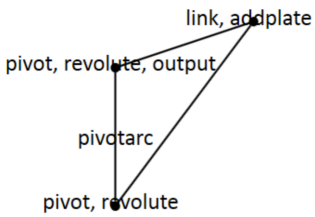
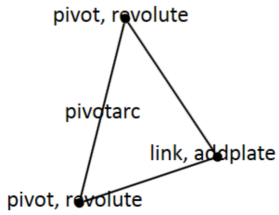
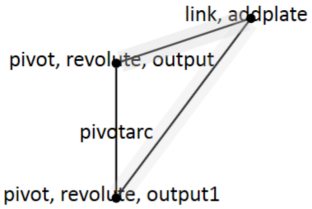
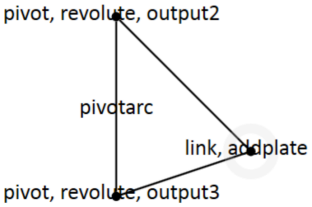
Table 4-12: Rules in rule set #3

<table>
<tr><th>Rule No</th><th>Left Hand side of the rule</th><th>Right Hand side of the rule</th></tr>
<tr><td rowspan="3">1</td><td> 
</td><td> 
</td></tr>
<tr><td>Negate Labels on “link” node: ground, ic
Negate Labels on “pivot” node: input, ground, slider, ic, sc</td><td></td></tr>
<tr><td colspan="2">Rule Properties:
L Labels: 1
L Negating Labels: 2
R Labels: 1,2</td></tr>
<tr><td rowspan="3">2</td><td> 
</td><td> 
</td></tr>
<tr><td>Negate Labels on “slider” node: input, sc, avoid</td><td></td></tr>
<tr><td colspan="2">Rule Properties:
L Labels: 1
L Negating Labels: 2
R Labels: 1,2</td></tr>
</table>

Note the “avoid” label in rule number 2 in the above table. This label is assigned during runtime in the optimization routine whenever the desired path to be traced by the mechanism is not a straight line. This is done to avoid unnecessary computations such as trying use a grounded slider to trace an elliptical curve, which is not feasible.

There is another set of rules that will be used in conjunction with rule sets #1 to #3 to solve challenge problems #2 and #3. This rule set is used to solve single input – multiple output (SIMO) scenarios. Two rules are part of rule set #4 and are shown below in Table 4-13. The first rule adds additional labels such as “output1”, “output2” and “output3” to different pivots and the second rule assigns the same labels but to two different ternary links.

Table 4-13: Rules in rule set #4

Rule No	Left Hand side of the rule	Right Hand side of the rule
1		
	Negate Labels on "pivot" nodes: output, ground, ic, slider	
	Rule Properties: L Labels: 1,2 L Negating Labels: 3 R Labels: 1,2,3	
2	 	 
	Negate Labels on "pivot,revolute" node: ic, ground, input, sc, slider, output1, output2, output3 Negate Labels on "link" node: ground	
	Rule Properties: L Labels: 1,2 L Negating Labels: 3 R Labels: 1,2,3	

The four rule sets (#1 to #4) are organized as per the flow chart given below in Figure 4-7. The flow chart depicts a typical tree-search scenario and the overall rule application process by which all possible one-degree of freedom planar mechanisms are generated at every level in the tree. The final list of candidates is passed onto the optimization routine where the candidates are parametrically optimized depending on the requirements of the user.

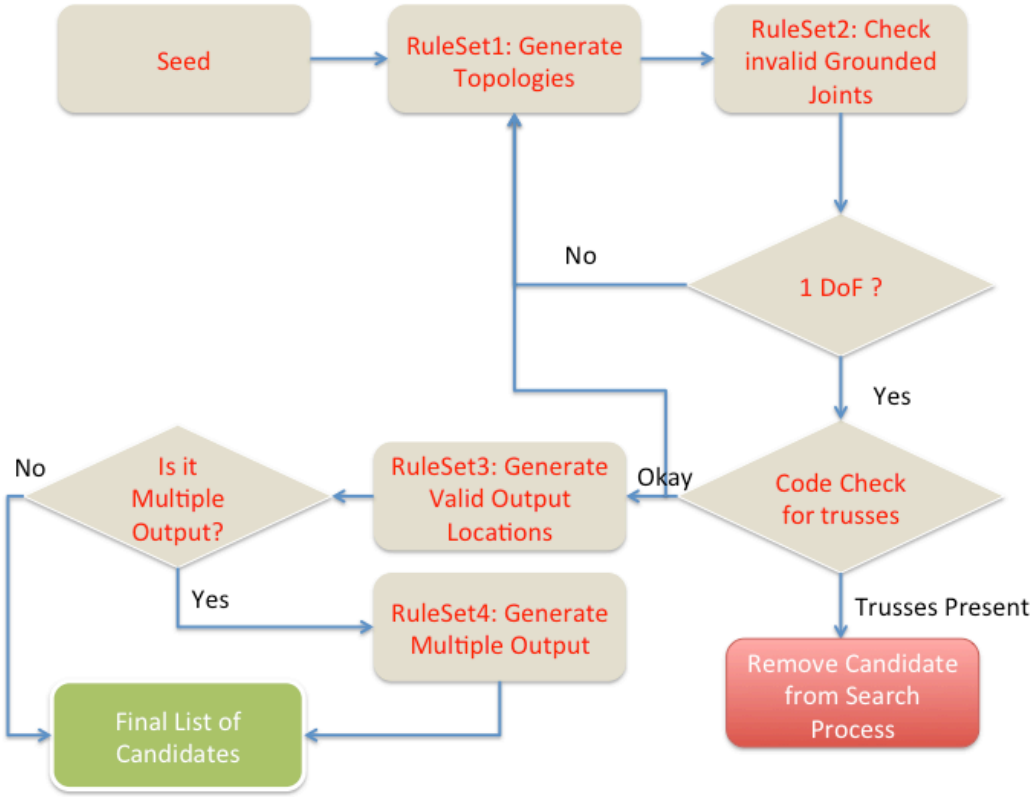


Figure 4-7: Flow chart to illustrate the rule application process

In the flow chart, the degree of freedom is calculated using Gruebler's criterion [2]. In the next section, the type of topologies generated for different levels in the search tree will be presented.

4.4 ENUMERATION OF TOPOLOGIES

During the grammar rule formulation process, all mechanism topologies that are described in textbooks and other literature for revolute and prismatic joints have been manually generated. This process helped in testing the rules as well as fine-tuning them. The developed grammar rules are used to generate all possible topologies through an exhaustive generation process. The enumeration was carried out using a program written in C#. Although the algorithm is very similar to the flow chart described in Figure 4-7, there are a few additional functions used in order to be computationally efficient and generate the maximum amount of mechanism topologies. Those functions serve two purposes. The first function is used to remove isomorphic candidates from the search process and the second is used to remove confluent rule options during the generation process. Though the number of confluent options (same rule is recognized at the same location – just the direction is different) has been minimized due to the use of directed arcs (arcs with arrowheads used in different), we still wanted to ensure that duplicate candidates are not generated and computational resources wasted. Moreover, since we are using the open-source mono for C# client, we are unable to completely take advantage of C#'s built-in parallelization routines. Due to this, the memory was maxed out and we had to restrict the generation process to level 11. The list of topologies generated till this level is presented below in Table 4-14. Detailed information about the different types of four-bar mechanisms generated is given as a sample in Table 4-15. This clearly shows the presence of isomorphic candidates and possibly confluent options during the rule recognition process.

Table 4-14: List of topologies generated till level 11 in the search process

No of Links	No of Pivots	Total Candidates Generated
4	4	50
6	7	497
8	10	360
10	13	2

Table 4-15: Types of four-bar mechanisms enumerated till level 11

No of Ground	No of Links	No of Pivots	No of Ternary Links	Prismatic Joints Present	Count
2	4	4	0	No	1
				Yes	1
			1	No	6
				Yes	4
			2	No	14
				Yes	9
			3	No	9
				Yes	6

A typical topology is described in the following manner "2-4-4-2-revolute-no prismatic". This should be read as: "2 ground pivots-4 links-4 joints-2 potential ternary links-revolute joints-and not prismatic joints". You may notice that we have a nomenclature stated as "potential ternary links". This is used to indicate that there are links containing three pivots but all the pivots may not be connected to other links. But as the curve produced by any point on those links will be different, we feel it is important to identify the presence of such links. Additionally, the graph names of ternary links may also be presented when listing the generated topologies. This gives an idea to the user about the topology before even looking at the appropriate candidate graph. The results presented in Chapter 7 will present the mechanism topologies in a similar manner.

There are a total of 909 valid candidates and a total of 4846 candidates when "output" label is assigned to the generated graphs using rule set 3. The number of candidates with links 10 and more is generated further down the tree, at levels 12 and greater and due to the insufficient capability in handling large stack of data using the open-source mono for c# implementation, we are unable to present data on the types of links that are generated at those levels. But on a survey of the four and six bar mechanisms, we have been able to confirm the validity of rules through a manual review of the topologies generated in this process.

The numbers of mechanisms listed in Table 4-14 clearly point the need for a rigorous isomorphism detection methodology since duplicate candidates can be prevented from being optimized. But this aspect has not been considered in this dissertation, as our primary goal is to implement a repeatable algorithm for automatically synthesizing the topologies and their parameters.

4.5 DISCUSSION

The grammar rules explained in the earlier sections are able to capture maximum information about the topology, including information about joints and links. There are a total of 16 rules in four different rule sets. Though we are able to generate different mechanisms, our rules are limited to generating binary and ternary links and permit only two links to be connected at any joint. Also, the prismatic joints are restricted to slide alongside the frame and are connected to a mechanism with four-bar input loop since we do not yet have a generalized routine for solving indeterminate mechanisms with prismatic joints (P). These constraints were primarily added to adequately manage computational resources when the complete program is executed i.e., when the search process is coupled with optimization and kinematic analysis, the resources required are enormous and the current implementations of the software (aka our programming as well as mono for C#) is not robust for multi-threaded multi-core processing. Despite these constraints, it is possible to extend this representation scheme to include different joint types in planar mechanisms with the availability of better tools.

4.5.1 Isomorphism and Confluence

Since our research deals with a methodology to represent mechanisms using graph grammars for synthesis purposes, isomorphism and confluence are important issues to be addressed. Isomorphism refers to the structural equivalence of topologies and researchers have developed different methods to identify and deal with isomorphic solutions as stated in the review by Mruthyunjaya[15]. While a particular degree of

freedom system is desired by the user-designer, there are usually constraints on kinematics that are not considered in isomorphism. Since our goal is synthesis where topologies generated by a search process will be evaluated, one could take isomorphism into advantage to reduce computation. This has been done in our topology enumeration code as well as the overall synthesis code, where a first-level isomorphism check has been introduced. This code basically checks if two mechanism configurations are basically the same by comparing a few parameters of the topology. The parameters considered are: number of ground pivots, number of links, number of joints, number of ternary links and the names of the nodes representing ternary links and positions. This helps in segregating some of the isomorphic candidates but not all of them. Through this first-level basic check, we have a slight reduction in the usage of computational resources. Though the best solution is to develop rules that reduce the occurrence of structurally equivalent topologies, it is not always possible to compose rules that do not generate any isomorphic candidate. This is because when the focus is on developing fewer rules to generate maximum candidates, there is a higher chance for producing isomorphic candidates and invalid solutions (as described in section 4-2). Thus, the 16 rules developed result in distinct topologies but with isomorphic variations. Also, the rich set of labels that are associated with every node and arc used in this research produce an information rich graph but at the same time make the detection of isomorphically equivalent candidates tougher. But then without labels, it is not possible to associate the parameters that uniquely define a mechanism topology, as is the case in other related research that make use of the systematic method. The topological variations, as shown in Figure 4-8 where the topology is the same but the desired output pivot's locations are different (using rule set 3), are also achieved using our rules since those mechanisms are characteristically different (can be seen by the curves generated by the respective pivots).

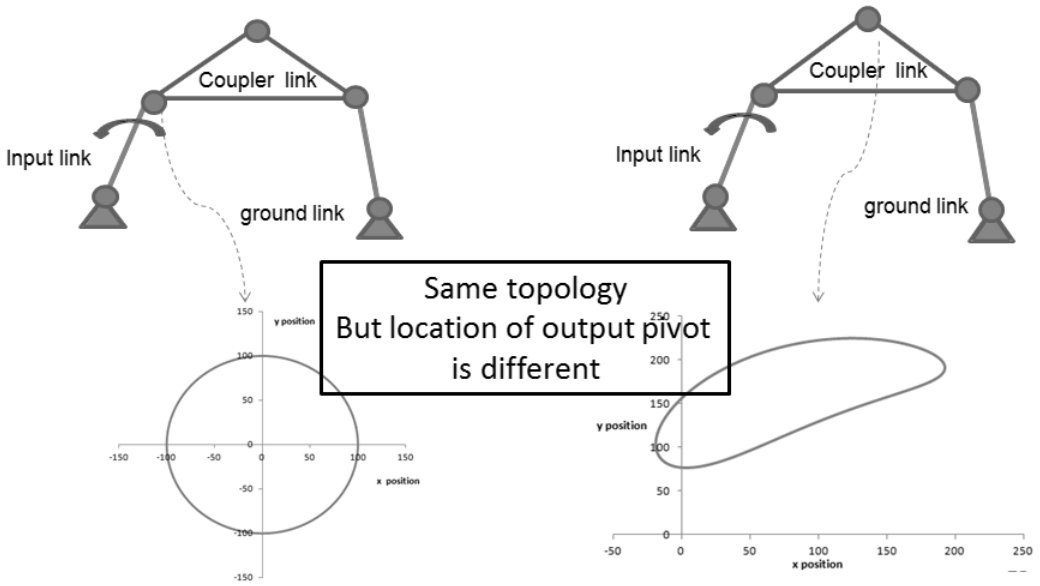


Figure 4-8 An instance of a four-bar mechanism with two different output locations that produce different output curves

From Table 4-14, the total number of valid solutions (before assigning output locations to each candidate) is 1936, while the actual number of solutions without isomorphic candidates and confluent options is 909. The first order isomorphic candidates and confluent option identification code has helped in removing about 47% of such candidates and helped in gaining significant computational resource as a result.

Also (refer to Figure 4-4), it is important to note that search trees may unavoidably include repeat states. This indicates that there may be multiple paths to the same configuration. This is an issue in graph rewriting systems known as confluence, wherein identical topologies at different locations in the tree can be traced to a common parent. This has been significantly reduced through a first-order confluent options check during the rule-application process but still we can see from the results displayed in Table 4-14 that there are duplicate candidates in the results (for instance, 50 four-bar mechanisms). Due to the generic nature of rules, it is not possible to completely remove all duplicate candidates through a first-order check since these topologies are generated at

completely different levels. In order to completely eliminate such candidates, detailed isomorphic and confluent check routines have to be incorporated.

4.6 CONCLUSION

A comprehensive representation scheme has been developed along with grammar rules to generate all possible topologies of one-degree of freedom planar mechanisms in a generic manner. Enumerating candidates using an exhaustive search process has tested the grammar rules for completeness. The implementation of first-order checks for isomorphism and confluence option reduction has helped in reducing computation though it is has been shown to not completely remove the occurrence of duplicate candidates. The generated candidates combined with kinematic analysis (Chapter 5) will be used to synthesize concept designs for various benchmark problems using an optimization algorithm that will be explained in Chapter 6 whose results will then be presented in Chapter 7.

Chapter 5: Kinematic Analysis

The topologies generated using grammar rules are parametrically optimized (explained in the next chapter) to user requirements. Typically the requirement is specified in terms of (x, y) coordinates of the path traversed by a joint in the mechanism or as an array of angles followed by a link depicting a particular motion. In order to ascertain these details during optimization, kinematic analysis is used to evaluate the position, velocity and acceleration of different joints and links in the planar mechanisms. This is an important part of the system proposed in Figure 3-2. Restricting the simulation to kinematics helps to quickly generate designs in kinematic outline form rather than exhaustively evaluating each mechanism for their dynamic characteristics too. There are several commercial programs available for kinematic analysis of planar mechanisms such as Working Model [6], ADAMS [7] and SAM [8], but these programs do not have an API (application programming interface) that would help in simulating the results of our optimization implementation. There are also no robust and generic open-source kinematic analysis tools available for this purpose. Hence, considerable time and effort have been devoted to developing a generic kinematic analysis tool for planar mechanisms that can be used in an automated setting.

This chapter details the development of this generic kinematic analysis tool for planar mechanisms. Section 5.1 explores the need for a generic tool for kinematic analysis. This is followed by the implementation procedures and results for planar mechanisms with one-degree of freedom consisting of four-bar loops in section 5.2. Section 5.3 highlights the method developed for solving positions of indeterminate one-degree of freedom mechanisms where the existing methods in literature are not applicable and the alternate solution methods are not scalable to a generic level.

5.1 INTRODUCTION

In order to computationally synthesize planar mechanisms, it is important to automatically define the boundary conditions and adjust the necessary parameters to evaluate the kinematics of the mechanism in consideration. This is in sharp contrast with the existing software available for kinematic analysis that requires the user to manually input the mechanism for analysis. Furthermore, such commercial tools analyze mechanisms through dynamics-based physics engine that can be erroneous in comparison to pure kinematic analysis. While the inclusion of dynamics information has benefits, it challenges the mechanism designer to fully specify all features and speeds in order to test whether a concept traces a desired path or motion.

As mentioned in the beginning of this chapter and in the literature review, there are no open source kinematic analysis tools available that can be integrated with a design generation tool as envisioned in this dissertation. One of the reasons is the absence of a method, which not only solves the kinematics reliably but also is applicable to generalized n-bar mechanisms. This has led the automated-synthesis projects in this area to be limited to fixed topologies such as a four-bar mechanism or a six-bar mechanism with revolute joints [48,52,60] and occasionally prismatic joints [17] as there are standard formulations in existing textbooks [2–4] to solve such topologies. As a result, only variations in the link lengths are produced and no alternate mechanisms are suggested. Designing planar mechanisms is a challenging activity, where mechanisms consisting of multiple links and different joint types have to be synthesized. Automating this task is beneficial but lack of kinematic analysis solvers that can be used to automatically analyze generic mechanism designs has hindered its progress.

The methods to determine the position kinematics of planar mechanisms are classified into two categories in the literature namely graphical and analytical. The graphical method is the dyadic decomposition method while the analytical method involves solving trigonometric loop equations. In the next section (Section 5.2), the generalization of kinematic analysis is presented for mechanisms with four-bar loops.

The generalized algorithm includes the instantaneous center of rotation method for velocity analysis, vector polygon approach for acceleration analysis and the dyadic decomposition method for position analysis of planar mechanisms. The graphical methods have an algorithmic nature and can be easily generalized. The presented implementation also includes the methods developed by Foster and Pennock [36] and Hernandez et al. [61] for determining the instant centers and positions respectively of the double-butterfly linkage. The implementation takes advantage of object-oriented programming and the graph representation of planar mechanisms into building a generalized kinematic solver that can operate on any single-degree of freedom system with at least one four-bar loop along with the double-butterfly linkage. Another advantage of the program is its ability to evaluate mechanisms consisting of R, P and R-P joints.

But this implementation is not applicable to multi-loop indeterminate mechanisms such as Stephenson II mechanism [3] and the double-butterfly linkage [28] since it is not possible to obtain the decomposition necessary to compute subsequent positions of pivots in the mechanism. Although there are geometric methods for double-butterfly linkages in the implementation, their performance is not reliable and hence alternate methods had to be explored to compute the positions of indeterminate mechanisms. The analytical loop equation method is a possible alternative on the other hand that involves formulating loop equations in terms of sine and cosine of the angles of the different links in the mechanism. The resulting equations are non-linear and there are solution forms available for simple four to six-bar mechanisms in the literature. But for a mechanism like the double-butterfly linkage, whose loop equation formulation results in six equations with six unknowns, there are no standard solution forms available and the existing numerical methods (in packages such as MATLAB) often fail to obtain any meaningful solutions. The lack of kinematic methods for solving such mechanisms has possibly impeded the use of such planar mechanisms in practical applications. Section 5.3 explains the optimization-based approach that has been developed for solving the position kinematics

of single-degree of freedom planar mechanisms by minimizing error of link lengths. The method is tested on indeterminate single-degree of freedom mechanisms consisting of revolute joints, where it is shown that precise results can be obtained with excellent computational efficiency. The capability of the method in solving both initial and finite position problems is also demonstrated, where it is also shown that the method and implementation are generic to any n-bar mechanism with revolute joints.

5.2 KINEMATIC ANALYSIS OF PLANAR MECHANISMS WITH FOUR-BAR LOOPS

The kinematic analysis routine requires the location information (coordinates) of the pivots at the initial point in time (time t=0). The evaluation function outputs the kinematic properties (namely position, velocity and acceleration) of all pivots. Assumptions made include a constant input angular velocity and single input-single output system while formulating the problem. The programming is carried out in C#. Since our implementation is integrated with the representation explained in the previous chapter, the following section is explained using the example of a four-bar mechanism (Figure 4-1 and Figure 4-2). While describing the algorithm, a brief overview of the kinematic method is provided followed by the generalization algorithm. The focus is on evaluating determinate n-bar one-degree of freedom systems with R, P and R-P joints. As shown in Figure 5-1, the analysis proceeds with velocity computation followed by acceleration and then position. This order is not a necessary condition as the position and velocity computations are independent for mechanisms with four-bar loops due to the implementation of graphical methods.

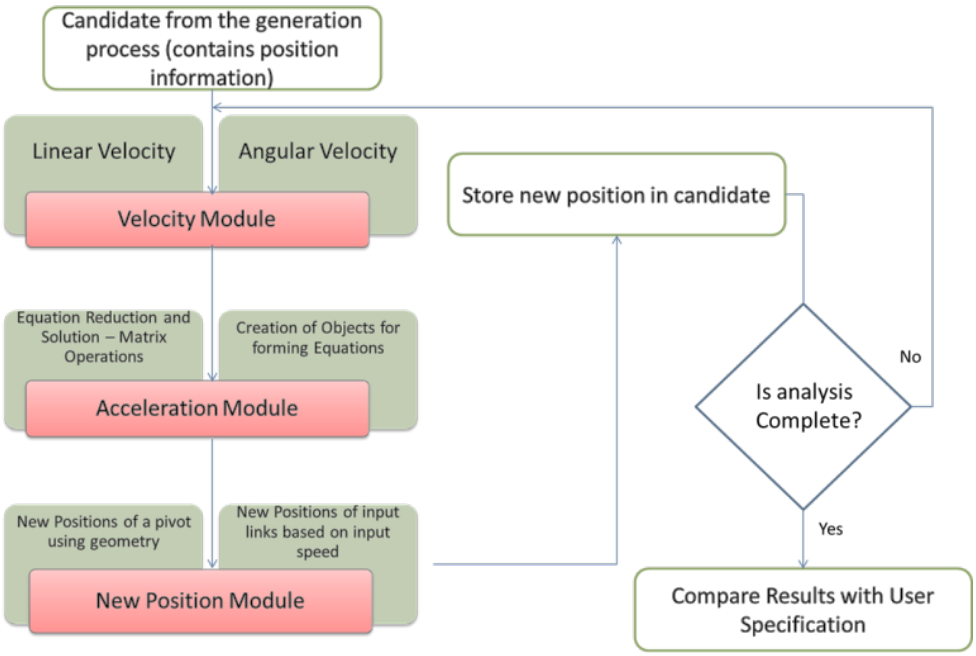


Figure 5-1: Flow chart for the kinematic analysis of mechanisms with four-bar loops

5.2.1 Velocity Formulation

Velocity is determined using the graphical instant center method, which involves comparing the instant centers between every link and every other link, which can be classified as primary and secondary. The instant centers are determined using the Kennedy-Aronholdt theorem [3]. This theorem states that the primary instant centers are those defined between connected links and are located at shared pivots. Each secondary instant center is located at the intersection of two lines (the end points of each line being instant centers), which can be determined using the circle-diagram method. The instant center technique is chosen for velocity determination since it exhibits an algorithmic logic that can be easily programmed, can be generalized to any topology and is completely analytical.

Computationally, the basis of solving the instant center method is to create a list of objects of type ϕ, for each pair of links;

$$\phi = \{x, y, \omega, link_i, link_j, pivot\}$$

where *(x, y)* is the location of an instant center; ω is the relative angular velocity between the links and *pivot* is the common pivot to the links if it is a primary instant center. The $link_i$, $link_j$ and *pivot* reference particular nodes in graph representation. Consider an instance of ϕ (referring to Figure 4-1 and Figure 4-2) where $link_i$ is the leftmost link node with labels "link,gp,ic" and $link_j$ is the topmost link node with labels "link". Since these two links are connected at the joint with labels "pivot,ic,gp,revolute,linked", this joint information (consisting of information such as the node name, (x, y) position, etc.) is assigned to the *pivot* variable in ϕ. For the entire mechanism, corresponding to each unique instant center, there are $\phi's$ defined that follows the condition *p*(p-1)/2*, where *p* is the number of pivots ("pivot" nodes in graph terms) in the mechanism. The information on primary instant centers is available from the topology of the mechanism since they are located at pivots common to two links. During the first pass of the velocity program, these primary instant centers are determined first and the corresponding entries in ϕ are filled. Following this, secondary instant-centers are obtained using an innovative programming logic that replicates the circle-diagram approach. The secondary instant centers are obtained by the intersection of the lines containing primary instant centers. The algorithm below indicates the method to determine primary and secondary instant centers. (Note: In the algorithm below, *PIS* indicates an R-P joint (pin-in-slot) and *Slider* indicates a prismatic (P) joint and *pivot* in general refers to a revolute (R) joint).

$Set\ \phi = \{x, y, \omega, link_i, link_j, pivot\}$ *for each N*
N=p(p-1)/2; p=number of pivots; N=number of instant centers*
//Primary instant centers are located on Pivots and are recorded in ϕ
Start Do
Let i=$link_i$ $and\ j = link_j$
If $link_i$ or $link_j$ $are\ not$ *PIS or Slider*
Create two new instances of ϕ -> $\phi 1, \phi 2$
$\phi 1$ = *CALL InstantCenters connected to i*
$\phi 2$ =*CALL InstantCenters connected to j*
Create Matrix K (2x2) for Circle Diagram Path
Obtain New Secondary Instant Center
//For Double-butterfly Linkage

Obtain Two Secondary Instant Centers [6]
Else //separate for PIS and Slider
$\phi 1$ = CALL InstantCenters connected to i
$\phi 2$ =CALL InstantCenters connected to j
Create Matrix K (2x2) for Circle Diagram Path
Obtain New Secondary Instant Center
End Loop after N instances of ϕ are determined

// To Obtain $\phi 1$ and $\phi 2$
Function InstantCenters
Start For Each ϕ
If $\phi\ (x, y)$is not NULL and i=$link_i$ and j = $link_j$
Add ϕ to $\phi 1$
End Loop
End

// To Determine Circle Diagram Path
Function Circle Diagram Path
Start For Each $\phi 1$
If $\phi 1(link_i)$ ==i -> Add to B
Else Add to B
End Loop
Start For Each $\phi 2$
If B== $\phi 2(link_i)$ OR B== $\phi 2(link_j)$
Then Add B to Matrix K
End If
End Loop
End

//To Determine Secondary Instant Center
//Matrix K is of the form [a b;c d]
//where a, b, c, d are (x,y) of known Instant centers
//Intersection of Line a-d and b-c will result in the New Secondary Instant Center

During the execution of the overall do-while loop, there could be situations when the required two equation paths in the circle diagram approach are not obtained. So the do-while loop would continue to the next instant center and revisit missing instant centers during subsequent cycles of the loop. The use of the do-while-loop makes the process generic since, until all instant centers are determined, the process repeats. If, during one

complete pass, new instant centers are not determined, the program exits, and this could be due to an infeasible topology (indeterminate mechanism). These built-in checks are some of the unique features of the generalization methodology presented in this chapter. There are a few special cases built-in for prismatic and pin-in-slot joints since the method of determining instant centers vary for such elements. These are incorporated in such a way that the generic architecture of the program is unaffected. Once all instant centers are obtained, the computation of angular and linear velocities is carried out using the standard procedure explained below for one of the links and pivots. For the coupler link (considered as link 3) in Figure 4-1 (in Figure 4-2, this corresponds to the node with only "link" label),

$$\omega_3 = \frac{\omega_2 \times (I_{1-2} - I_{2-3})}{(I_{1-3} - I_{2-3})} \text{ rad/s}$$

$$V_3 = \omega_2 \times (I_{1-3} - I) \quad \text{unit/s}$$

where ω_3 denotes angular velocity of the link and V_3 denotes the linear velocity of the pivot between input and coupler links (in Figure 4-1). "I" in the V_3 equation above represents the instant center located at that joint between input and coupler links. Given a known input angular velocity, other angular and linear velocities can be easily determined once the instant centers are obtained. The velocity module also computes slip velocities and Coriolis component if they exist in the particular topology. The algorithm also includes the method demonstrated by Foster and Pennock [36] to determine two secondary instant centers of a double-butterfly linkage. The inclusion of this method enables solving velocities of the indeterminate double-butterfly linkage mechanism, which is an eight-bar one-degree of freedom mechanism.

5.2.2 Acceleration Formulation

Angular and linear accelerations are computed by forming the appropriate acceleration equations as listed below for the four-bar mechanism in Figure 4-1 and Figure 4-2.

$$\begin{aligned}
a_A &= a_D + 2\,v_A \times \omega_1 + r_{A/D} \times (\omega_1 \times \omega_1) + a_{Slip-A} + \propto_1 \times r_{A/D} \\
a_B &= a_A + 2\,v_B \times \omega_2 + r_{B/A} \times (\omega_2 \times \omega_2) + a_{Slip-B} + \propto_2 \times r_{B/A} \\
a_C &= a_B + 2\,v_C \times \omega_3 + r_{C/B} \times (\omega_3 \times \omega_3) + a_{Slip-C} + \propto_3 \times r_{C/B} \\
a_D &= a_C + 2\,v_D \times \omega_4 + r_{D/C} \times (\omega_4 \times \omega_4) + a_{Slip-D} + \propto_4 \times r_{D/C}
\end{aligned}$$

where *a* refers to the absolute acceleration; $2\,v \times \omega$ corresponds to the Coriolis acceleration; $r \times (\omega \times \omega)$ is the radial acceleration and $\alpha \times r$ corresponds to the tangential acceleration. Subscripts A, B, C and D refer to the four links of the four-bar mechanism (ground, input, coupler and follower). The acceleration equation is linear and the unknown acceleration terms can be obtained by solving these simultaneous equations using the form *Cx=b*, where *x* is the list of unknowns, *C* is the coefficient matrix and *b* is the list of constants. While solving the linear equations is trivial, the challenge lies in automatically creating *C* and *b* for an arbitrary n-bar mechanism for each time step. In order to formulate these simultaneous equations, an object ψ is generated for each acceleration equation (each row of *C* and *b*),

$$\psi = \{dir, node_x, node_j, \propto, \omega, radA, radV, A, V\}$$

where *dir* refers to the x or y acceleration component, $node_x$ and $node_y$ refer to the pivots or links relative to absolute and relative accelerations, $\propto$ is the angular acceleration, ω is the angular velocity obtained from the velocity program, *radA* is the radial acceleration, *radV* is the radial velocity, *A* is the absolute acceleration and *V* is the absolute velocity. Initially, there is an instance of ψ created for acceleration along x and y directions, which results in eight unique ψ*'s* for the four-bar mechanism corresponding to eight equations. As the equations are formed, terms such as ω_1 are eliminated since it refers to the angular velocity of ground link, which is zero. This automatic equation reduction ensures that we

have the same number of equations as unknowns. The number of equations that are eventually solved depends on the topology of the mechanism. For example, the acceleration equations for the four-bar mechanism used in the illustration reduce to solving six unknowns in six equations. This would be different for a six-bar mechanism or a double-butterfly linkage since those mechanisms consist of more number of links and pivots. The equations are solved using a matrix inversion technique. Cramer's rule is not practical in this case, since the method is extremely inefficient for matrices with order six or more when solved on a typical desktop computer. Likewise, the Gauss-Elimination and Gauss Seidel techniques require dominant diagonals, which are not guaranteed in this automated method for generic topologies. Therefore, the LU Decomposition technique is chosen wherein the existing matrix is subject to a reordering to ensure non-zero diagonals. The inversion technique gives appreciable results with errors on the order of 10^{-9}. The algorithm for generalizing the acceleration program is given below.

Start For
Form Acceleration Equation for Each Pivot in x & y directions
$$\psi = \{dir, node_x, node_Y, \propto, \omega, radA, radV, A, V\}$$
//Important to be unidirectional to prevent repetition
Get dir
$node_x$, $node_y$, $\omega, radV \text{ and } V$
End Loop
Form Ax=b
//x=Column Matrix of Unknown Acceleration terms
//A=Coefficient Matrix; b=Column of Known Values
Eliminate Ground link data and Reduce Order
$x=A^{-1}b$

5.2.3 Position Formulation

After velocity and acceleration analyses, position kinematics can be determined by employing a Taylor's series expansion or by using the graphical decomposition method. The pivot positions are obtained geometrically through dyadic decomposition. During this process, it is possible for a pivot to be in one of two positions (also referred to

as a solution branch in literature). The choice between the two is made based on previous position information as well as from the results of the numerical approximation (where position is approximated from velocity and acceleration information using Newton's laws). Due to limitations in the dyadic decomposition technique, mechanisms such as the double-butterfly linkage cannot be analyzed [29]. In order to overcome this disadvantage, the geometric iterative technique proposed by Hernandez et al. [61] has been included to handle such situations. The following algorithm gives the overall methodology for determining position.

Set Counter to 1
Develop Adjacency Matrix for Distance between Pivots
Set time-steps
Rotate Input Link by Θ
Assign NULL values to all other pivots' (x,y) except ground
Start Do
If No PIS or Slider Connection
    If Four Bar Loop Present
        Start from Two Known Positions
        Link Lengths as Radii
        Intersect Two Circles
        Compare with NewtonMethod & PrePos
        New Pivot Position is Obtained
    Else
        Geometric Iterative Technique
Else
    PIS or Slider Program //Circle-Line intersection
End Loop until Pivots have (x,y)

As one may notice in this implementation, pin-in-slots and prismatic joints require separate computation (like circle-line intersection), which is adapted into the program structure to increase the generality. The geometric iterative method is also programmed into this algorithm to operate on mechanisms that cannot be solved using dyadic decomposition. The algorithm checks if existing methods are applicable before computing velocities and position using the new method. The position module is also generic since the do-while loop operates in the same way as explained during instant

center determination and continues until all pivots are assigned new positions. The position module is also programmed to determine in-feasibilities (such as the limitation preventing the input from rotating a full 360°) in the mechanism. At the same time, the direction of the input crank can be reversed to determine the maximum travel in the opposite direction. In this way, rocker type mechanisms can also be analyzed and therefore the method is not restricted to solving only those mechanisms where the input crank can be rotated by 360°.

Once the kinematic properties are determined, the path generated by a path or the motion of a link is compared with the original problem specified by the user. This process is cast as an objective function, which is optimized to synthesize appropriate designs.

5.2.4 Results of Implementation

This generalized implementation of kinematics of planar mechanisms with the different joint types is validated using mechanisms available in standard textbook references. The mechanisms (such as a four-bar, a quick return and a six-bar) are manually created following the graph approach explained in Chapter 4 and the pivots are assigned coordinate locations as specified in various textbook references. The simulation is carried out for different time steps for 360° rotation of the input crank and the output (position, velocity and acceleration) of the links and pivots are obtained in a text (.txt) file. Figure 5-2 shows the kinematics of a four-bar mechanism (shown in the center) obtained using this tool for 500 time-steps. The plot on the top-left corner shows the acceleration profile of pivot *D* while the one on the bottom-left displays the predicted velocity of pivot *C* located on the ternary coupler link. Similarly, the plot on the top-right shows the path traced by the pivot *B* on the input link, which is a circle, while the one on the bottom-right predicts the profile traced by the coupler point *C*. The position, velocity and acceleration profiles obtained for the mechanism in the figure have been verified using the analytical loop equations for a four-bar mechanism.

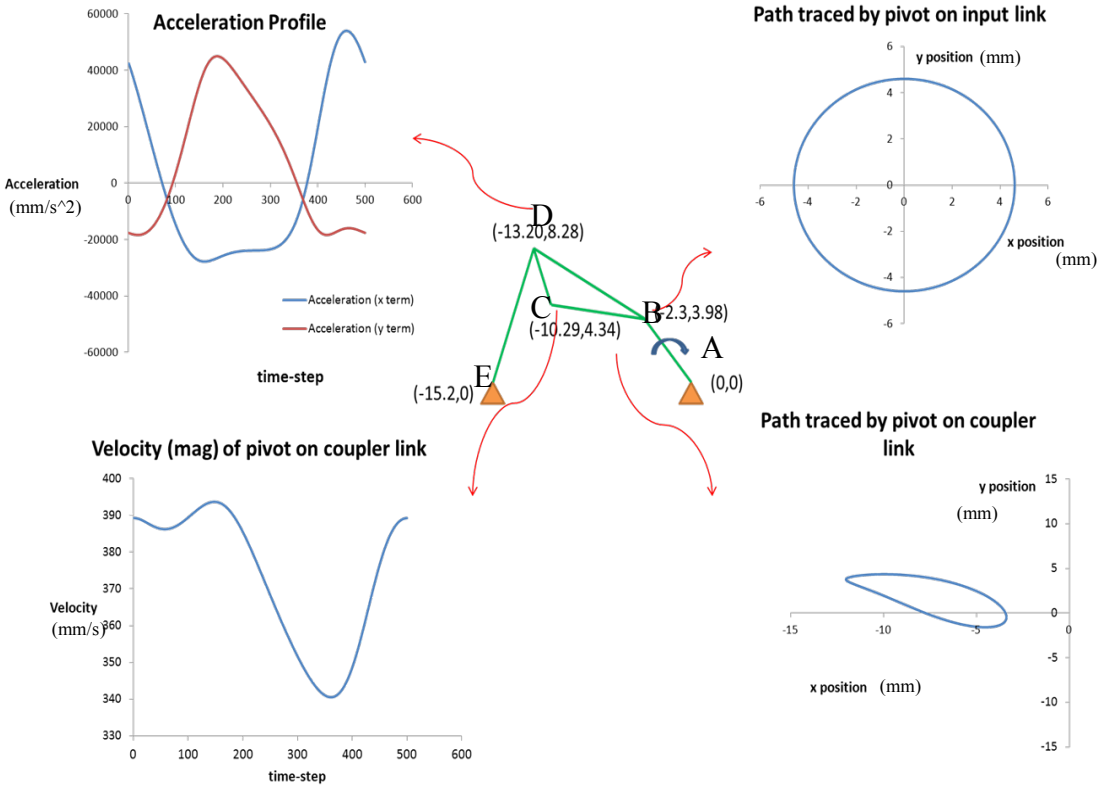


Figure 5-2: Kinematic properties of a four-bar mechanism

Similarly, the position kinematics of a quick-return mechanism is shown in Figure 5-3, where the positions of points *C*, *D* and *E* are traced for 500 time-steps. These results are verified using the procedure given in the textbook references as well as through commercial packages such as Working Model and SAM. The quick-return mechanism example demonstrates the capability of this tool in analyzing a mechanism with R, P and R-P joints. The examples shown here, though simple, demonstrate the tool's ability to analyze different topologies (different links and joints) within the same generic structure. Through a constant input angular velocity assumption, the accuracy of the implementation has been verified. It may also be pointed to the reader that significant numerical errors in Working Model affect the output values, which is eliminated in this

implementation and thereby results in an accurate prediction of position, velocity and acceleration.

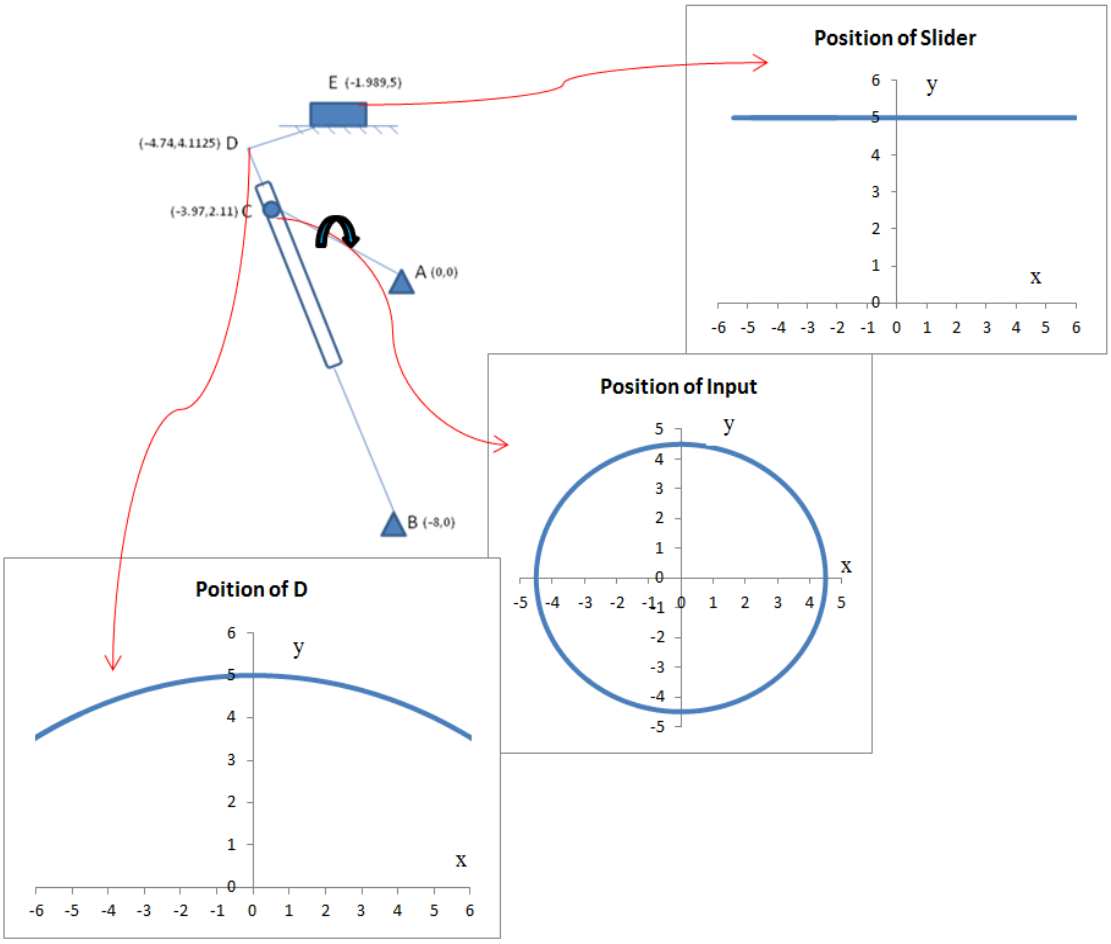


Figure 5-3: Position kinematics of a four-bar mechanism

Figure 5-4 shows the deviation in the position (defined as a ratio between the original value and the actual value obtained) of different links in a Watt-II mechanism between Working Model and our implementation. Similarly Figure 5-5 shows the variation in the input angular velocity (460 rad/s) of a four-bar mechanism in Working Model. Since this implementation is based purely on the kinematic methods shown above, it is not prone to the errors experienced in Working Model, which is really solving

the dynamics of the mechanism (forces as well as position, velocity and acceleration). This accuracy is essential for synthesizing different planar mechanisms.

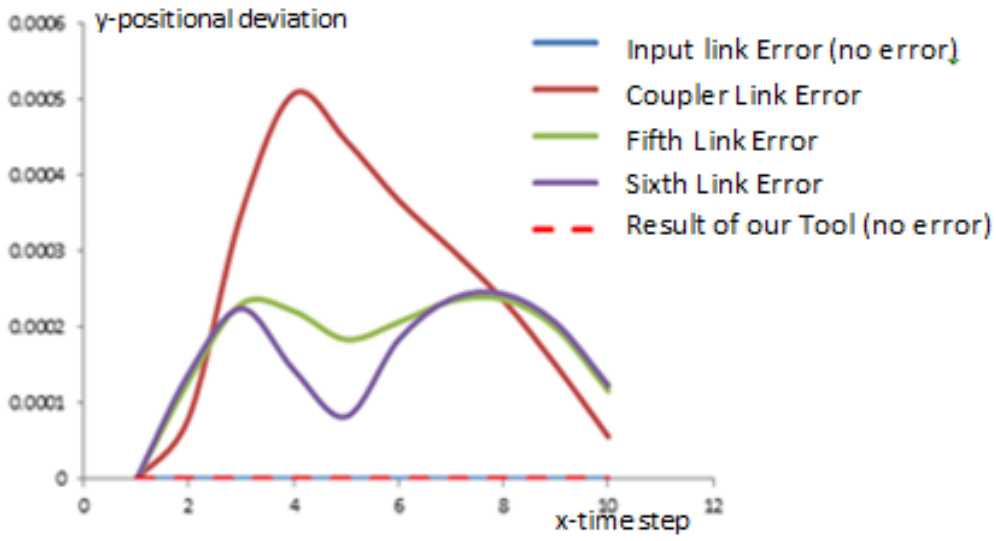


Figure 5-4: Variations in position values between results of Working Model and this implementation

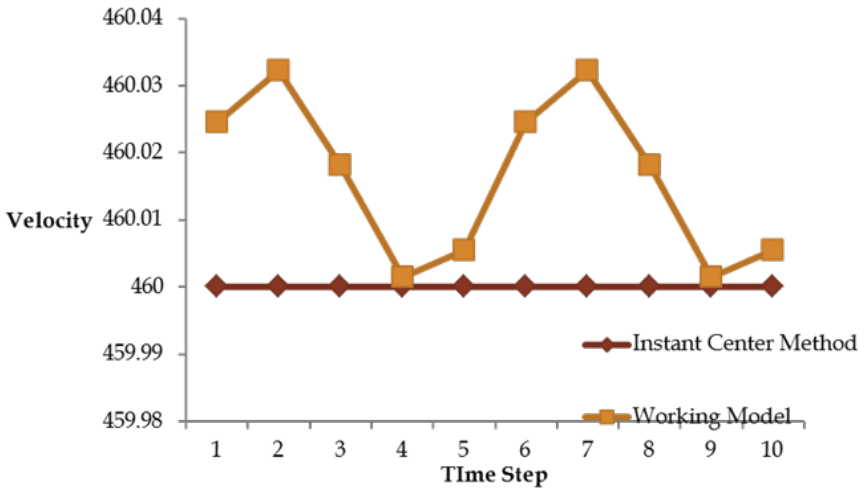


Figure 5-5: Variation in velocity values between Working Model and the instant center method in this implementation

The new methods integrated in this implementation, namely the instant center method for the double-butterfly linkage and the geometric iterative method for position analysis have also been tested. Table 5-1 displays the differences in angular velocity between the analytical solution demonstrated by Wampler [32] and the graphical instant center method [36] of all link velocities in a double-butterfly linkage at an instant in time. In this method, the result of the analytical method (from [32]) is taken as the reference and compared with solutions from the graphical method and the Working Model simulation in terms of percentage deviation from the reference value. It could be seen from Table 5-1 that the link velocities of the double-butterfly linkage (described in [36]) obtained using the graphical method and Working Model have an error of up to 10% and 8% respectively when compared to the analytical method [32]. The reason for the difference in velocities obtained using the graphical instant center method for double butterfly linkage is not known despite having tested the original instant center method extensively on mechanisms from standard textbook references. One possible way to overcome this deviation would be to derive a method based on the curvilinear locus assumption of the secondary instant center as against the rectilinear locus assumption in the new method. The error in the results of Working Model could be attributed to the numerical approximation within Working Model's simulation engine. The geometric iterative method fails for the above double butterfly linkage since the method handles finite position problems better than initial position problems. Only through such generalized implementation as in this paper, we are able to truly assess their capability and utility in automated design synthesis. Due to their inconsistencies, the graphical methods for double-butterfly linkage have not been included in our final implementation.

Table 5-1: Comparison of angular velocities of links of a double-butterfly linkage using different methods

Link	Angular velocity (rad/sec)				
	Analytical Method	Graphical Instant Method	%Deviation	Working Model	%Deviation
3	0.7432	0.7858	5.7320	0.7853	5.3576
4	1.9248	1.9819	2.9665	1.8178	-5.3989
5	0.6377	0.6759	5.9903	0.6926	8.1225
6	1.6684	1.8379	10.1594	1.5642	-5.6695
7	0.5469	0.5482	0.2377	0.5459	-0.1824
8	0.7432	0.7819	5.2072	0.783	5.0902

Despite analytical loop equations resulting in accurate solutions and being applicable to any class of mechanisms, there are no generalized implementations for solving these non-linear equations on an n-bar scale. Therefore, this generalized implementation will greatly advance the field of automated synthesis of planar mechanisms that has so far been limited to mechanisms with fewer links and joints (mainly revolute joints).

The next section will describe the new optimization based technique for solving the position kinematics of indeterminate mechanisms. Once the position kinematics is determined, the velocity and acceleration can be ascertained using existing linear loop equations.

5.3 Position Analysis for Indeterminate Mechanisms

There have been several methods developed by researchers to solve the kinematics of indeterminate mechanisms such as the double butterfly linkage and their details are available in section 2.2. The new methods show promise but there are neither generalized implementations available for [29,32,33] nor are these methods [35,37–39] simple to implement and computationally efficient. Generalization, reliability and computational efficiency are important goals in our efforts to automatically synthesize

planar mechanisms and this has led to the development of an optimization-based approach for solving position kinematics of indeterminate mechanisms.

In the optimization-based approach developed here, the lengths of different links in the mechanism are cast into an objective function, where the mean squared difference between the actual and the desired lengths is minimized. This formulation can be easily solved using Newton's method since the first and the second derivatives are analytically obtained. This method shows great promise and is also easy to implement and generalize as discussed further in this section. The length-error minimization method is described in detailed in sections 5.3.1 to 5.3.5 followed by its applicability for different types of indeterminate mechanisms in Section 5.3.6. Section 5.3.7 will highlight the benefits of this approach followed by concluding remarks.

5.3.1 Length-error minimization method

The optimization-based length-error minimization approach is based on a second order (i.e., gradient and Hessian method) method commonly referred as Newton's method. The overall process is illustrated using the flowchart in Figure 5-6. A walkthrough of the flowchart will be followed by a detailed explanation using an example of the Stephenson II mechanism. The algorithm begins with the specification of the known pivot positions (ground and input) and unknowns in the mechanism by the user, which is followed by the formulation of the objective function. The objective function is a length-error minimization function where the gradient (∇f) and the Hessian (H) are analytically computed. There are two kinds of start vectors used; one for the finite position problem where the pivot positions at time t are used to obtain the positions at time $t+1$ and the other being random pivot positions for the initial position problem where information regarding lengths of all links are available. The Newton method commences with the calculation of the perturbation vector, δ, which is a product of the inverse of H and ∇f of the objective function (which is described below). This vector is then passed onto an optional golden section routine, which is employed to determine if

the perturbation is too large. If it is, then the golden section method reduces the perturbation magnitude to prevent instabilities. From this perturbation, a new candidate state $\vec{x}_{new}$ is determined and its objective function is calculated. The new direction vector is subtracted from the start vector, $\vec{x}_{old}$ and the value of the objective function with these new positions is calculated. If the specified convergence criterion is met, then the values within $\vec{x}_{new}$ define new positions for the pivots. If that criterion is not met, the cycle repeats. There may be cases where the maximum number of iterations is exceeded in which case, the mechanism cannot be assembled in the given configuration while solving the finite position problem. If the maximum number of iterations is exceeded while solving the initial position problem, a different randomized start vector will be used and the process continued. This algorithm will now be explained in detail using the Stephenson II mechanism shown in Figure 5-7.

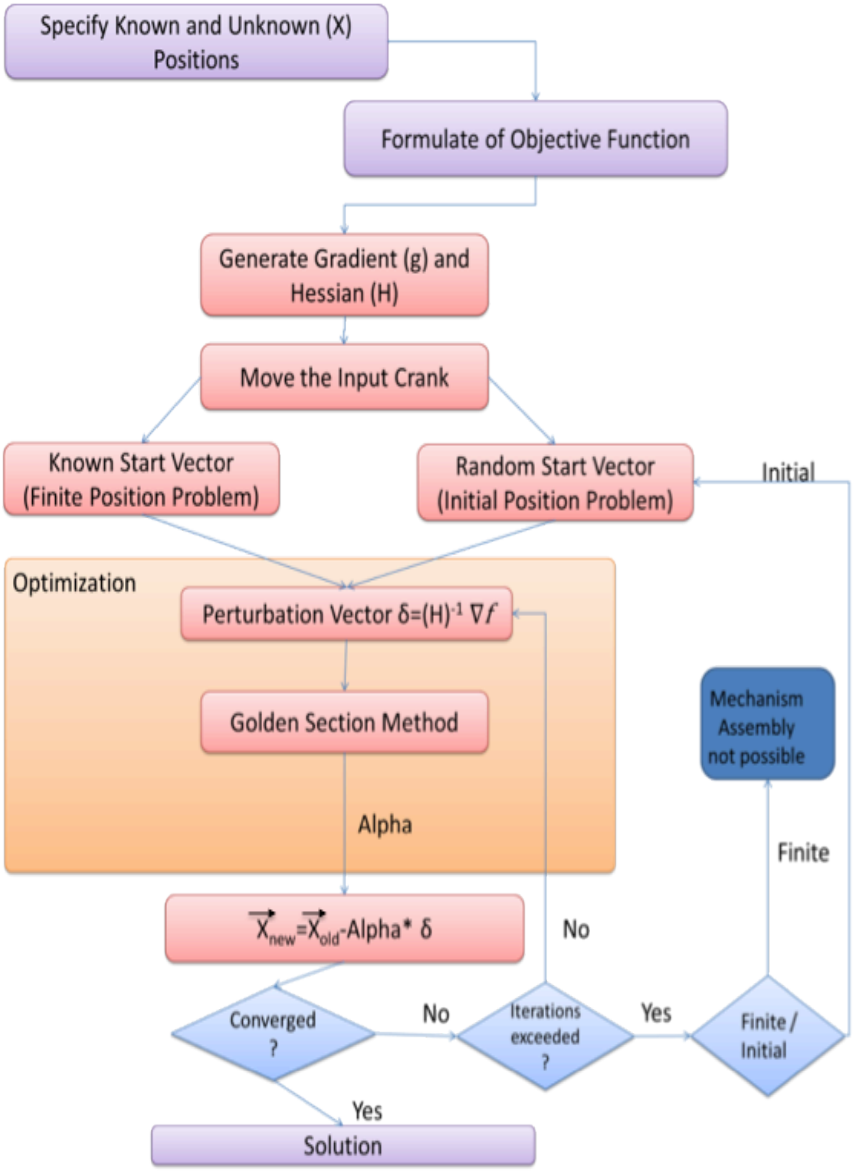


Figure 5-6: Flow chart for the optimization-based position kinematics method

5.3.2 Illustrative Example

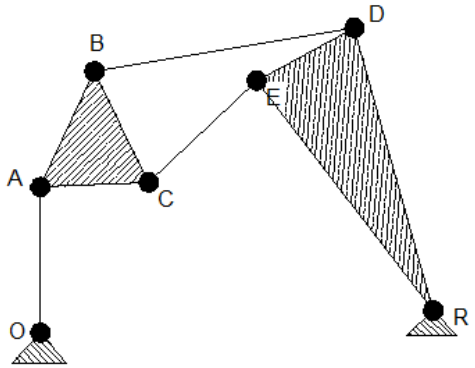


Figure 5-7 Stephenson II mechanism example

The process begins with the specification of the known and the unknown positions of pivots in terms of their coordinates (x, y). For the finite and the initial position problems, the ground pivots and the input crank are the known elements in the mechanism. Additionally, the finite position problem specifies information on the positions of the remaining pivots at a previous time step. At this time, lengths of different links in the mechanism are determined. Since the methodology is being developed for rigid bodies, there should be no change in the lengths of links at any instant. The coordinates of the pivots for the Stephenson II mechanism shown below are listed in Table 5-2 and the lengths between all pairs of pivots connected by known binary or ternary links are listed in Table 5-3. The pivots whose positions are known throughout the process are O, R and A. The other pivots namely B, C, D and E have their positions known at time t and the algorithm is used to determine their subsequent positions. The initial coordinates for pivots B, C, D and E will be considered as the starting vector for the finite position problem. As the input link OA is rotated, the corresponding positions of the pivots B, C, D and E will change. For the initial position problem, the user is required to specify the grounds and the input as before along with the lengths of various links. The initial starting vector is randomly chosen in this case.

Table 5-2: Pivot positions of the Stephenson II mechanism shown in Figure 5-7

Pivot	Coordinate
O	(0.18, -6.65)
R	(10.1390, -5.9360)
A	(0.0, -2.751)
B	(1.3970,0.2370)
C	(2.7960, -2.6260)
D	(8.1040,1.3580)
E	(5.5810,0.0)

Table 5-3 Lengths of different links in Stephenson II mechanism

Link	Length (indicated using variable names)
OA	$L_1 = 3.903$
OR	$L_2 = 10.1641$
AB	$L_3 = 3.2984$
AC	$L_4 = 2.7987$
BD	$L_5 = 6.800$
BC	$L_6 = 3.1865$
CE	$L_7 = 3.8278$
DE	$L_8 = 2.8653$
ER	$L_9 = 7.4841$
DR	$L_{10} = 7.5726$

5.3.3 Objective Function Formulation and Derivatives

The next step in the process is to formulate the objective function, which is a length-error minimization function of $2n$ variables, where n is the number of unknown joints (collectively referred to as $\vec{\mathbf{x}}$). In the test case, n is 4 and there are 8 variables to solve (x_B, y_B, x_C, y_C, x_D, y_D, x_E, y_E). The ground pivots O and R and the pivot A connected to the input link are the known parameters and not part of the optimization. The terms in the objective function correspond to the lengths of links where one or more pivots of the

link, l_k, may be unknown. The number of unknown lengths is denoted by m. In the test case, m is equal to 8 (AB, AC, BD, BC, CE, DE, ER, and DR). Therefore, the objective function is the sum of the squared-difference in the actual length, l_k, and the distance between candidate points:

$$\min f(\vec{\mathbf{x}}) = f\left(\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ \vdots \\ y_n \end{bmatrix}\right) = \sum_{k=1}^{m}\left(l_k - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right)^2 = \sum_{k=1}^{m} D_{ij}$$

For simplicity of notation in the remaining derivation, each squared term is indicated as $D_{ij.}$ As an unconstrained optimization problem, this equation alone could lead to acceptable results. While using optimization to solve a system of equations seems imprudent (as opposed to any non-linear equation solving approaches such as root-finding), the squaring of the entire term in D_{ij} leads to a well-behaved, smooth and locally convex objective problem. Furthermore, the expression is readily and analytically differentiable which drastically improves our ability to employ optimization. Methods to solve uni-modal non-linear objective function spaces are strongly dependent on the quality of the search direction that can be obtained. As mentioned earlier, a pure Newton's method can be employed without requiring a numerical approximation of the first (∇f) and second derivatives (H).

For each term, D_{ij}, in the objective function, the partial derivative with respect to x_i can be expressed as:

$$\frac{\partial D_{ij}}{\partial x_i} = 2(x_i - x_j)\left(1 - \frac{l_k}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}\right)$$

This derivative with respect to x_j yields the same result- only negative and similar equation holds for y_i, and y_j as well:

$$\frac{\partial D_{ij}}{\partial y_i} = 2(y_i - y_j)\left(1 - \frac{l_k}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}\right)$$

Obviously the derivative is zero with respect to all other variables in the objective function, *f*. With this analytical result, the gradient can be exactly calculated for all values of *f* nearly as quickly as finding the value of *f*. Using only the gradient information in optimization to determine search direction leads to the well-known Steepest-Descent method, which is rarely the most efficient optimization method. Fortunately, the second derivative is also determined analytically, thus eliminating the need to employ quasi-Newton methods. The second derivative is the Hessian matrix and is indicated by H. The analytical equations for the terms are summarized in Table 5-4.

Table 5-4: The second derivative of D_{ij} can be expressed by the following analytical expressions

$\frac{\partial^2 D_{ij}}{\partial x_i^2} = 2\left(1 - \frac{l_k}{s_{ij}}\right) + \frac{2\Delta x l_k}{s_{ij}^3}$	$\frac{\partial^2 D_{ij}}{\partial y_i^2} = 2\left(1 - \frac{l_k}{s_{ij}}\right) + \frac{2\Delta y l_k}{s_{ij}^3}$
$\frac{\partial^2 D_{ij}}{\partial x_i x_j} = -2\left(1 - \frac{l_k}{s_{ij}}\right) - \frac{2\Delta x l_k}{s_{ij}^3}$	$\frac{\partial^2 D_{ij}}{\partial y_i y_j} = -2\left(1 - \frac{l_k}{s_{ij}}\right) - \frac{2\Delta y l_k}{s_{ij}^3}$
$\frac{\partial^2 D_{ij}}{\partial x_i y_i} = \frac{2\Delta x \Delta y l_k}{s_{ij}^3}$	$\frac{\partial^2 D_{ij}}{\partial x_i y_j} = -\frac{2\Delta x \Delta y l_k}{s_{ij}^3}$
where $s_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$; $\Delta x = (x_i - x_j)$; $\Delta y = (y_i - y_j)$	

In the case of the Stephenson II mechanism, the gradient has 8 elements each comprised of two or three terms. For instance, the lengths AC, BC and CE are related to joint C and thus the gradient has three terms from the three relevant D_{ij} expressions. The second derivative is an 8-by-8 symmetric matrix.

5.3.4 Perturbation Vector and Golden-Section Search

As indicated by the Newton method, the gradient and Hessian are used to determine the perturbation vector, d.

$$H \times \delta = \nabla f$$

The new value for the variable, $\vec{x}_{new}$ is found from

$$\vec{\mathrm{x}}_{new} = \vec{\mathrm{x}}_{old} - \delta$$

This is iteratively determined until a value of $\vec{\mathrm{x}}$ is found where $f(\vec{\mathrm{x}})$ is insignificantly close to zero (a value of 10^{-9} is used in the experiments shown here). The "Converged" box on the flowchart in Figure 5-6 indicates this condition. In fact, other convergence criteria are also provided in order to prevent cases of divergence (e.g. if a maximum number of iterations is exceeded) or stagnation (e.g. no continual reduction in the value of f).

An additional step that is part of the optimization approach is the Golden Section line search. This is used to reduce the step taken by the perturbation vector, δ. Given that quick changes can exist in the objective function space, we are concerned that blindly accepting the move might inadvertently lead to a worse solution as is shown in Figure 5-8. Therefore, if the perturbation vector leads to better solutions than that at the former position ($f(\vec{x}_{new}) < f(\vec{x}_{old})$) and of the two intermediate positions $\vec{x}_1$ and $\vec{x}_2$, then the change is accepted. If it is not better, than the iterative Golden Section algorithm commences to find the local minimum. This routine adds robustness to this method.

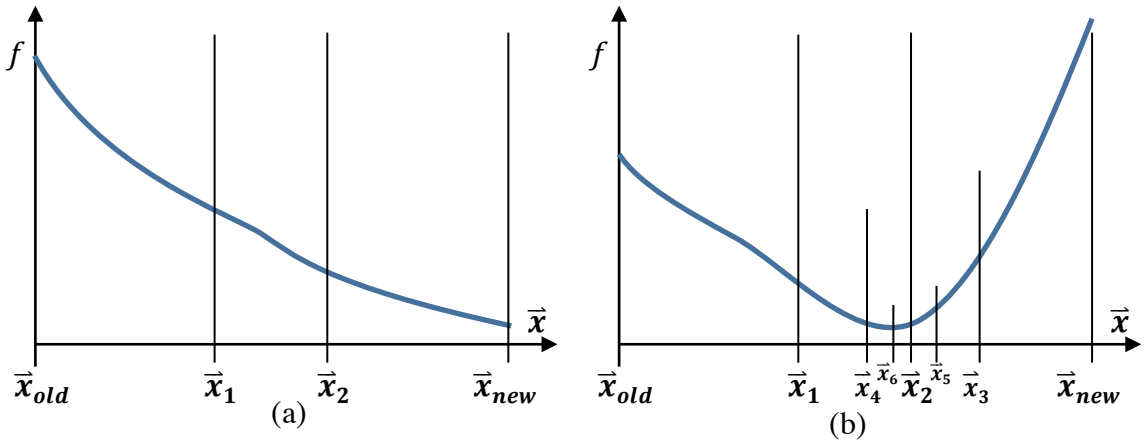


Figure 5-8: An example of how Golden Section line search is used. In case (a), the perturbation (between $\vec{x}_{new}$ and $\vec{x}_{old}$) is sufficient, but in some cases as in (b) the predicted perturbation may lead to a worse solution ($f(\vec{x}_{new})>f(\vec{x}_{old})$. By recursively finding the golden sections, a local minimum can quickly be found.

5.3.5 Optimization Initialization and Restart

The output of this process is an optimal vector, $\vec{x}^*$, which is comprised of the individual x and y positions of all unknown pivots in the mechanism. This entire process is then repeated for each position of the input crank (discretized by a specified angle; usually 0.1° or 1°). The start vector for the finite position problem is the last calculated position (for the last input angle). Given that the change in the input angle is small, the optimization rarely needs more than two or three iterations to find the subsequent positions with a high degree of accuracy. This is validated in our experiments shown in section 5.3.6. As mentioned above, the approach can also be used to solve initial position problems. In this case, the starting vector, $\vec{x}$, is randomly defined with values in the range of the lengths provided. This only occasionally leads to a candidate solution without an acceptably low value of f. The approach then continues to try new random starting vectors until an acceptable value is found.

5.3.6 Results

The algorithm explained in the previous subsection has been tested on several one-degree of freedom mechanisms. These mechanisms have been subject to both finite and initial position testing. The algorithm for these experiments is coded using Visual C# and makes use of the Object Oriented Optimization Toolbox [62] that is available as an open-source tool. The planar mechanism code also consists of a generic routine to determine the gradient and the Hessian of objective functions of any given mechanism consisting of revolute joints.

5.3.6.1 Finite Position Problem

The solution to the finite position problem is a multitude of positions that results in an overall path for each of the pivots in the mechanism. The various paths plotted in the respective figures are of those pivots not connected to ground or the input crank. The algorithm is tested by stepping the input link by 0.1° and 1° increments for each of the five mechanisms. This section lists the results from the Stephenson II example shown above as an illustrative example and an eight-bar mechanism known as the Single-flier mechanism. Appendix A then shows similar results for two double-butterfly mechanisms (eight-bar mechanism) and a ten-bar mechanism.

5.3.6.2 Stephenson II Example

The Stephenson II mechanism shown in Figure 5-6 is used to illustrate the solution to the finite position problem using our algorithm. OA is the input link and O and R are the ground pivots of this mechanism. The coordinates of the pivots are listed in Table 5-5. The results of the algorithm on the Stephenson II mechanism are shown in Figure 5-9, where the paths traversed by pivots B, C, D and E are displayed.

Table 5-5: Pivot positions of the Stephenson II mechanism for the finite position problem

Pivot	Coordinate
O	(0.1800, -6.6500)(input CW)
R	(10.1390, -5.9360)
A	(0.0000, -2.7510)
B	(1.3970, 0.2370)
C	(2.7960, -2.6260)
D	(8.1040, 1.3580)
E	(5.5810, 0.0000)

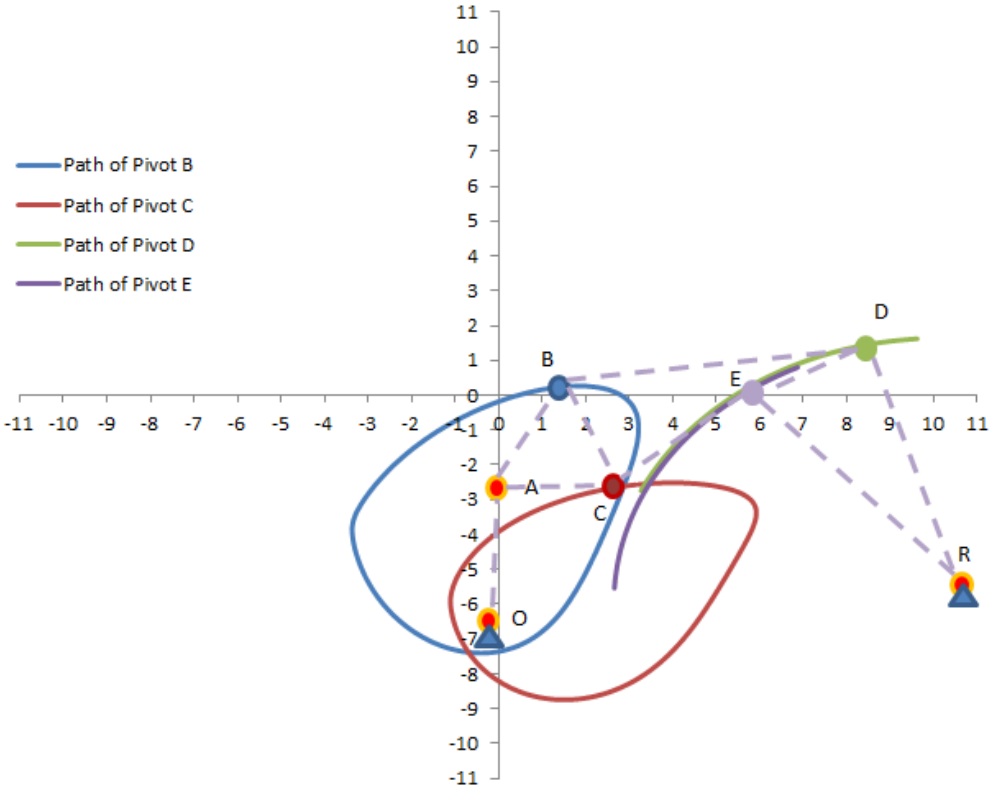


Figure 5-9: Path traversed by the four pivots (B,C,D and E) of the Stephenson II mechanism in Figure 5-7

5.3.6.3 Single-flier Example

Figure 5-10 shows the model of a Single-flier mechanism (an eight-bar, single degree of freedom system) whose pivot positions are listed in Table 5-6. Link OAH is the input link on this mechanism. The paths traversed by different pivots are plotted in Figure 5-11. Through this plot, it is clear that the input link rotates a full 360°.

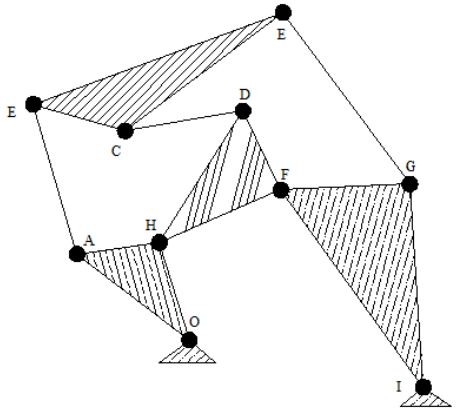


Figure 5-10: Single-flier mechanism [35]

Table 5-6: Pivot positions of the Single-flier mechanism shown in Figure 5-10 for the finite position problem

Pivot	Coordinate
O	(-1.5000, -8000) (input CW)
I	(3.0010, -8.9020)
A	(-3.6550, -6.3460)
B	(-4.4950, -3.4660)
C	(-2.7350, -3.9830)
D	(-0.4680, -3.5960)
E	(0.3030, -1.7010)
F	(0.2590, -5.1160)
G	(2.7400, -4.9990)
H	(-2.0790, -6.1230)

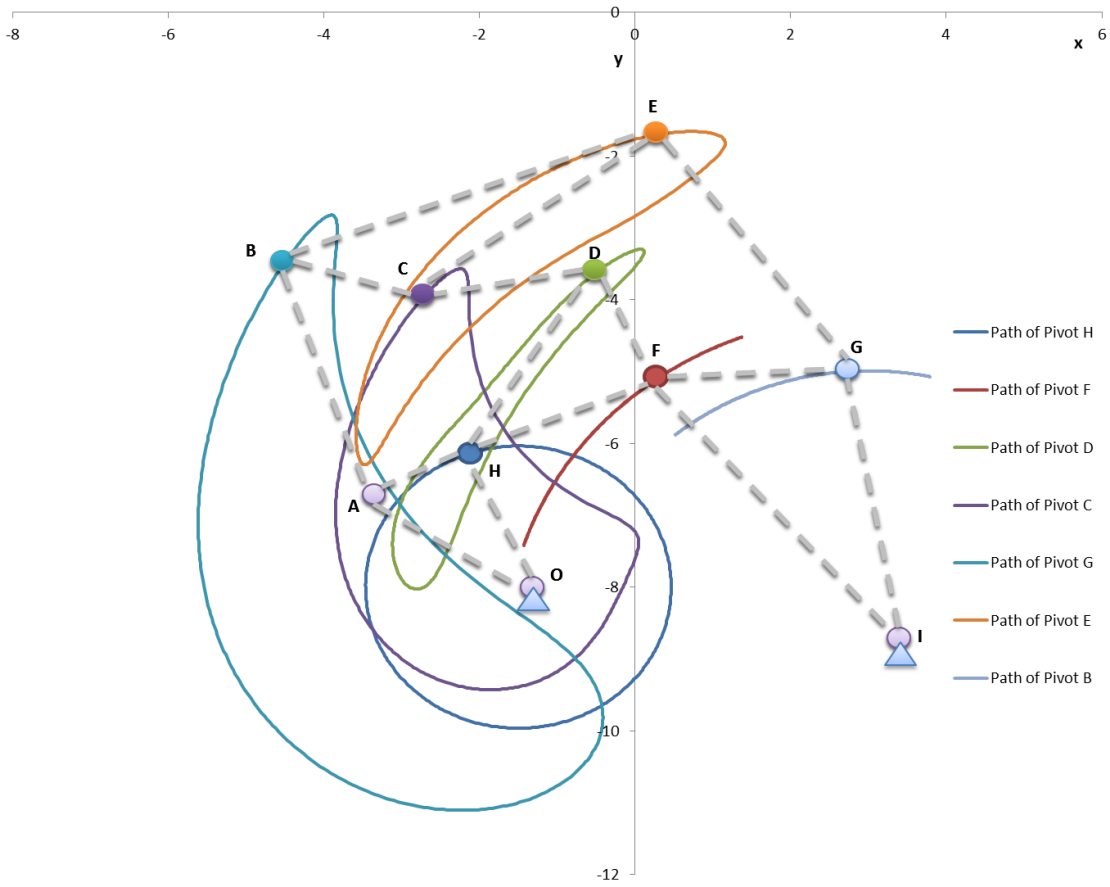


Figure 5-11: Path traversed by pivots B,C,D,E,F,G,H in the Single-flier mechanism of Figure 5-10

In order to estimate the accuracy of our method, we computed the percentage error in link lengths by comparing with actual lengths as done in axial strain (e.g. Dl / l). For the single-flier mechanism, of the sixteen lengths, fourteen are compared with the output from the algorithm using different angle increments such as 0.1°, 1° and 3°. Two lengths namely that of the input link (OA) and the ground link (OI) are not considered since they are not subject to the optimization. For an angle increment of 0.1°, the

algorithm results in a variation of 10^{-9} to 10^{-5} compared to the 10^{-4} obtained using 1° and 3° angle increments. The difference is more pronounced when the mechanism is close to a toggle position. At such locations, there are two minima in the objective function space that are close together, which could affect how the optimization progresses. An example of 10^{-4} of strain is a 10cm bar that is stretched by 10μm. It is also noteworthy that there is no error accumulating in our technique since during each step the optimization must meet the criteria for each known position of the input crank.

Upon observing how the optimization process progresses, we find that the solution is obtained in just a few iterations. For instance, for 0.1° increment, the method requires only two objective function evaluations at every position, while four and five iterations are required respectively for angle increments of 1° and 3°. Due to the fairly few objective-function evaluations required, this method is able to compute solutions very quickly. The number of objective function evaluations for different angle increments is evaluated for a convergence criterion of 10^{-9}.

5.3.6.4 Time of Computation

The length-error minimization method was tested on these different mechanisms using a Visual C# program executed on a laptop computer with a 2.1 GHz processor and 4GB RAM. The computational efficiency is measured in terms of the clock time from start to finish. This time value is measured programmatically to achieve high accuracy (i.e. using the *stopwatch* class in Visual C#). It is surprising that the total time is highest for the simplest of the four mechanisms. We conjecture that this is a result of the tight interplay between the links of Stephenson II mechanism that form a four-bar. The optimization is forced to solve a highly coupled problem, which requires more iterations than when solving more variables that are less coupled. Table 5-7 lists computational times for different mechanisms for the finite position problem using two angle increments, 0.1° and 1°. The angle indicated in parentheses against each mechanism in the table is the maximum permissible angle of rotation of the input crank measured from

the initial position. This is important because the total time will be less for mechanisms where the input crank is incapable of rotating a full 360°. It is clear from the table that the method produces accurate results quickly, with the first position being computed in less than 0.2s and entire mechanism (up to the permissible angle of rotation of the input crank) in about 15s or less.

Table 5-7: Speed of computation for 0.1° angle increment of the input link

Mechanism	0.1°		1°	
	Time for first Position (sec.)	Total Time (sec.)	Time for first Position (sec.)	Total Time (sec.)
Stephenson II (360°)	0.075	15.593	0.071	1.82
Single Flier 8 bar (360°)	0.069	1.973	0.198	0.579
Double-butterfly 1(75°)	0.13	3.02	0.068	3.012
Ten Bar (9.3°)	0.204	0.424	0.095	0.142

5.3.6.5 Initial Position Problem

The results of the algorithm on initial position problems are given in this subsection. As explained previously, the initial position problems require computing the joint coordinate data given different link lengths, thereby generating the assembly. The program terminates when the algorithm finds a single configuration using the same error-limits used in the finite position problem (i.e., 10^{-9}). It is important to realize that there may be multiple equally correct solutions. In the following results, two distinct solutions are displayed along with their pivot positions. Each of these solutions is obtained through the process illustrated in the flowchart given in Figure 5-6, which includes multiple restarts of the optimization – each with different random start vectors. The results for a Stephenson II mechanism are shown in Figure 5-12 and Figure 5-13 whose parameters are listed in Table 5-8 and Table 5-9 respectively. Similarly, the solutions for a single-

flier mechanism are shown in Figure 5-14 and Figure 5-15 and their respective pivot parameters are listed in Table 5-10 and Table 5-11.

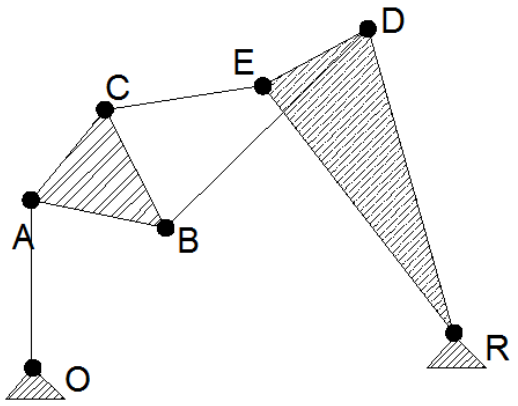


Figure 5-12: Initial position problem solution #1 for Stephenson II mechanism

Table 5-8: Pivot parameters of the Stephenson II mechanism shown in Figure 5-12

Pivot	Coordinates
O	(0.0000, -6.6500)
R	(10.1390, -5.9360)
A	(0.0000, -2.7510)
B	(1.7732, -0.5856)
C	(3.2230, -3.4196)
D	(5.5586, -0.0172)
E	(8.0765, 1.3503)

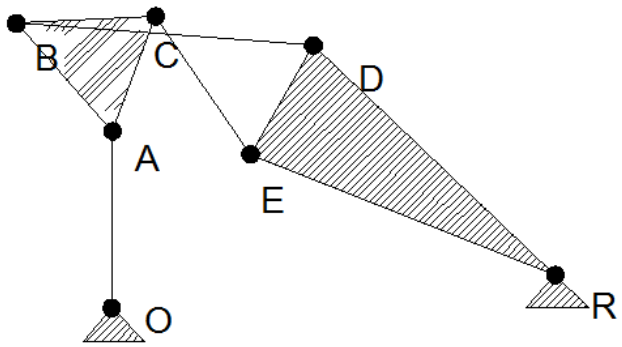


Figure 5-13: Initial position problem solution #2 for a Stephenson II mechanism

Table 5-9: Pivot parameters of the Stephenson II mechanism shown in Figure 5-13

Pivot	Coordinates
O	(0.0000, -6.6500)
R	(10.1390, -5.9360)
A	(0.0000, -2.7510)
B	(0.9833, -0.1228)
C	(-2.2032, -0.2832)
D	(3.1517, -3.2718)
E	(4.5847, -0.7875)

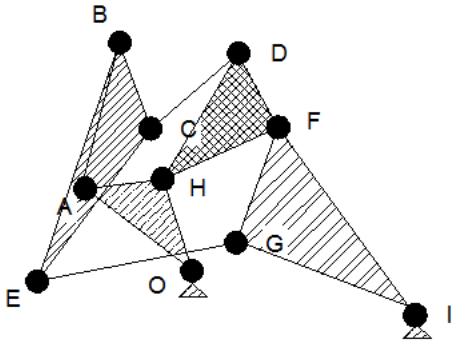


Figure 5-14: Initial position problem solution #1 for a Single-flier mechanism

Table 5-10: Pivot parameters of the Single-flier mechanism shown in Figure 5-14

Pivot	Coordinates
O	(-1.5000, -8000)
I	(3.0010, -8.9020)
A	(3.6550, -6.3460)
B	(-2.9589, -3.4035)
C	(-2.3330, -5.1366)
D	(-0.5629, -3.6164)
E	(-4.6219, -8.2149)
F	(0.2386, -5.1102)
G	(-06114, -7.4352)
H	(-2.0927, -6.1534)

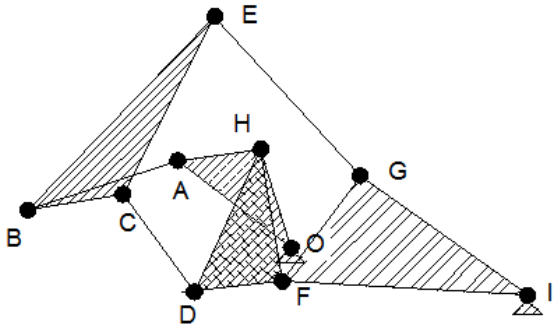


Figure 5-15: Initial position problem solution #2 for a Single-flier mechanism

Table 5-11: Pivot parameters of the Single-flier mechanism shown in Figure 5-15

Pivot	Coordinate
O	(-1.5000, -8000)
I	(3.0010, -8.9020)
A	(3.6550, -6.3460)
B	(-6.5056, -7.2841)
C	(-4.6950, -6.9849)
D	(-3.3384, -8.8421)
E	(-2.9522, -3.6080)

Table 5-11 continued.

Pivot	Coordinate
F	(-1.6661, -8.6349)
G	(-0.1890, -6.6381)
H	(-2.0789, -6.1229)

5.3.6.6 Other Mechanisms

The other mechanisms that have been tested under the finite and initial position analysis methods include the double butterfly linkage and the ten-bar linkage whose results can be found in Appendix B.

5.3.7 Discussion

The length-error minimization method presented here is based on an objective, which is a function of *x*- and *y*-coordinates of a planar mechanism. This is an alternative formulation to the simultaneous non-linear loop equations presented in other works where angles are solved instead of coordinates. Furthermore, an optimization approach is used to solve this function since it is well suited and easy to solve by optimization. This is because the objective function is nearly convex (given the profusion of quadratic terms) and the gradient and the Hessian can be obtained analytically – all of which are rare and fortuitous in engineering optimization. In addition, the use of the golden section method when the perturbation vector suggested by Newton's method is flawed adds robustness and speed to the process. It is interesting to note that this method makes use of twice the number of variables in the objective function in comparison to other methods in the literature. This is because this formulation uses link lengths that are specified using x and y coordinates, while other formulations use trigonometric loop equations solving only for a single angle for each unknown position.

This algorithm was also compared with the results for the double-butterfly linkage example given in Porta et al. [39]. This paper was selected since the authors have stated

that their results are in agreement with other methods based on polynomial continuation. This comparison is for the initial position problem, where given the link lengths and coordinates of the known pivots, all possible configurations of the mechanism are generated. The implementation presented here was configured to generate 200 different mechanisms in as many trials where different random starting values for the unknown links are provided at the onset of the optimization. Upon analyzing those resulting configurations, the length-error minimization method was able to generate 16 unique configurations (as shown in Figure 5-16) out of which four are in agreement[1] with those presented in [39]. A histogram over the 200 trials is shown in Figure 5-17. Each of the 200 trials completes in an average of 0.258 seconds with an error of 10^{-9}. Though a direct comparison of the time for computation is not appropriate considering differences in the computation hardware as well as the convergence criterion used, the results from the reference are still presented to give a perspective of the new method's ability. The reference paper gives a figure of 0.3s and 8s respectively for the box-approximations approach and the continuation approach to generate the required number of solutions. Supposing the method in the reference paper produces all six solutions in about 0.3s, that method is computationally efficient compared to our algorithm and the polynomial continuation method. Also if their convergence criterion is increased to the levels used in this algorithm, one may expect that method's performance to be slower than what has been presented. The timing produced by our algorithm for the finite position problem is also less than 0.2s for every angular orientation of the input crank (as noted in Table 5-6). This shows the capability of this algorithm and has been equal to or better than results obtained using commercial programs.

[1] It must be pointed to the reader that in Figure 3 in [39], the figures and the corresponding angular data below are not in direct agreement and care should be taken before utilizing the data.

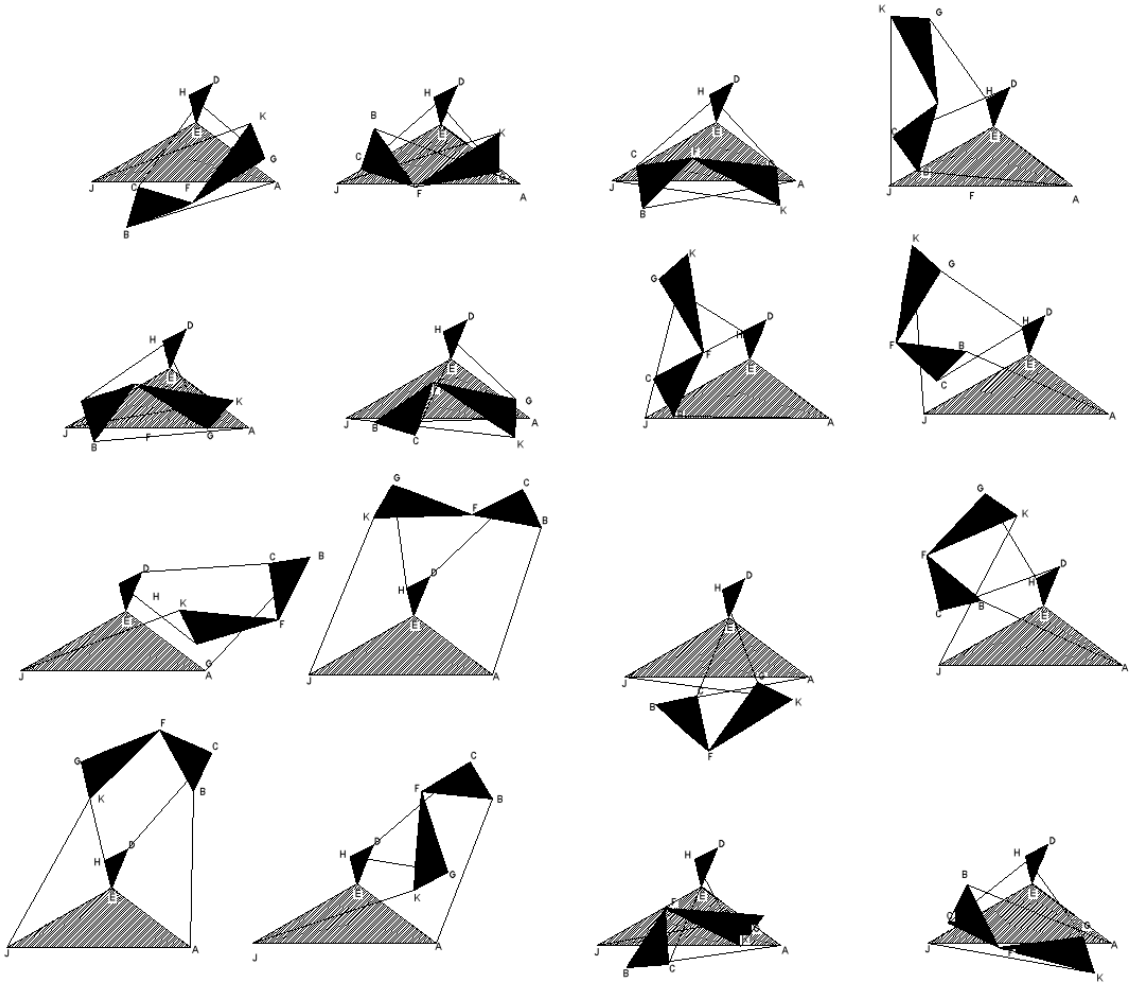


Figure 5-16: Different configurations of a double butterfly linkage generated using our algorithm

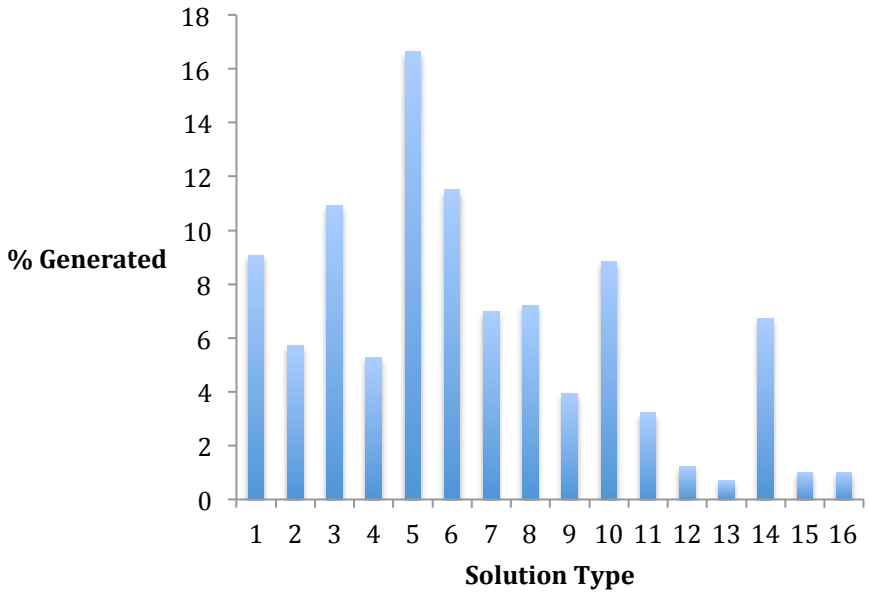


Figure 5-17: Percentage of unique configurations generated out of 200 solutions

It may seem counterintuitive that this approach generates accurate results in comparison to those methods that solve for angles since more variables must be solved. The simplistic objective function and the additional variables give this approach robustness as has been demonstrated in our results. In addition, this method is also able to generate these results fairly quickly. The method solves two types of problems; finite position and initial position and is able to generate the position kinematics of different mechanisms such as the Stephenson II, Single-flier, double-butterfly (in Appendix A and Appendix B) and the ten-bar linkage (in Appendix A and Appendix B) without any additional case-by-case tweaking. The method has also been tested on simple mechanisms like a four-bar (results not shown), which demonstrates the universality of this method.

5.4 CONCLUSION

This chapter clearly explains the development of the kinematic analysis routines applicable for one-degree of freedom mechanisms with both four-bar loops and indeterminate mechanisms with revolute joints. The implementation described has been tested against commercial software as well as results from kinematics literature and found to be accurate. The implementation is able to generate solutions quickly. This is a very important characteristic since during optimization we do not want results delayed owing to a slow output kinematic analysis tool. This kinematic analysis implementation is also available as an open-source code at http://pmksim.codeplex.com and is hosted online at http://purl.org/pmks/ through Prof. Matthew I Campbell's efforts who has not only incorporated the two implementations, but has also worked on graphics and some advanced implementations for joints such as the R-P joints.

As part of future activities, the extension of the implementations and algorithm to solve mechanisms with different joint types such as prismatic (P) and revolute-prismatic (R-P or pin-in-slot) joints and also non-dyadic components such as gears and cams are being considered. The initial position problem also produces accurate results but additional work is required to generate all possible configurations for a given position. The method shows promise for initial position problems, but it may be of interest to add constraints on the feasibility of assembling such mechanisms. Finally, the availability of a tool, such as the one described here, would benefit the mechanisms community, and will be shown in the next chapter as to how this tool is helpful in automatically synthesize planar mechanisms for solving path and motion problems.

Chapter 6: Optimization

Dimensional synthesis of planar mechanisms has been carried out using graphical and analytical methods stated in textbook references for simpler path generation problems. As the problems increase in complexity, the graphical method does not work and the analytical equation method requires solving complex non-linear equations and is tedious. In light of these difficulties and due to the increased computational capabilities currently, numerical optimization techniques have been employed to explore the search space to dimensionally synthesize mechanisms. The gradient-based numerical optimization methods involve computation of gradients that are easier to obtain in simpler problems but computationally expensive for complex problems [57] and have resulted in poor solutions based on the experiments conducted during the course of this dissertation. Researchers have used several global (or direct) optimization algorithms over several years to synthesize four-bar and six-bar planar mechanisms for different path problems. The most common algorithms used in recent times are of evolutionary nature namely Genetic algorithms [63], Differential evolution [64] or Particle Swarm Optimization [65] or a variation of these methods primarily due to the notion that these methods do not require in depth information about the search space [53]. Also, the literature has several instances (see section on related work) of using a single planar mechanism like a four-bar mechanism with revolute joints for synthesis purposes. The aim here is to use our grammar rules and kinematic analysis to generate the topology and simultaneously synthesize the parameters of several different planar mechanisms for the same application. That is, a path-tracing problem can be solved using different mechanisms such as a four-bar mechanism or a six-bar mechanism also.

The chapter is organized as follows. Section 6.1 will briefly highlight the overall process flow that will be used to generate and synthesize planar mechanisms. This will be followed by section 6.2 on objective function formulation for different problems along with the associated constraints. Section 6.3 will describe our algorithm selection

methodology that will test the benchmark problems as well as the challenge problems, whose results are given in Chapter 7. Finally, concluding remarks will be presented in section 6.4.

6.1 PROCESS FLOW FOR AUTOMATED DESIGN OF PLANAR MECHANISMS

The pseudo code for the overall process flow (refer Figure 3-2) followed in this dissertation is given below.

Input: Problem Definition
IF Problem is PATH or PATH with TIME,
Obtain PATH characteristics
SET Optimization parameters
ADJUST TIME Parameters if necessary
END IF
For Search Level 1 to N
Do
Generate all possible 1-DoF mechanisms
While (options>0)
Function: Optimize All 1-DoF using Optimization ToolBox
Generate possible locations for "output" label
Add Kinematic Analysis, Objectives to Optimization ToolBox
Optimize
Return Results to Main Loop
End Optimization Function
Store RESULTS
End For-Loop
Output: RESULTS for Levels till N

While Chapters 4 and 5 focused on design space generation and kinematic analysis respectively, this chapter is where all that work will be combined to generate meaningful solutions for different user specifications. The first step in the process as shown in the pseudo code above is to describe the problem. The problem can be either to trace a trajectory or describe a motion. In the case of tracing a trajectory, a joint in the mechanism is required to trace the desired trajectory. The joint can either be part of a

binary link or a ternary link. The benchmark problems have mostly used four-bar mechanisms with a ternary coupler link or in rare cases six-bar mechanisms with sliding members. The trajectory to be traced is usually specified in the form of Cartesian coordinates (x, y) that has to be traced by the concerned joint. The trajectory can also be time bound where the path is related to the angle of rotation of the input link. To describe a motion type mechanism, the angles followed by a link are specified.

For simplicity sake, let us consider a path-tracing problem going forward to explain the entire process. The desired path is at first analyzed to check whether "slider" nodes can be assigned the "output" label (refer Chapter 4 for related discussion). During this time, some optimization parameters will be set at this time and their details will be discussed in section 6.4. The next stage is the search process where the candidates are generated by combining different rules. One-degree of freedom planar mechanisms are segregated at every level in the search tree (refer Figure 4-7) and then passed onto the optimization routine where the actual parametric synthesis of mechanisms takes place. Within this optimization routine, the first step is to generate candidate graphs with "output" labels appended to the "pivot" nodes. Following this, random (x, y) coordinates are set for each pivot in the topology. This is different from literature where the formulation is in terms of loop equations and hence the lengths and angles are the initial specifications. The Optimization Toolbox [62] used here can incorporate custom objective functions that can be calculated based on the results of kinematic analysis for each perturbation of the design vector within the optimization algorithm. After optimization, the results are stored and the original candidate graph is passed to the main loop to generate other candidates at further levels in the search tree.

Depending on the number of levels traversed in the search-tree, the list of possible solutions (graphs that are parametrically synthesized) is collected and presented to the user on a webpage (shown in Chapter 7 Figure 7-1). Those solutions, where the error between set of points that describe the desired path and the ones synthesized by the algorithm, is close to 0 (or near optimal) is ideal since that represents the mechanisms'

ability to exactly (or nearly) satisfy the requirements of the user. The main purpose in this dissertation is to explore the design space and parametrically synthesize different topologies to solve a particular problem. Another focus area in this dissertation is also to develop an algorithm that can ensure a higher rate of obtaining near optimal solution as our experience in implementing different algorithms has shown that guaranteeing near-optimal solutions is a challenging task. Though our algorithm is able to generate more near-optimal solutions, a study of the search space (in Chapter 8) will reveal potential reasons for not being able to guarantee solutions for this class of problems.

6.2 PROBLEM FORMULATION

The path synthesis problem is formulated as

$$minimize\ \varphi(X)$$
$$subject\ to\ g_i(X) \leq 0, i = 0,1, \dots n$$

where $\varphi(X)$ is the objective or error function, whose ideal value is close to 0.0 and $g_i(X)$ refers to the different constraints ranging from none to n that are used to define the search space. The objective function that is commonly used in the literature is the sum of the squares of the distances between the points denoting the desired path and the points synthesized by the algorithm. There are also instances in the literature (refer Table 3-1) where the average distance error is used and very rarely do we find root mean square distance formulation being used. In our case, we have used the sum of the distances and the equation is given below,

$$\varphi(X) = \sum_{j=1}^{m} \sqrt{(X_{ax} - X_{dx})^2 + (X_{ay} - X_{dy})^2}$$

where X_a is the actual value obtained from optimization and X_d is the point corresponding to the desired path. Note that the (x, y) coordinates of the joint node with

"output" label is converted into a vector X and this vector is input to the optimization toolbox. So in effect

$$X = (x_0, y_0, x_1, y_1 \cdots y_n)$$

where *0,1,…n* denote the points on the desired path at different time intervals.

There are several constraints used in the literature where the ranges for various link lengths, maximum angle of rotation of the input crank, etc. are set. While these are valid design constraints especially the length since we do not want to have a very small or very large link length, we are not clear from a review of the literature as to how these constraints have been determined for most of the problems. Also, it may not be possible at the conceptual stage to arrive at these minute details. It should be noted that since the dimensional synthesis in the literature is usually limited to four-bar mechanisms, it is easy to specify constraints for link lengths. But if the topology is varied where there are several links and sliding joints, setting precise bounds for each of the links and sliding members becomes a tedious process and is not desirable or possible at the conceptual design level. Also, Grashof's criterion is valid for a four-bar mechanism (as specified in the literature) but not for higher order mechanisms. Due to this, the following two generic constraints are used,

$$g_1(X): Mechanism\ Bounding\ Box - (Max\ Width, Max\ Height) < 0$$

$$g_2(X): Space\ between\ Ground\ Pivots - Minimum\ Ground\ Pivot\ Spacing < 0$$

The first constraint sets a bounding box that will completely house the mechanism. The values for this constraint are determined based on the maximum and minimum bounds of the coordinates that describe the desired path. The logic used for setting the bounds is given below:

$$Max\ Width, Max\ Height = \begin{cases} 10{,}10 & 0 < X < 10 \\ 150{,}150 & 10 < X < 100 \\ 750{,}750 & 100 < X < 500 \end{cases}$$

The first two bounds are based on the different benchmark problems evaluated (described in Table 3-1 and further in Section 6.4) using our method and are based on the absolute values of vector X. The third bound is based on two challenge problems that will be discussed in section 6.5. It should be noted that the bounds can be varied and the resulting solutions will be different.

The second constraint sets a minimum distance between any two-grounded pivots of the mechanism. The reason for introducing this constraint is to prevent the tendency of the algorithm to gravitate towards a minimum where the grounded pivots are on top of each other (explained in Chapter 8). The use of these two generic constraints mounts a significant challenge in trying to arrive at an optimization algorithm that can be used to solve different kinds of problems. If these constraints are violated, a squared exterior penalty (p^2) term is added to the objective function. Therefore the modified objective function can be stated as

$$\varphi(X) = \sum_{j=1}^{m} \sqrt{(X_{ax} - X_{dx})^2 + (X_{ay} - X_{dy})^2} + \langle\, p^2, if\ g_i(X) > 0 \,\rangle$$

The objective function remains the same for path and path-time problems. In this work, as we are interested in evaluating several mechanisms simultaneously, the results from the kinematic analysis have to be quick and at the same time accurate. The time for geometric computations involved in kinematic analysis increases if the angle increments of the input crank are very small say 1°. Therefore, we decided to evaluate mechanisms at 10° increments of the input crank. But this would mean loss of information, especially if the kinematic analysis predicts that the mechanism is a poor candidate for the problem whereas in reality, the mechanism is tracing the path at positions other than the 10°

increments of the input. For example, consider a link that moves from position 1 to position 2 in Figure 6-1.

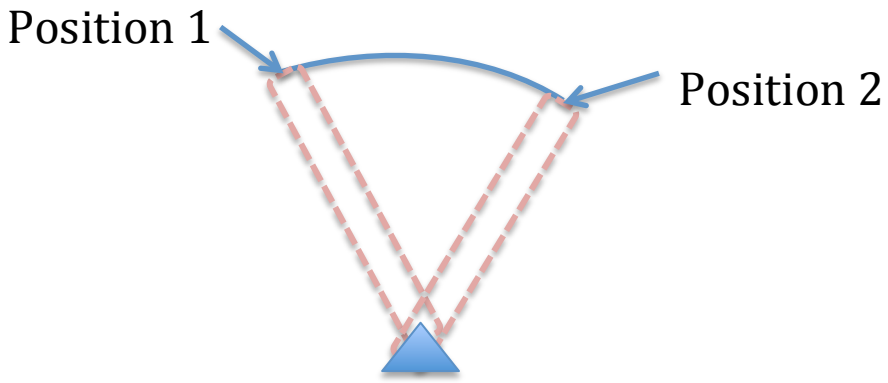


Figure 6-1 A link at two positions tracing a curve

Assume that the angular deviation between the two positions is 10°. The data (position, velocity and acceleration) is only available for those two points. If the desired path has a location that corresponds to one of the intermediate positions as shown in Figure 6-2, then we will not be able to detect the presence of those valid positions by using higher angle increments during kinematic analysis.

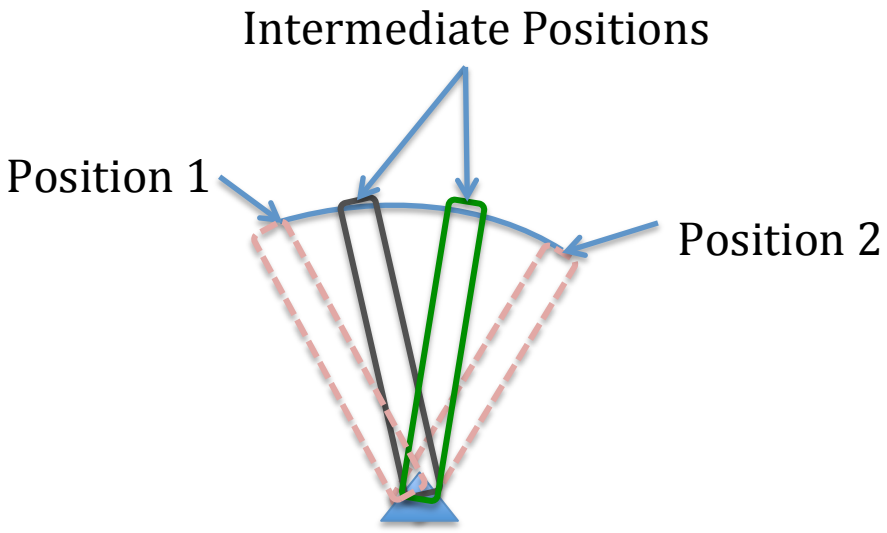


Figure 6-2: Valid intermediate positions between position 1 and position 2

In order to prevent this situation, we have incorporated a numerical approximation based on the Wilson-Theta Method [66] to estimate the kinematics at intermediate positions (i.e., between Position 1 and Position 2 shown in Figure 6-2). This numerical approximation technique is able to fairly accurately predict values for positions, velocities and accelerations since the kinematic input to this method are predicted using analytical techniques (as described in Chapter 5). Its accuracy increases based on the granularity in the approximation i.e., a coarse subdivision (say 10 subdivisions) between the two positions may result in a poor approximate compared to a fine subdivision (say 100 or 1000 subdivisions). In this dissertation, we have considered 100 subdivisions between two positions since the results are obtained fairly quickly and accurately at that level of granularity between two positions. The same procedure is used in the case of path-time problems, where the positions related to intermediate times (between times at Position 1 and Position 2) are obtained. The pseudo-code below gives an overview of the

objective function calculation for a path problem based on the results from kinematic analysis. The same has been extended to problems that are based on path and time formulation.

```
Input: X(x,y)=Kinematic Analysis Results of "output" pivot, Y(x,y)=Desired Path
FOR every Y,
      Minimum Distance =0
      FOR every X
          Distance = Distance between X and Y
          If Distance < Minimum Distance
            Minimum Distance = Distance
      END FOR Loop

     IF Minimum Distance > 0.4
        Use Wilson-Theta Method to evaluate intermediate positions
            For both PATH and PATH-TIME problems
        Compute Distances and Minimum Distances
    END IF
END FOR Loop
```

A careful observation of the pseudo code will highlight the fact that the order of the path to be traced is not enforced to be in the exact same order as the specification. This is done in order to generate those mechanisms that trace different sections of the same curve at different instances. This will be evident in the results that are presented in the next chapter.

One simplification that has been included in our formulation is the assignment of one (x, y) point from the desired set to the "pivot" node with "output" label. Since this pivot has to trace the desired path, we felt it was prudent to carry out this assignment thereby reducing the dimensionality of the problem by 2. Therefore, as soon as the "output" label is generated, this joint is assigned a default (x, y) from the desired Path. In path-time problems, where it is possible that the input angle required might not start at 0°, in those cases, the time vector is adjusted to start from 0°, without loss of any generality. Also, it may be pointed to the reader that the "output" label is assigned to sliding joints

only if the path has straight-line characteristics. That is, if the angle between consecutive points in the desired path is the same, then we can conclude that such paths can be traced using a slider. This way, unnecessary computation is avoided. There are also a few convergence criteria set into the optimization toolbox such as the maximum number of iterations and delta convergence (to exit as a result of sustained stagnation). Further details about these convergence parameters will be specified in section 6.3.

All the benchmark problems as well as the Challenge problem #1 can be subject to the above objective function. Challenge problems #2 and #3 require modification in the objective function calculation. In challenge problem #2 (ref Figure 3-4 and Table 3-4), the objective is to determine the right combination of linkages that can produce the motion of the coconut crab. To do so, we have adopted two approaches. The first is to purely consider the problem as a single-input multi-output path synthesis problem. During the generation process, rule #1 from the rule set #4 (ref Chapter 4 for related discussion) is applied on the concerned 1-degree of freedom mechanism graph after the assignment of "output" label. This way, the mechanism will have four "pivot" joints with one of the following labels: "output", "output1", "output2" and "output3". Each of these joints will trace one of the four paths specified in the problem. This means there are four objective functions that are simultaneously solved and a multi-objective formulation given below will be used to estimate the resulting error,

$$minimize \sum_{i=1}^{4} w_i \varphi_i(X)$$

$$subject\ to\ g_i(X) \leq 0, i = 0,1, \dots n$$

where $\varphi_i(X)$ is the objective function corresponding to the i^{th} desired path corresponding to each joint and w_i is the weight assigned to that objective. The constraints $g_i(X)$ are the same as the benchmark problems. Equal weights are assigned to the objective functions. The second approach is to use rule #2 instead of rule #1 from rule set 4. The idea in using that rule is that in addition to path synthesis, the two ternary links will resemble the joint

connection in the coconut crab (ref Figure 3-4), so that a better bio mimicking is achieved. Even for this case, the multi-objective formulation shown above is only used.

Challenge problem #3 is interesting due to its varied curves. The presence of several inflection points indicate that no single joint in a mechanism will be able to produce that curve. Therefore, one approach is to use the single-input multi-output scenario that was carried out in challenge problem #2, whereby rule #1 from rule set #4 is used. The second approach adopted involves a multiple mechanism approach, where different output pivots will trace different segments of the curve. In both these approaches, the desired path is split into several segments (as shown in Figure 6-3) at extreme inflection points and then each curve is independently optimized. The reader may also notice that the curve in Figure 6-3 is one half of the overall curve in Figure 3-5. Due to the symmetric nature of the original curve, the mechanisms generated for the curve shown in Figure 6-3 can be used to replicate the entire curve as shown in Figure 3-5. If the overall curve (Figure 3-5) is considered and is split into several segments, then the topological and parametric synthesis results will be different than what is presented in this results chapter.

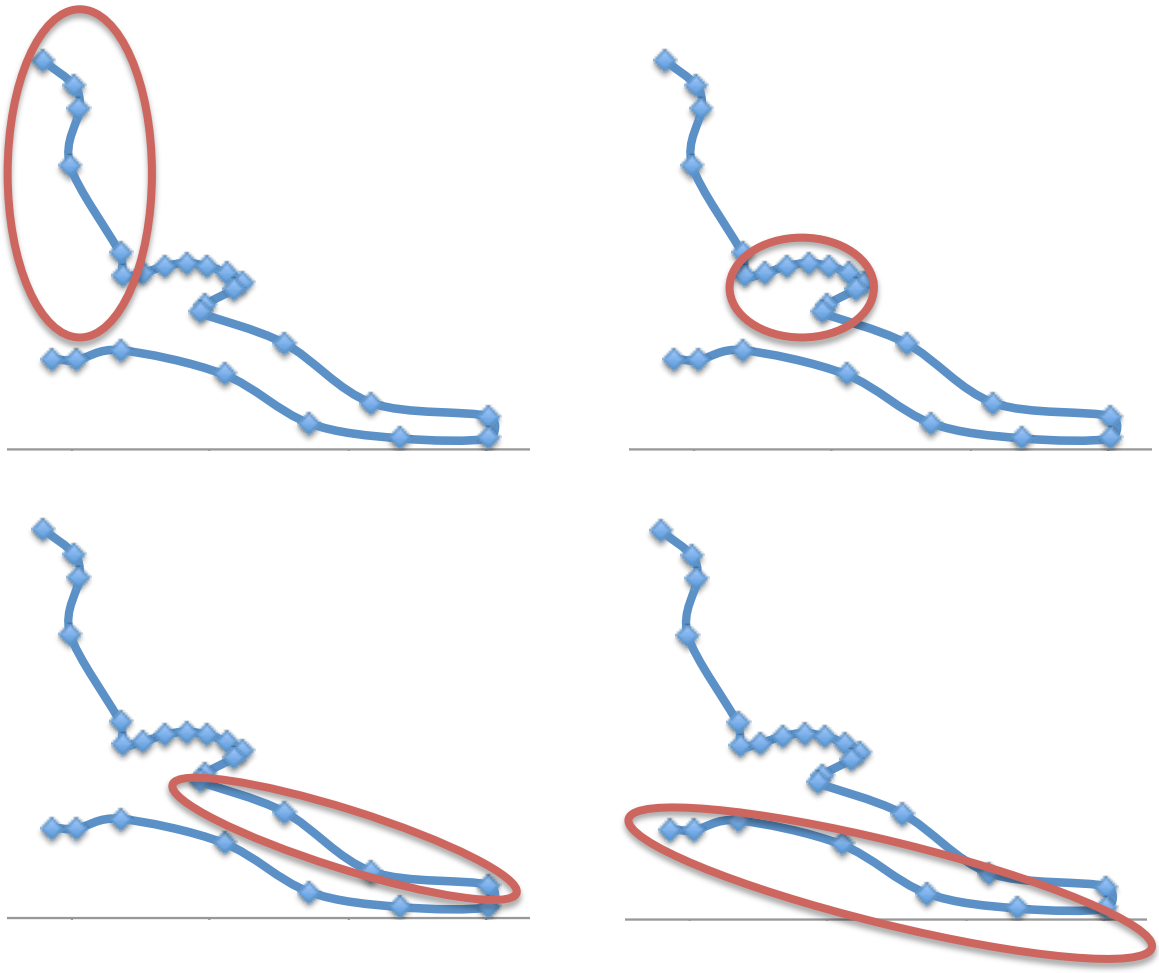


Figure 6-3: Segmentation of the curve for challenge problem #3 from Figure 3-5. Each red oval highlights a different section of the curve

Challenge problem #3 becomes a multi-objective problem with the exception that the four individual objectives are each single-input single-output cases. The results for the benchmark problems as well as the challenge problems are presented in Chapter 7.

6.3 Algorithm Development

A brief overview of the benchmark problems will be provided followed by investigations into the algorithms that are ideal for solving these mechanisms.

6.3.1 Review of Benchmark problems

Since parametric synthesis of planar mechanisms has been carried out for several years, there are a few problems, which have been subjected to synthesis using different algorithms by different researchers, classified as benchmark problems. The benchmark problems are predominantly path problems or path-time problems where the path is dependent on the angle of the input crank. The different problems are listed in Table 3-1 along with the best results for those problems. Further details of these problems are available in the respective references.

The solution for most benchmark problems is a four-bar mechanism with a ternary coupler link, with the pivot on the ternary coupler link traversing the path with or without the prescribed timing. The only exception to this is the work presented by Sedlaczek et al. [17] who have obtained results such a four-bar slider crank mechanism and two six-bar mechanisms with sliding members through a generative process in a genetic algorithm formulation. The objective function commonly used is the sum of the squares of the distances between the point traced by the mechanism and the desired point. The objective function is usually appended by a penalizing factor for violation of different constraints such as the nature of the input crank, Grashof's criterion and lengths of different links. Also, most of the benchmark problems have used some form of evolutionary algorithm for dimensional synthesis. The results obtained are impressive and essential to be replicated before proceeding to solving the challenge problems. This will give a good indication about the robustness of the algorithms being used on the challenge problems. It will also be an interesting study to present a list of alternate mechanisms to these benchmark problems taking advantage of our rule-based generative process.

6.3.2 Algorithms Tested

Based on the literature, different algorithms were tested on a few benchmark problems to check their suitability and to check if ever the solutions presented in the

literature are repeatable. The algorithms that were tested include Genetic Algorithm [63], Simulated Annealing [67], Hill Climbing [68], Nelder-Mead simplex [69], Particle Swarm Optimization [65], Multi-swarm optimization [70] and Pattern Search [71]. The implementations for Genetic Algorithm, Simulated Annealing, Hill Climbing and Nelder-Mead simplex were already part of the optimization toolbox and it was decided to use the same. Other algorithms have been coded into the toolbox using pseudo codes available in http://msdn.microsoft.com. The reason for studying different algorithms is to come up with a single formulation that has a high probability of finding solutions to any given problem. In our tests with Genetic Algorithm, Simulated Annealing, Hill Climbing, Multi-swarm and Pattern search optimizations, we were unable to obtain any good results (i.e., a near optimal solution or a trend towards the goal) for a variety of path problems. Moreover, there are no details in the literature specifying any limitations in the algorithms (apart from limitations that arise from stochastic formulations) that explains our inability to use the same algorithm (in most cases Genetic Algorithm) to attain similarly good results. Hence, it was decided to explore alternate algorithms and in the process, we were able to obtain better results with the Nelder-Mead simplex algorithm although it is widely perceived to be ideal for unconstrained problems. Nelder-Mead simplex algorithm is part of the class of global (or direct) optimization algorithms that are used to obtain a global minimum. Similarly, our studies with Particle Swarm optimization showed that the algorithm when integrated Nelder-Mead is able to generate consistently better results compared to techniques illustrated in the literature. Therefore, the next subsections will describe our tests on these algorithms and eventual usage and results.

6.3.3 Tests with Nelder-Mead Simplex Algorithm

Nelder-Mead simplex algorithm is a global optimization algorithm [69,72] that works best for unconstrained optimization problems with fewer dimensions though it has been shown (in [73]) in recent times to be scalable to problems with several dimensions. In order to test the algorithm, we took a four-bar mechanism with a known solution and

tried to obtain the same using this algorithm. Consider the four-bar mechanism with the curves traced by different pivots (B, C and E) shown in Figure 6-4 below.

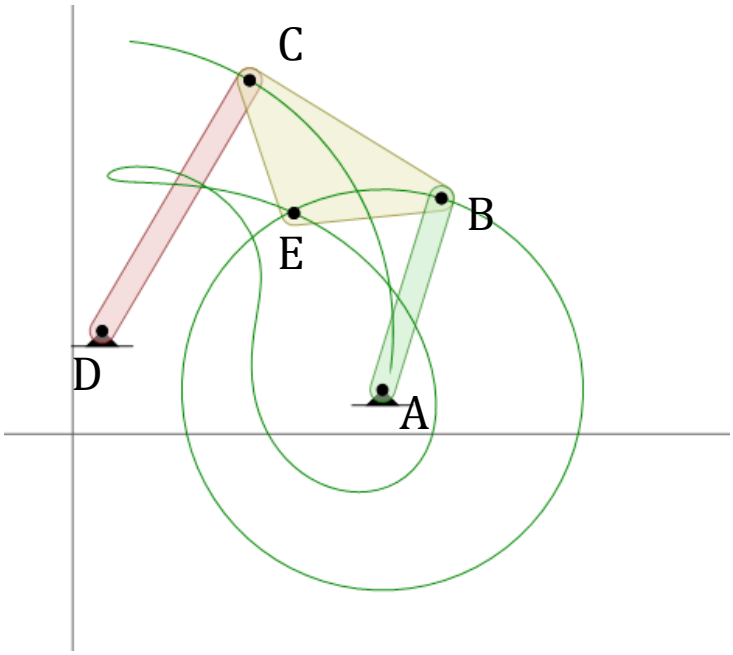


Figure 6-4: A four-bar mechanism screenshot from http://purl.org/pmks/sim

The input joint is located at A(21,3) and the other joints are located at B(25,16), C(12,24), D(2,7) and E(15,5). The coordinates of point E are extracted for every 30° as the input rotates a full 360° to be the desired path for the optimization problem. The bounding box is defined by a maximum width of 50 units and a maximum height of 50 units while the input ground has to be located at a minimum of 1 unit from other grounded joints. The goal is to test whether the Nelder-Mead algorithm is able to reproduce the original solution or synthesize a different near-optimal solution.

Since the algorithm requires a starting vector, the approach followed here is to generate a set of random vectors based on a design of experiments method called Latin Hyper Cube sampling [74]. A maximum and minimum value for each element in the vector will be specified so that the eventual vector generated through the Latin Hyper Cube sampling will fall within the specified range. This range is usually based on the

maximum and minimum values of the points that define the desired path and varies depending on the nature of this path. Generating many vectors based on this range will ensure sufficient exploration of the design space. These random vectors are kinematically analyzed and the objective function is evaluated for each case and ordered from the lowest to the highest objective function value. The vector that produces the lowest objective function is used as the starting vector for the Nelder-Mead simplex algorithm. A pseudo code for the process described is given below.

Obtain MAX and MIN number from the Desired PATH
MAX = MAX + a; MIN=MIN +a //where $a, b \in \mathbb{R}$ *and based on Desired PATH*
Set Required Number (N) based on Desired PATH
VECTOR= Latin Hyper Cube Sampling (N, Max, Min)
FOR EACH VECTOR
    Perform Kinematic Analysis
    Compare with the Objective Function
    Add to Sort List //sorted based on the least objective function
END FOR EACH Loop
Return the Best Vector from the Sort List

A random vector that was arrived using the above procedure for the problem in Figure 6-4 is

$$X = \{\, 6.29054, 12.8167, 21.48074, 22.26838,$$
$$10.51004, 6.79688, 18.21766, 7.69704 \,\}$$

While testing the algorithm, we set the maximum number of iterations to 100. The result is shown in Figure 6-5 below, where it can be seen that there is very limited change in the objective function value. The objective function value starts at a little above 49 and then only marginally reduces to 48.80. This contradicts our earlier assertion that this method worked better than other techniques in the literature. This behavior will be explained in Chapter 8.

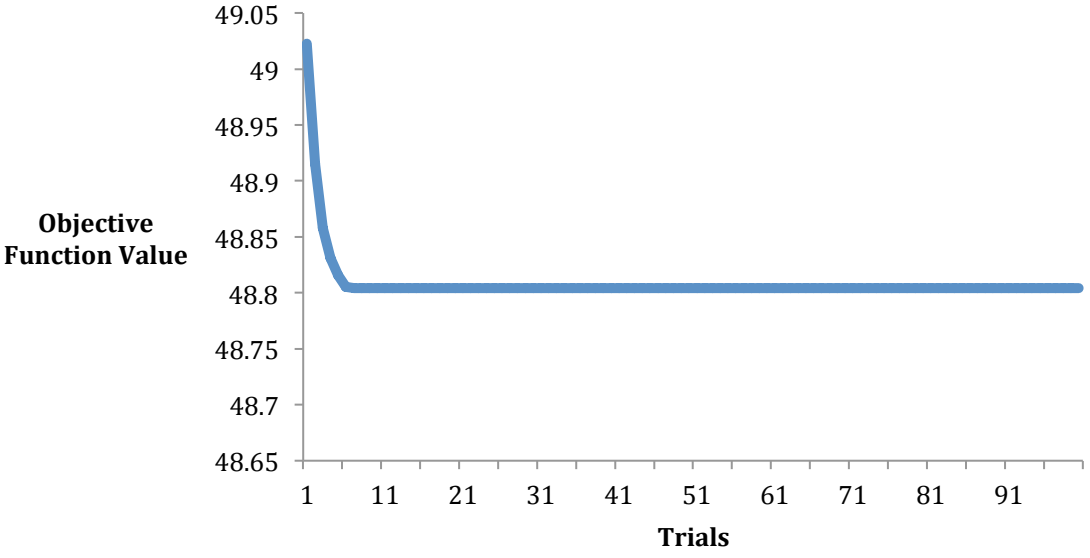


Figure 6-5: Results of the Nelder-Mead algorithm starting from vector X for problem in Figure 6-4

A similar trend is observed when many different random vectors were used. While investigating the knobs $(\chi, \psi, \rho, \sigma)$ of the algorithm that are used to generate the different simplex shapes through reflection, expansion, contraction, etc. , the work in [73] suggested using alternate values for these knobs. Those values are based on the dimension of the problem and are given below:

$$\chi = 1 + 2/n$$

$$\psi = 0.75 - \frac{1}{2 * n}$$

$$\rho = 1,$$

$$\sigma = 1 - \frac{1}{n}$$

where n is the problem dimension. In the case of a four-bar mechanism, the dimension (n) is 8 (4 pivots and each pivot is represented by (x, y)). Therefore, the knobs translate to (1.25,0.6875,1,0.88) respectively when n = 8 for our example. Using the modified knobs,

the algorithm, now referred to as the Adaptive Nelder-Mead algorithm, predicts a slight improvement as shown in Figure 6-6 below.

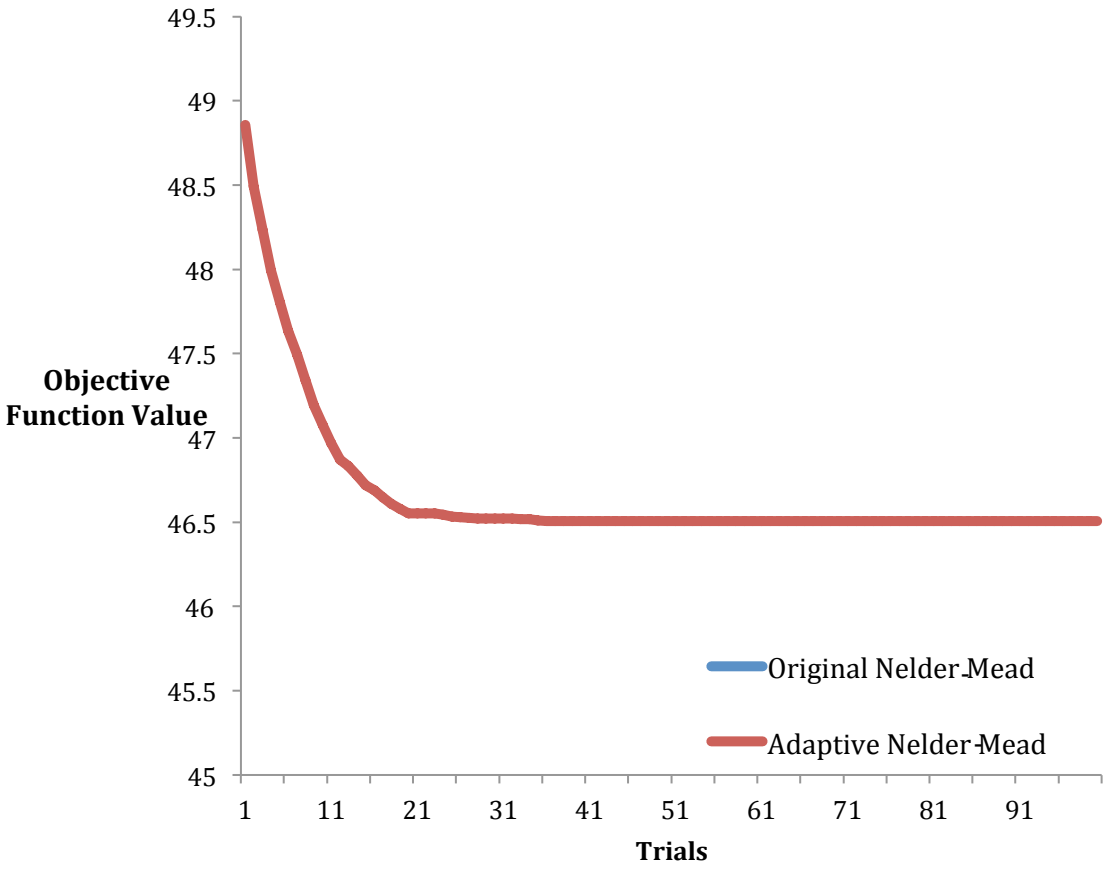


Figure 6-6: Comparison between the Original Nelder-Mead and the Adaptive Nelder-Mead methods

You may notice from the above figure that though there is a slight improvement between the two methods, the overall goal is not attained. While investigating ways to improve the performance, it was decided to do a line-search around the Nelder-Mead solution in order to jump over any local minima. Therefore, a line-search technique namely the Golden Section [75] is appended to improve the coordinates of the Nelder-Mead simplex. The Golden Section algorithm is already part of the Optimization toolbox and the only input

required is the step-size. The Golden Section routine is applied to the resulting vector of a Nelder-Mead simplex iteration so that core process is not affected by the line-search technique. This in turn has the effect of restarting the Nelder-Mead process each time. The results of appending Golden section routine to the two types of Nelder-Mead algorithm are shown in Figure 6-7 below. It can be seen from the figure that the Golden Section routine is improving both the formulations but is more pronounced in the Adaptive Nelder-Mead formulation.

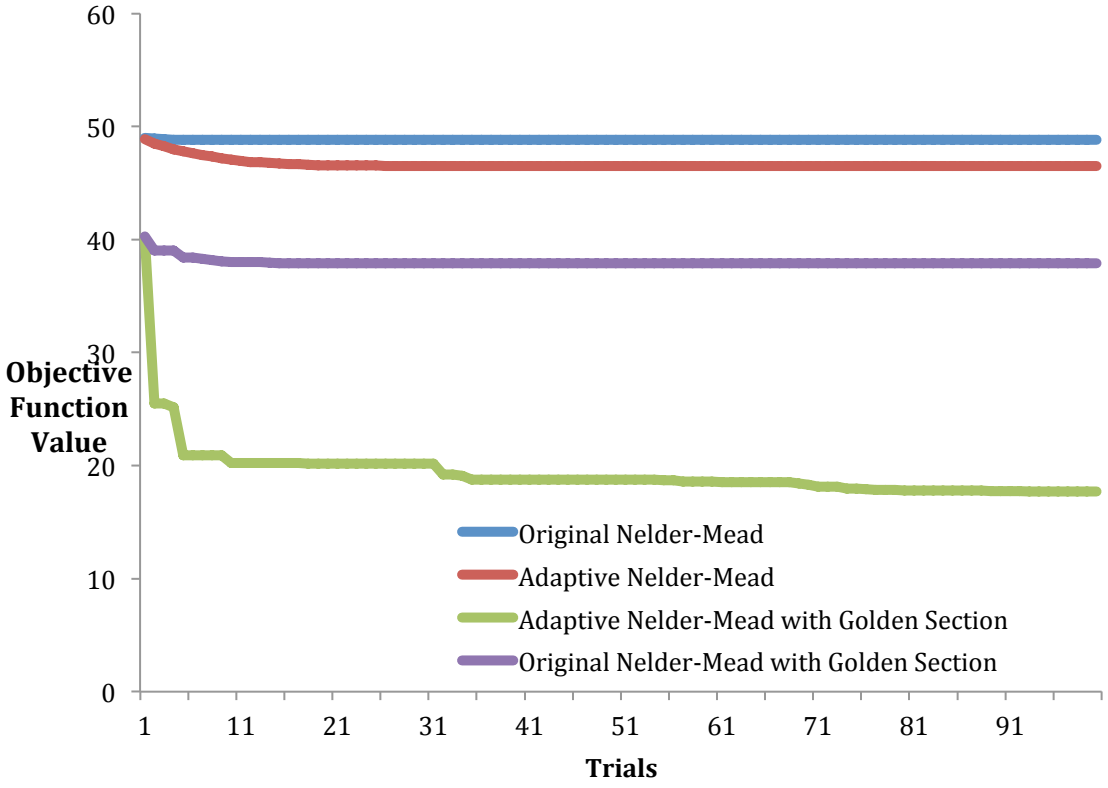


Figure 6-7: Effective of including Golden Section method as part of Nelder-Mead simplex algorithm

This trend was witnessed in several iterations using different values and we can fairly conclude that appending Golden Section to the Adaptive Nelder-Mead algorithm is

definitely beneficial. But these improvements are not sufficient in getting to the goal and hence another algorithm namely the Particle Swarm Optimization was tested.

6.3.4 Particle Swarm Optimization

The two main parameters governing Particle Swarm Optimization are the number of particles and maximum and minimum values for the parameters. The number of particles required is based on the desired path specified by the user and is given below

$$n = \begin{cases} 5, & 0 < max, min < 10 \\ 25, & 10 < max, min < 50 \\ 50, & max, min > 50 \end{cases}$$

where *max*, *min* are the maximum and minimum coordinate values in absolute terms in the desired path. This formulation ensures balance between exploration and exploitation of the search space. These values have been arrived after several testing. The algorithm also requires initializing the particles. For this purpose, we decided to assign the vector that was selected based on the Latin Hyper Cube Sampling technique to be the position of one particle. That particle's velocity along with other particles' positions and velocities are randomly assigned within the algorithm. This approach was arrived after several trials and has been able to produce consistent results.

The algorithm is tested on the problem shown in Figure 6-4 using the same starting vector used in the Nelder-Mead algorithm and one result trend is displayed in Figure 6-8 below. The number of iterations is limited to 100 in these trials.

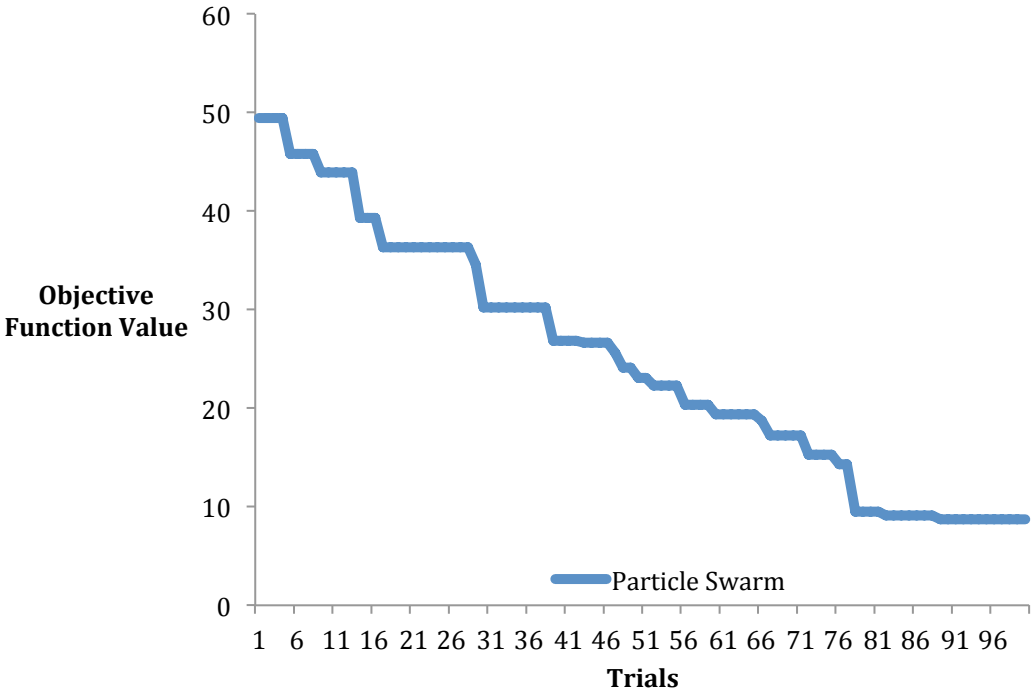


Figure 6-8: The trend in the objective function value using Particle Swarm Optimization

You may see from the above figure, that the algorithm is able to produce an objective function value that is significantly better than what was produced using the Adaptive Nelder-Mead Algorithm with Golden Section routine. But still the algorithm is unable to produce a near-optimal value. In this scenario, it was decided to combine the two algorithms with the understanding the results from the Particle Swarm Optimization will be improved by the Adaptive Nelder-Mead algorithm so that near-optimal solutions can be obtained. The result of this hybrid approach for the sample problem is shown below in Figure 6-9, where the first 100 iterations correspond to the Particle Swarm Optimization and the remaining 100 iterations correspond to the improvement using the Adaptive Nelder-Mead algorithm with Golden Section routine. The reason for limited correction using Nelder-Mead may be due to the fact that the vector that resulted from Particle Swarm Optimization is already at a local minima and the second algorithm is unable to improve much further from there.

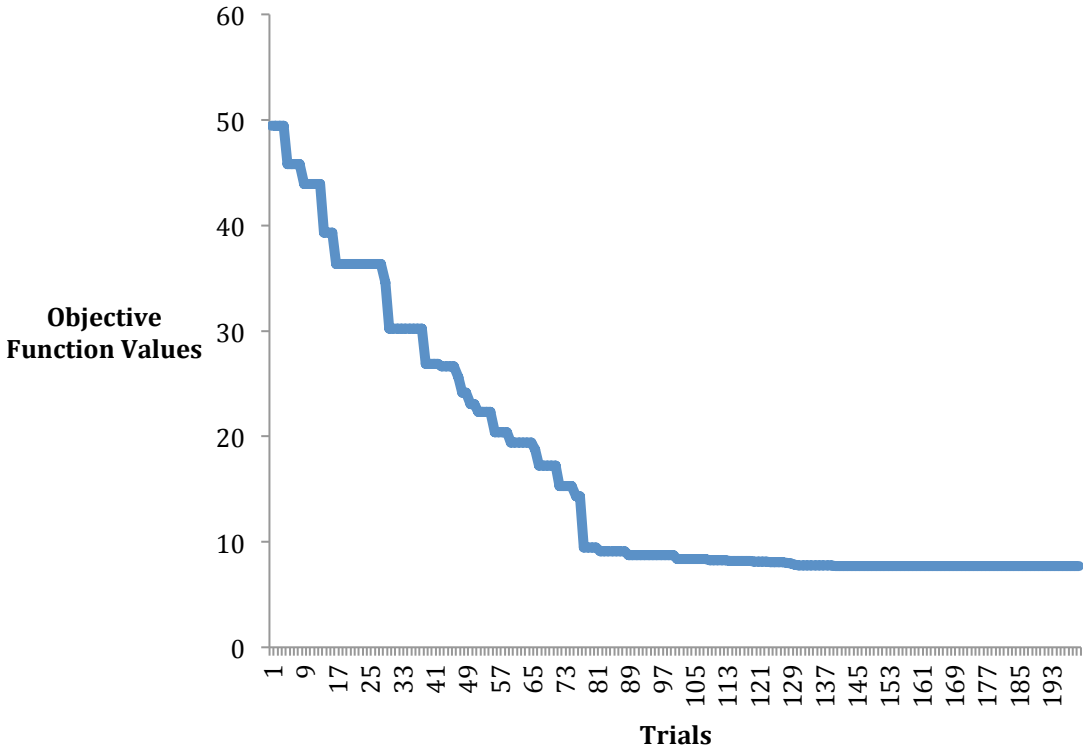


Figure 6-9: Results of the hybrid algorithm combining Particle Swarm Optimization and Adaptive Nelder-Mead algorithm with Golden Section

You may notice that the random vector and the random particles are unable to synthesize the desired curve. It should be pointed out to the reader that this output will be different for different random starting vectors X. Though the hybrid approach is unable to attain good results for the sample problem presented here, these are still better than the results that we were getting using other algorithms stated in the literature. Repeated trials were also conducted using all the algorithms mentioned on the benchmark problems and the percentage of better results was higher in this hybrid method compared to the rest and that is the reason for selecting this hybrid implementation over other algorithms. This will be evident in our results in Chapter 7. Further experiments in Chapter 8 will

highlight the limitations in the search space, which will also explain the behavior of different algorithms in the way they do for this class of problems.

The overall pseudo-code for the Automated Design of Planar Mechanisms is shown below.

```
Input: Problem Definition
IF Problem is PATH or PATH with TIME,
  Obtain PATH characteristics
  SET Optimization parameters (number of particles,n, maxWidth, maxHeight, (max,min)
from PATH)
  ADJUST TIME Parameters if necessary
END IF
For Search Level 1 to N
    Do
       Generate all possible 1-DoF mechanisms
     While (options>0)
     Function: Optimize All 1-DoF using Optimization ToolBox
         Generate possible locations for "output" label
        Assign (x,y) from the desired path to the "output" pivot //remove this from
optimization
          Add Kinematic Analysis, Objectives to Optimization ToolBox
        Generate random Numbers based on Latin HyperCube Sampling
        Sort based on the Objective Function Value
        Using the Top 2 Vectors
                Optimize
                X = Function(Particle Swarm (750 Iterations))
                X1 = Function(Nelder-Mead (100 Iterations), X)
                Store X1 to Results
          Return Results to Main Loop
     End Optimization Function
   Store RESULTS
End For-Loop
Output: RESULTS for Levels till N
```

You may notice that we have assigned 750 iterations for the Particle Swarm Optimization and 100 iterations for the Nelder-Mead algorithm. This setting was based on the improvements in the objective function value that these algorithms were able to produce over several trials. Some of the other parameters that are provided to the

optimization toolbox are listed in Table 6-1. These parameters have also been arrived after several trials and also with an intention to optimally use the available computational resources so that the results are generated quickly but at the same time include sufficient exploitation by the algorithms.

Table 6-1: Optimization Parameters

Maximum No of Iterations	750 (PSO), 100 (NM)
Maximum Age	750
Maximum Age Convergence	0.01
Delta X Convergence	0.001
To Known Best Function Convergence	0.01
Squared Exterior Penalty	10

The results are automatically organized into a HTML page that lists the configuration, the objective function value as well as a link to the actual mechanism viewable online at http://purl.org/pmks/sim.

6.4 CONCLUSION

The chapter describes the formulation of objective functions for different benchmark problems as well as the challenge problems. A hybrid implementation has been introduced by combining Particle Swarm Optimization and Nelder-Mead simplex with Golden Section line-search and basic experiments detailing their trend is shown. The different constraints are explained and so are the pseudo codes for various sections of this implementation.

Chapter 7: Results

The results of optimization to the benchmark problems and the challenge problems are presented in this chapter. Section 7.1 will present the solutions obtained using our method to the benchmark problems. Two sets of solutions will be presented. The first set will include the results of optimizing a four-bar mechanism for the benchmark problems and the second set will include a few alternate mechanisms for the same benchmark problems that have been generated using our technique. The solutions to the challenge problems will be presented in section 7.2.

The synthesis results are automatically listed on a webpage. A screenshot of such a page is shown below in Figure 7-1, where MechSynth refers to Mechanical Synthesis. The page displays the configuration, objective function value and a link to the online implementation PMKS (http://purl.org/pmks/sim) where all the parameters of the mechanism can be obtained and the user can see the mechanism in operation.

MechSynth Results:

S.No	Configuration	Objective Function Value	Mechanism Link
1	2-4-4-0--revolute-no prismatic	1.31057789639538	Click to see the mechanism
2	2-4-4-0--revolute-no prismatic	0.701851588813826	Click to see the mechanism

Figure 7-1: Snapshot of the HTML page displaying the results of optimization

For conciseness, we will present screenshots of the generated mechanism along with its characteristics such as the paths traversed by each joint (in certain cases only the path of interest is shown for clarity) in green along with the desired path in gray. Depending on the orientation of the mechanism, axis lines from the online implementation will be visible. There are instances where the mechanisms have a

challenging scale. For those cases, a comparison plot between the desired and the actual path obtained is presented. You may also notice that the number of solutions listed vary between problems. This is primarily due to computational limitations and the nature of the search space and will be further explained in the next chapter.

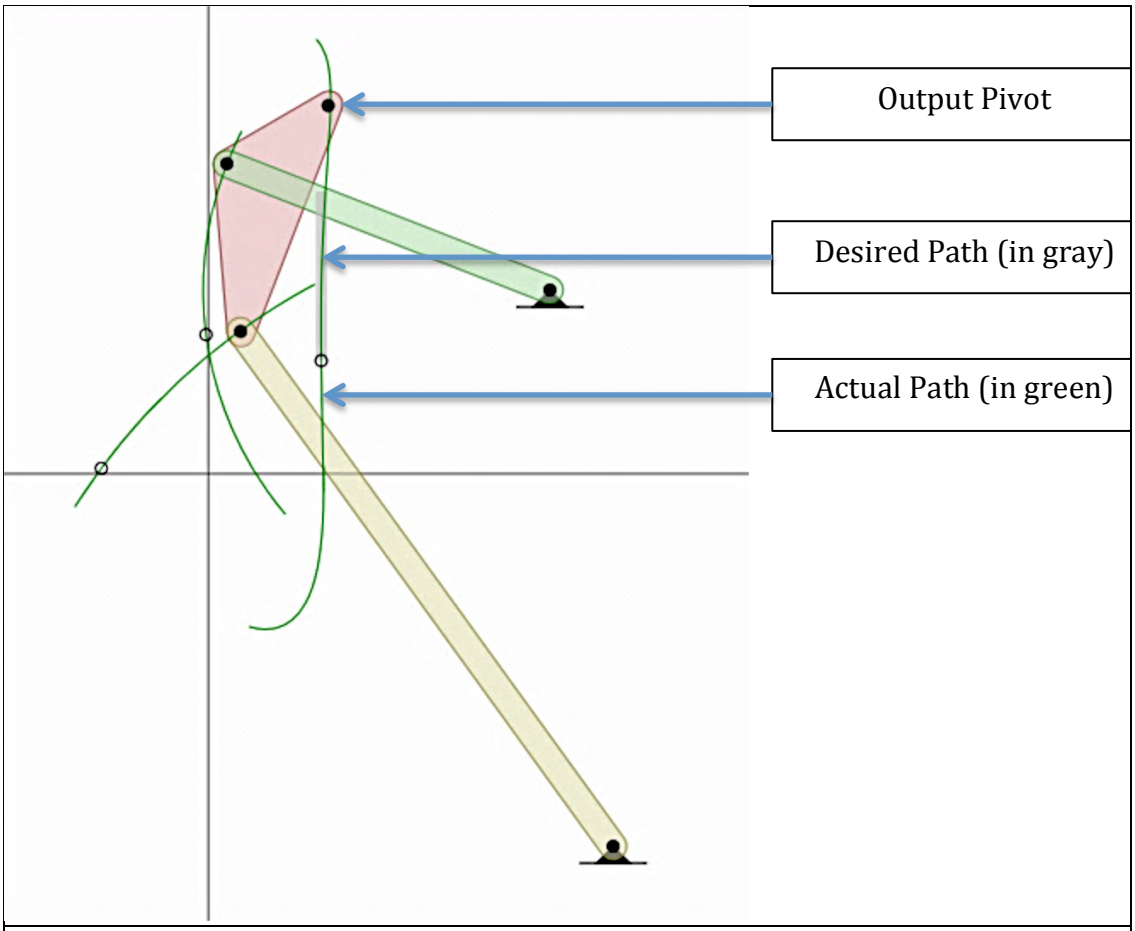
7.1 RESULTS TO BENCHMARK PROBLEMS

The following list of tables (Table 7-1 to Table 7-10) will present the solutions generated by our technique to the different benchmark problems. The desired path for the benchmark problem will be displayed followed by the synthesis results and a listing of the errors obtained in comparison to the literature. The objective function will be specified in terms of the sum of the squares of distances so as to compare with the literature.

Table 7-1: Results to Problem #1

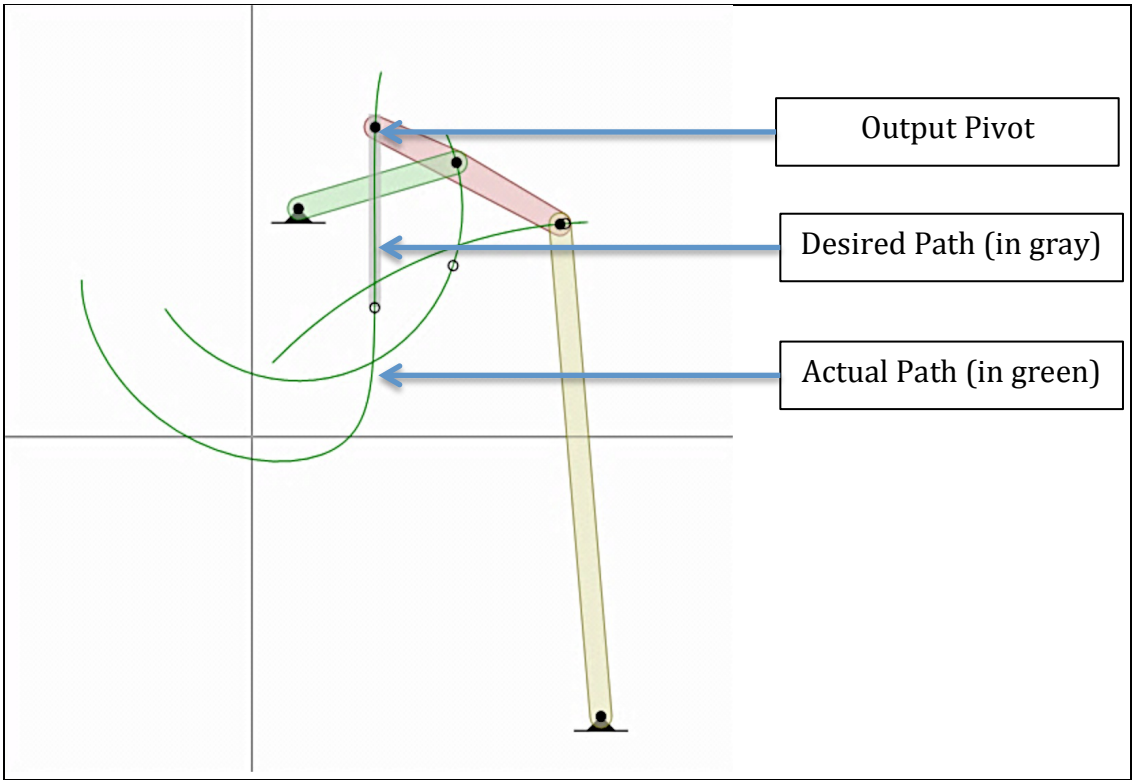
Problem 1: (20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45)
Results: Four Bar Mechanism

Table 7-1 continued.



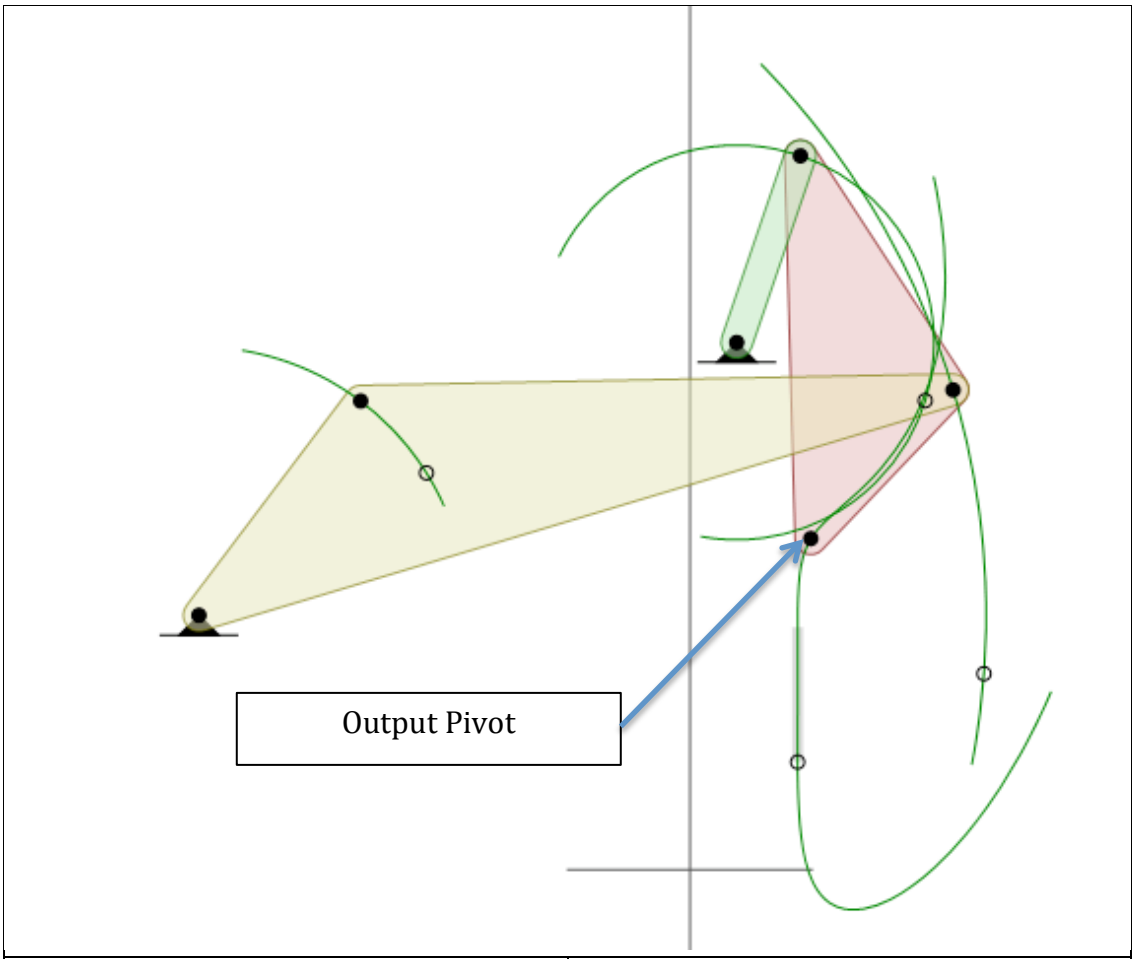
URL: http://goo.gl/dAaUZi
Objective function value: 0.02| **0.00007** (as per sum of the squares of distances)
Best results from literature: **0.0002**	**0.0178**

Table 7-1 continued.



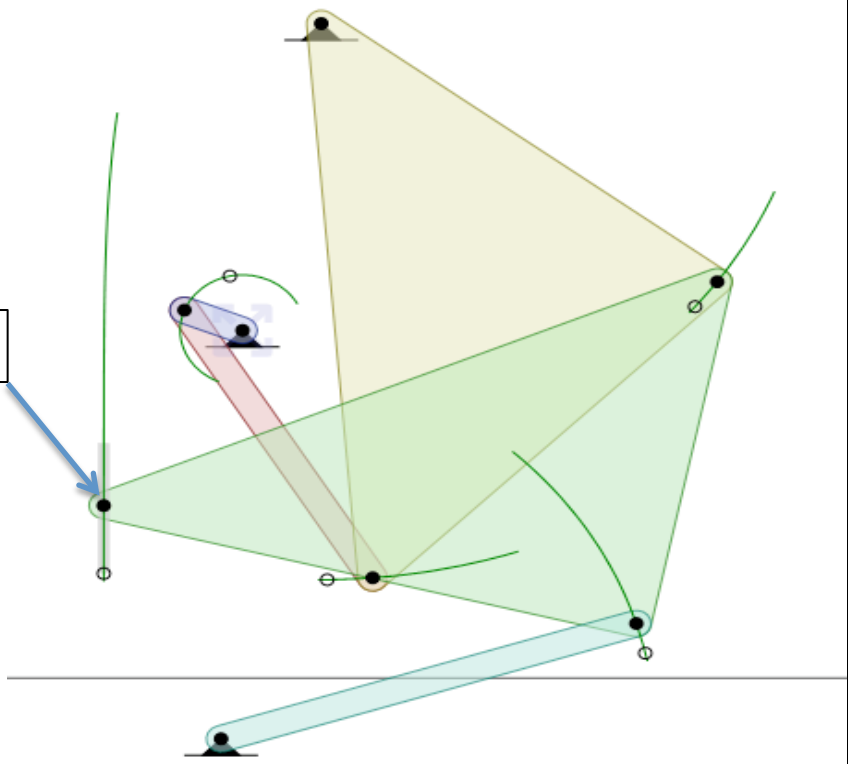
URL: http://goo.gl/jNzlrp
Objective function value: 0.1| **0.00018** (as per sum of the squares of distances)
Best results from literature: **0.0002**	**0.0178**
Results Combining search and optimization given below:

Table 7-1 continued.



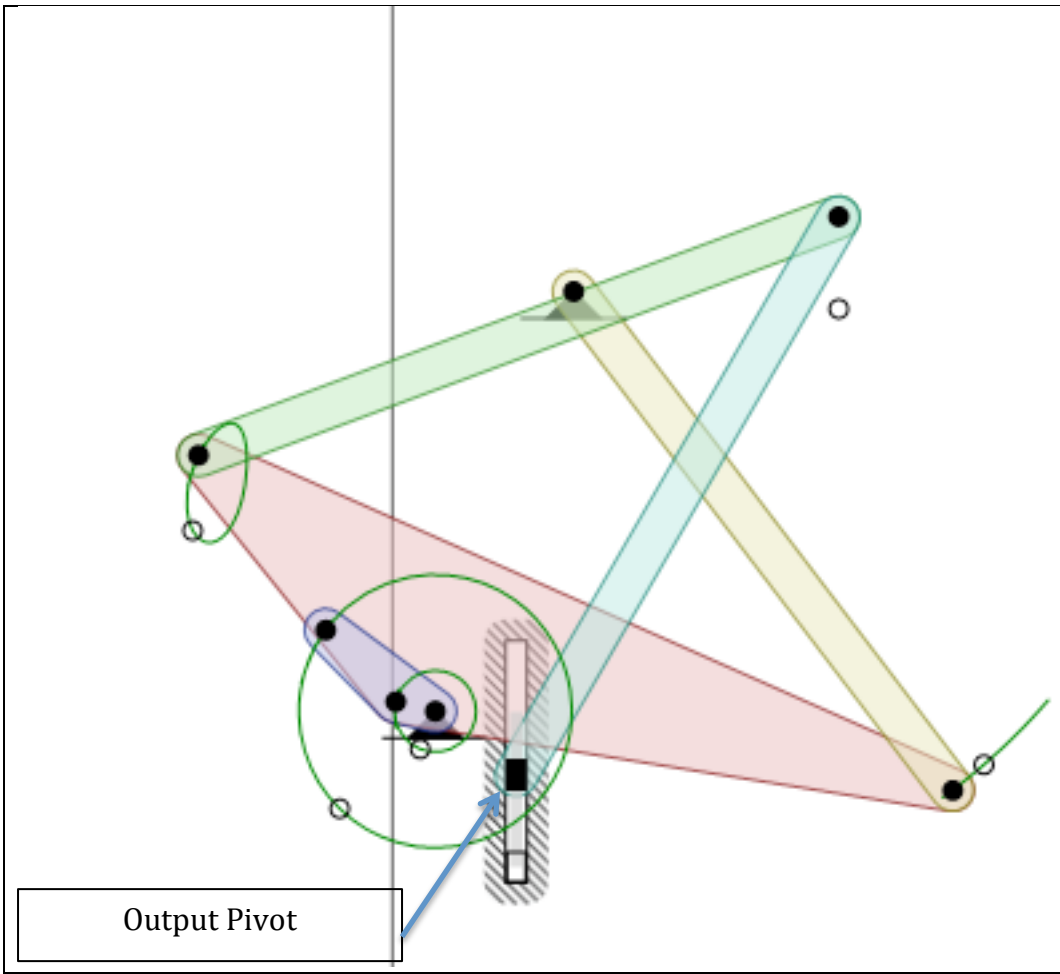
URL: http://goo.gl/xCeGzy	Objective function value: 0.051 | 0.0005 (as per sum of the squares of distances)
Configuration: 2-4-4-2 -revolute-no prismatic	

Table 7-1 continued.



URL: http://goo.gl/vtNR8x	Objective function value: 0
Configuration: 3-6-7-2 -revolute-no prismatic	

Table 7-1 continued.

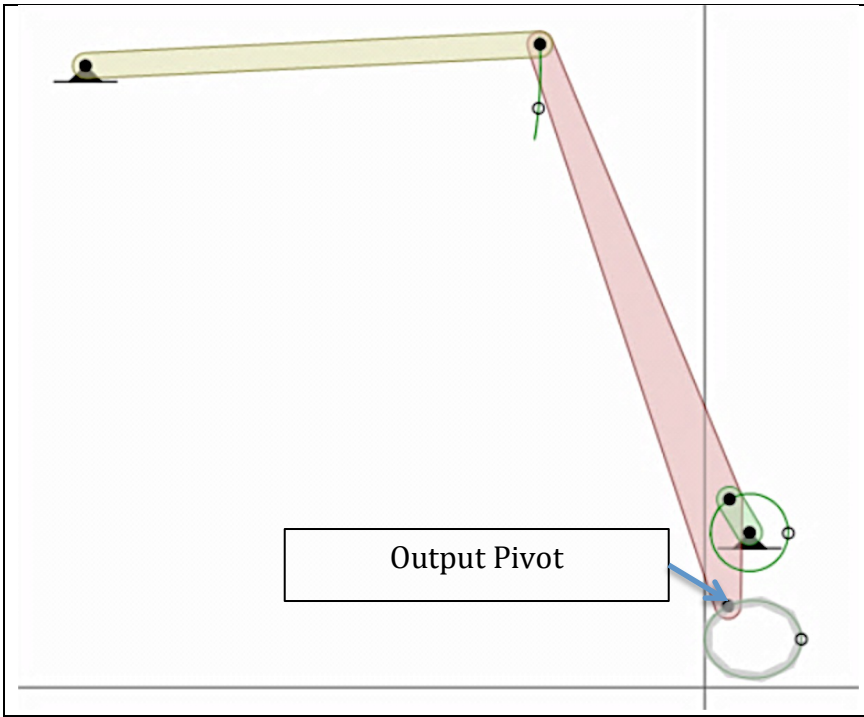


URL: http://goo.gl/nGj9Je	Objective function value: 0
Configuration: 3-6-7-2-revolute-prismatic	

Table 7-2: Results to Problem #2

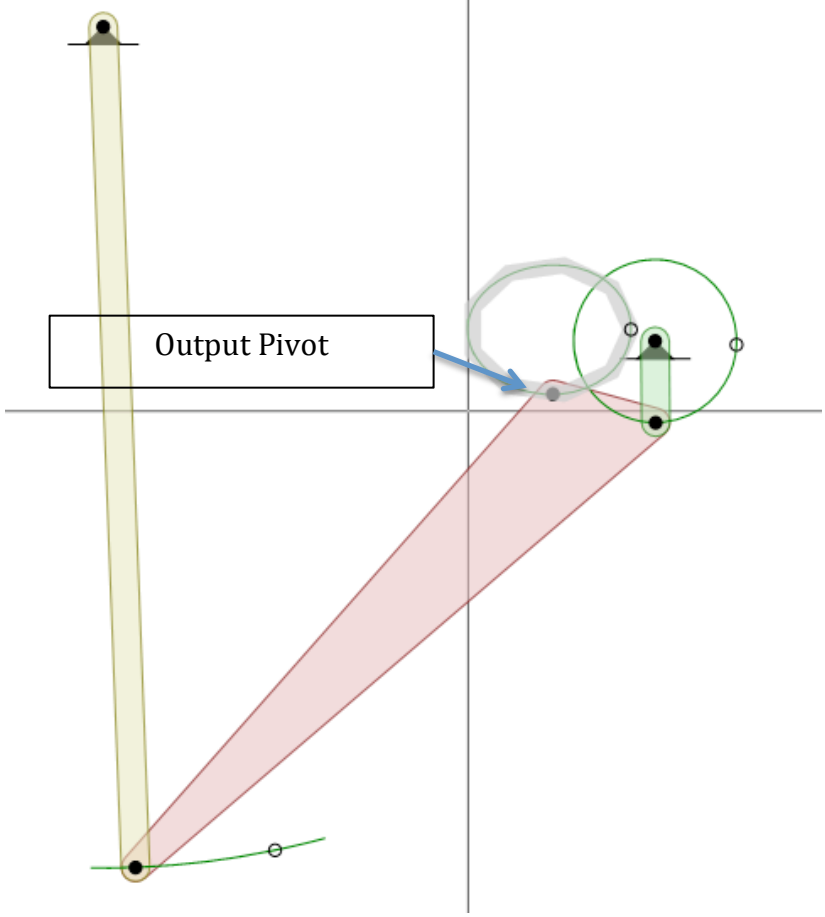
Problem 2: (20, 10), (17.66, 15.142), (11.736, 17.878), (5, 16.928), (0.60307, 12.736), (0.60307, 7.2638), (5, 3.0718), (11.736, 2.1215), (17.66, 4.8577), (20, 10)
Results: Four Bar Mechanism

Table 7-2 continued.



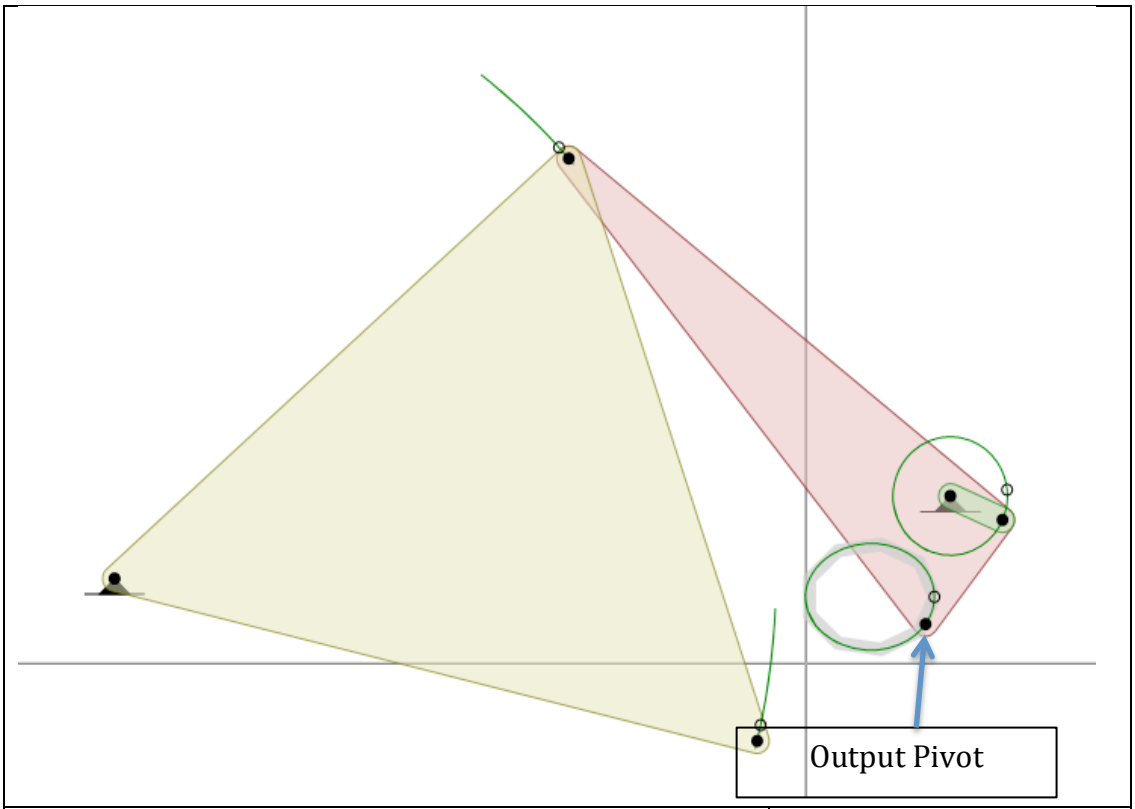
URL: http://goo.gl/yWSLcI
Objective function value: 0.11 | **0.0013** (as per sum of the squares of distances)
Best results from literature: **0.0047** | **1.9523**

Table 7-2 continued.



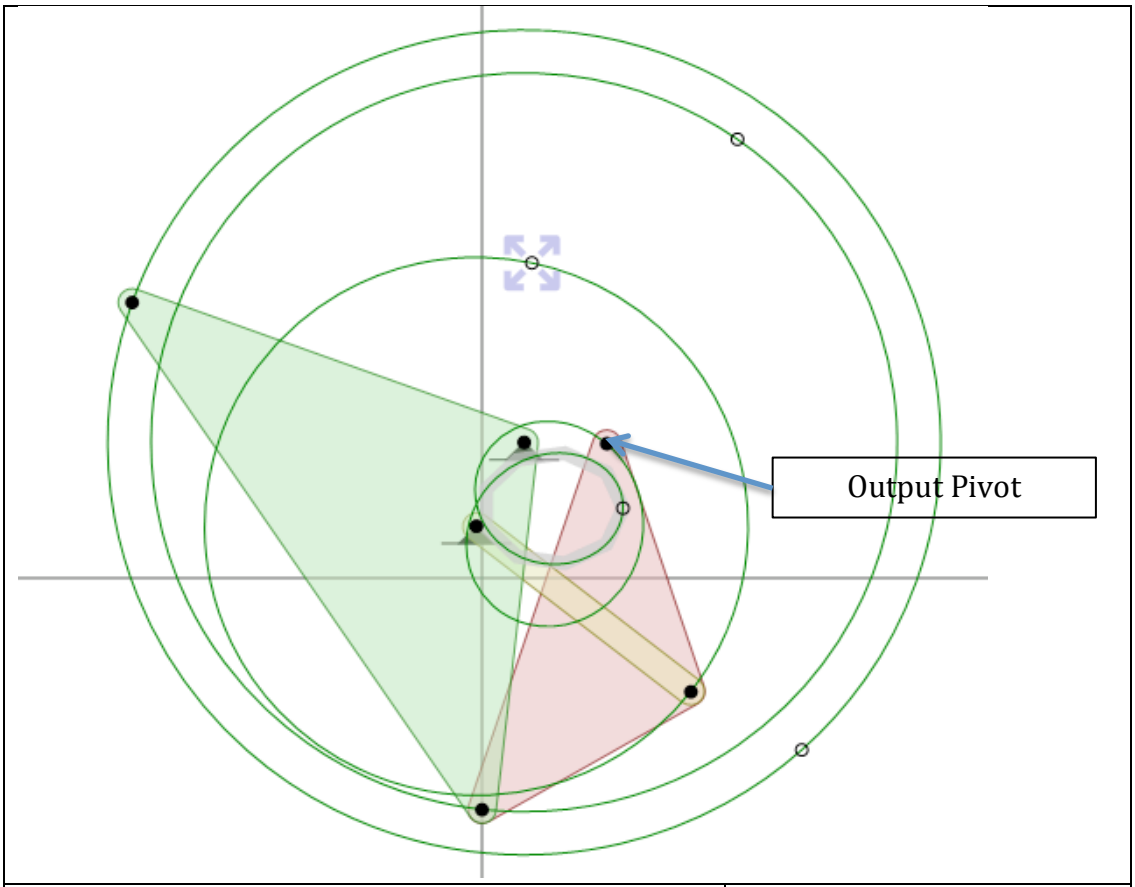
URL: http://goo.gl/vv7LQH
Objective function value: 0.46 | **0.02** (as per sum of the squares of distances)
Best results from literature: **0.0047** | **1.9523**
Results Combining search and optimization given below:

Table 7-2 continued.



URL: http://goo.gl/j3MTdp	Objective function value: 0.19 | 0.0038 (as per sum of the squares of distances)
Configuration: 2-4-4-2- revolute-no prismatic	

Table 7-2 continued.



URL: http://goo.gl/zXEPrG	Objective function value: 0.87 | 0.08 (as per sum of the squares of distances)
Configuration: 2-4-4-2-revolute-no prismatic	

Table 7-2 continued.

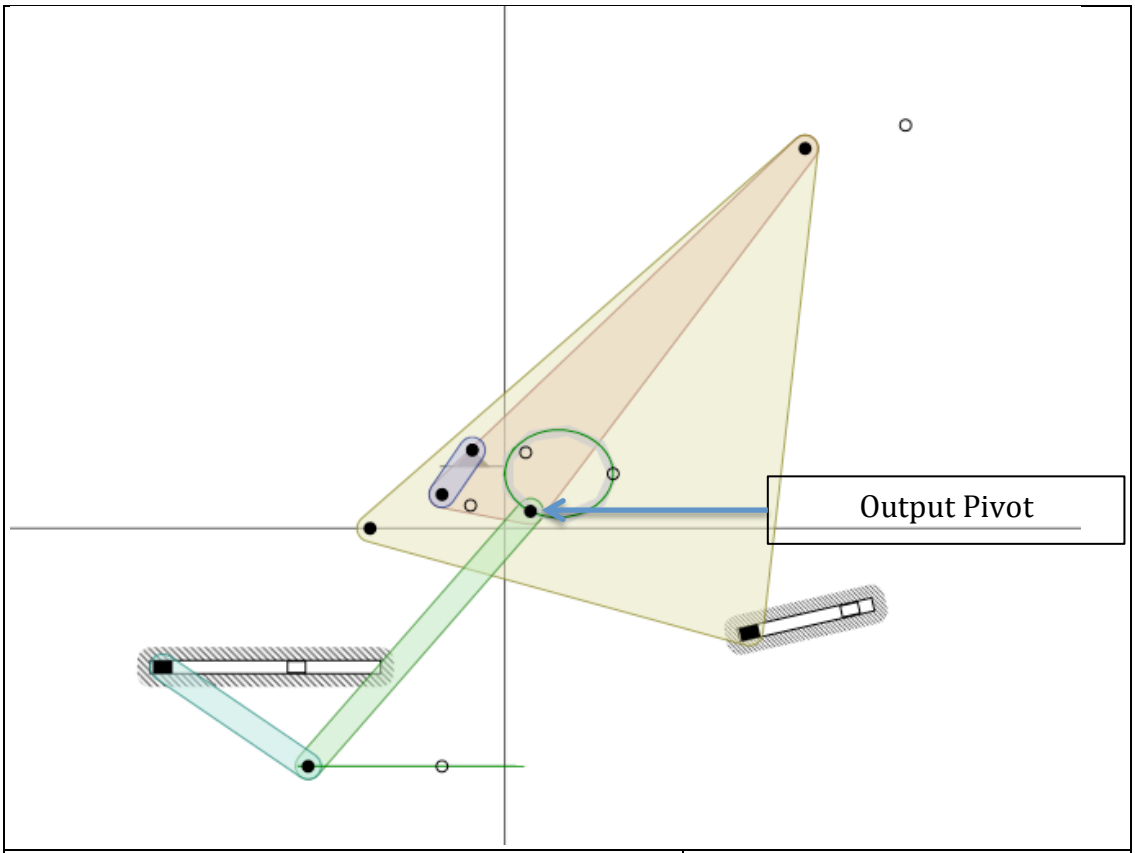
<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>URL: http://goo.gl/NkpcBW</td><td rowspan="2">Objective function value: 0.17 | 0.003 (as per sum of the squares of distances)</td></tr>
<tr><td>Configuration: 3-6-7-2-revolute-prismatic</td></tr>
</table>

Table 7-2 continued.

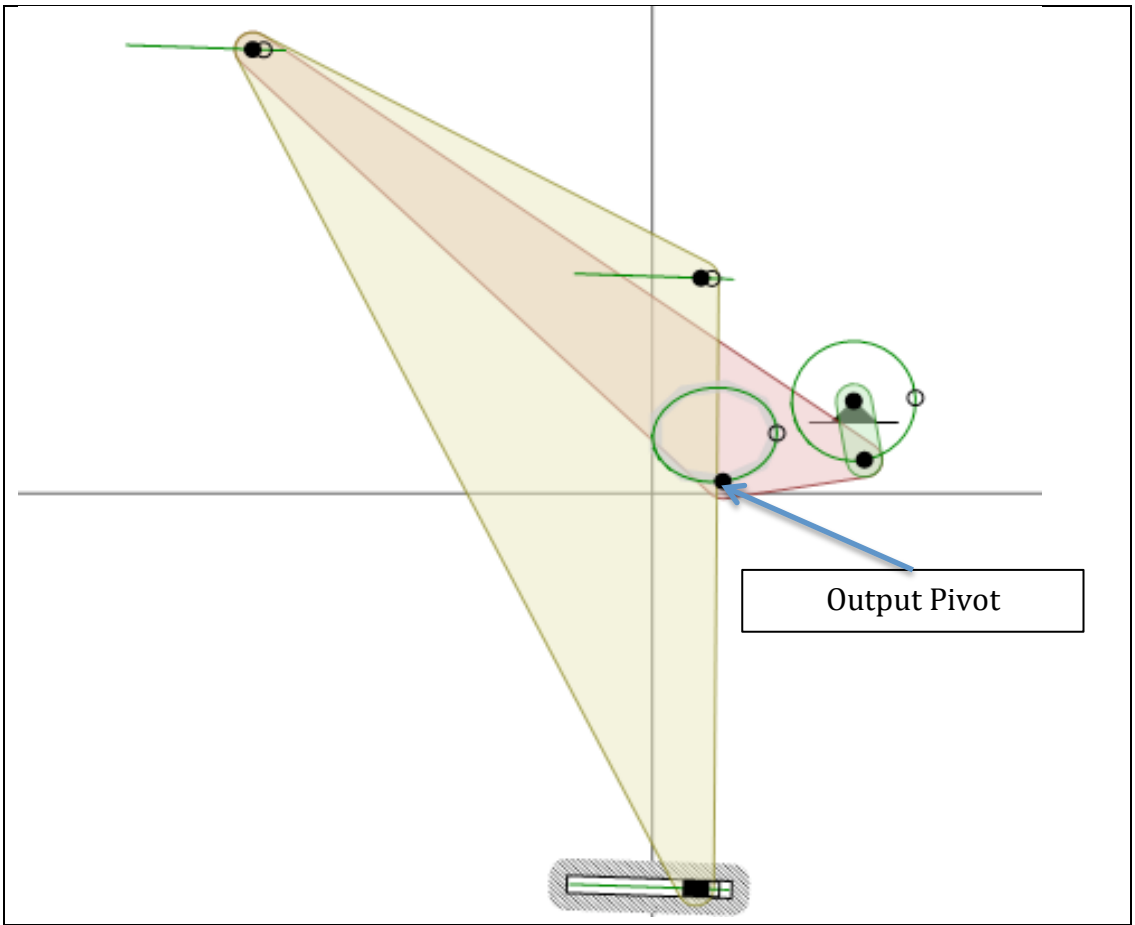
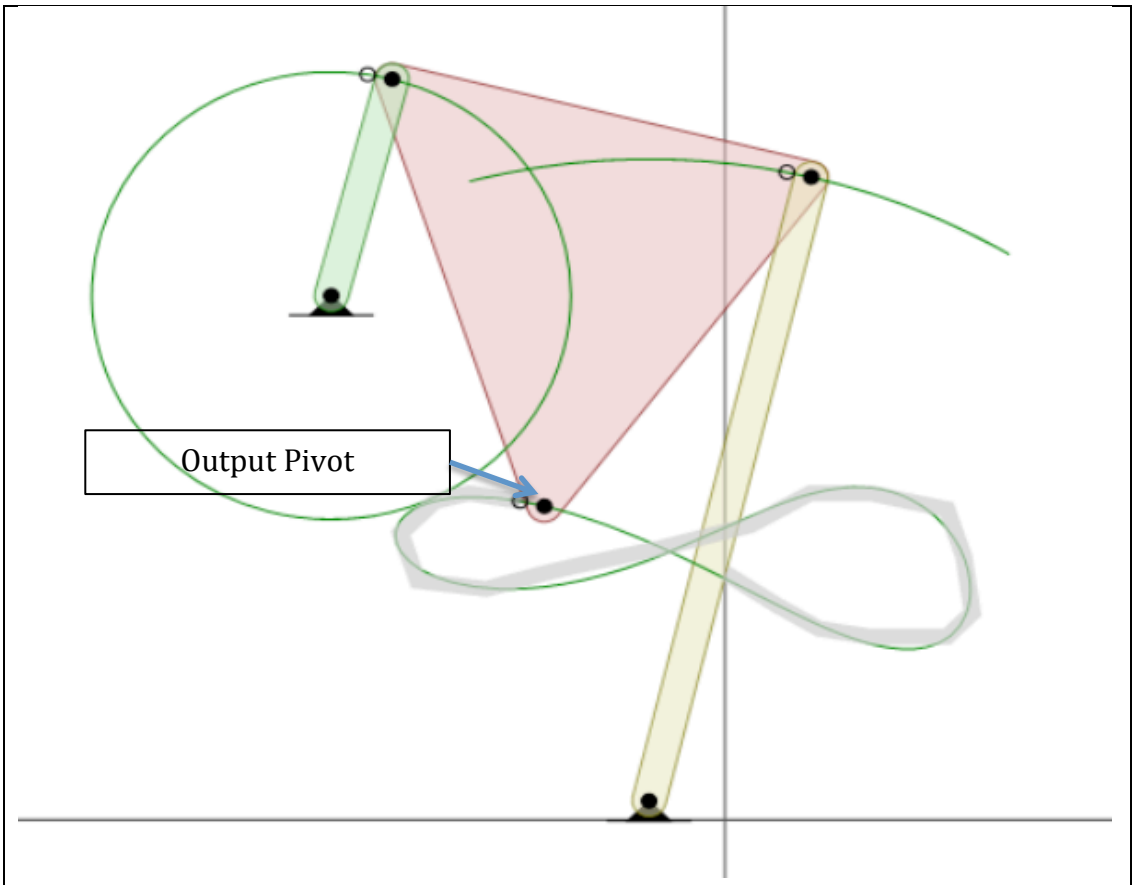
<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>URL: http://goo.gl/FL3erL</td><td rowspan="2">Objective function value: 0.07 | 0.0005 (as per sum of the squares of distances)</td></tr>
<tr><td>Configuration: 2-4-4-2-revolute-prismatic</td></tr>
</table>

Table 7-3: Results to Problem #3

Problem 3: (-24, 40), (-30, 41), (-34, 40), (-38, 36), (-36, 30), (-28, 29), (-21, 31), (-17, 32), (-8, 34), (3, 37), (10, 41), (17, 41), (26, 39), (28, 33), (29, 26), (26, 23), (17, 23), (11, 24), (6, 27), (0, 31)
Result: Four-bar Mechanism

Table 7-3 continued.



URL: http://goo.gl/h4d6NT
Objective function value: 12.23	**0.63** (average distance error)
Best result from literature: **0.98**

Table 7-3 continued.

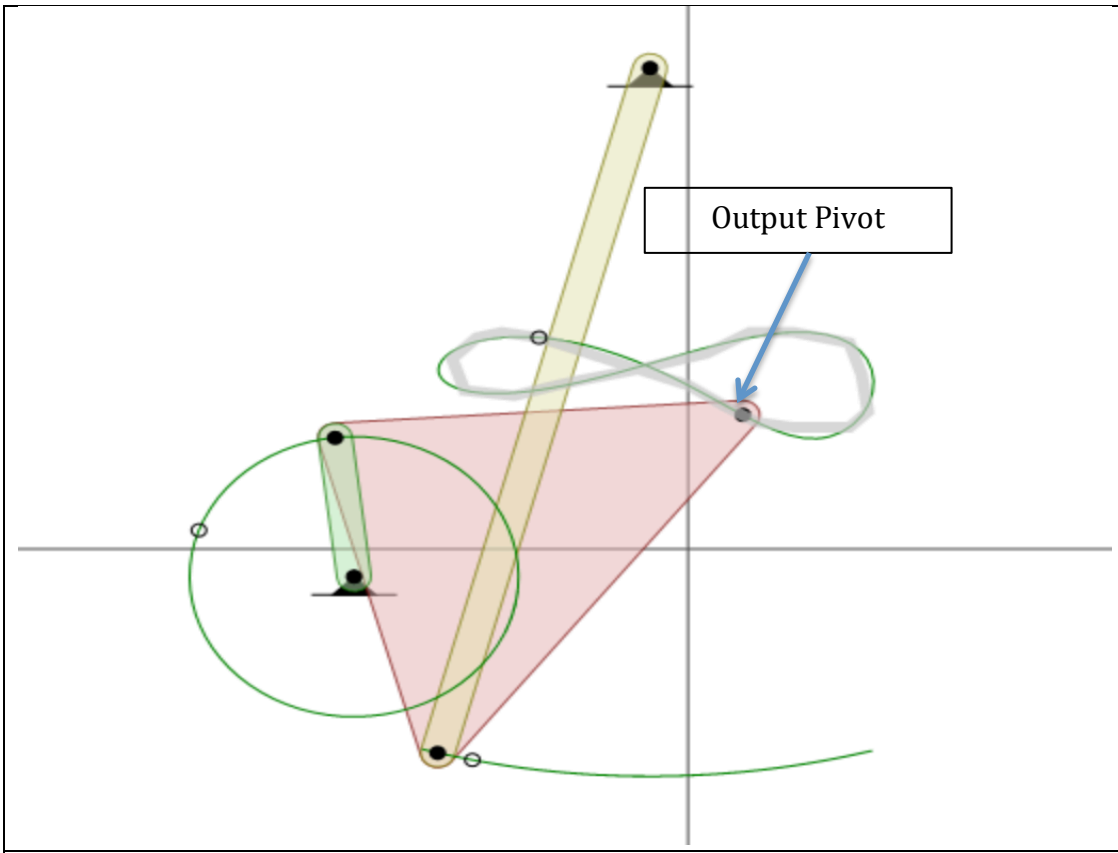
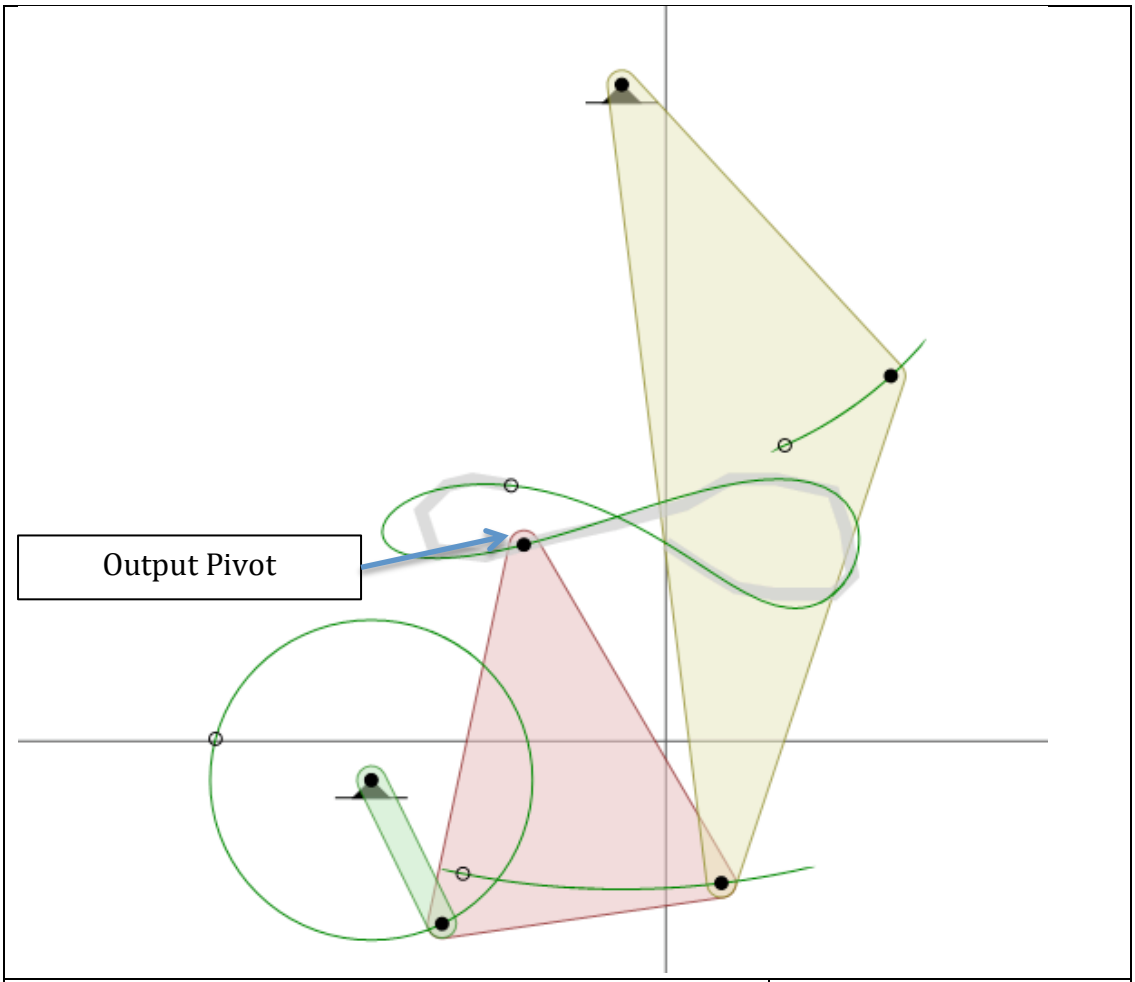

URL: http://goo.gl/PfHySZ
Objective function value: 10.40 | **0.52** (average distance error)
Best result from literature: **0.98**
Results Combining search and optimization given below:

Table 7-3 continued.



URL: http://goo.gl/OHkz3f	Objective function value: 12.96 | 0.648 (average distance error)
Configuration: 2-4-4-2-revolute-no prismatic	

Table 7-3 continued.

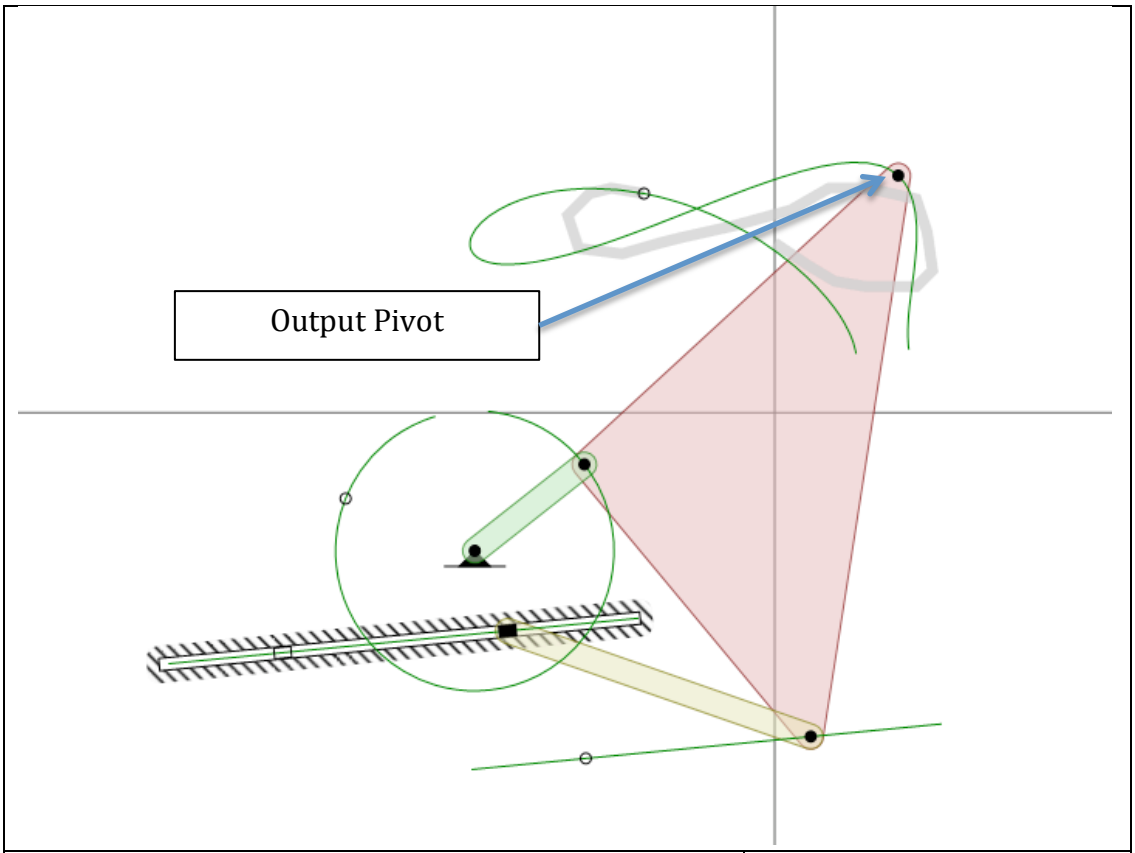
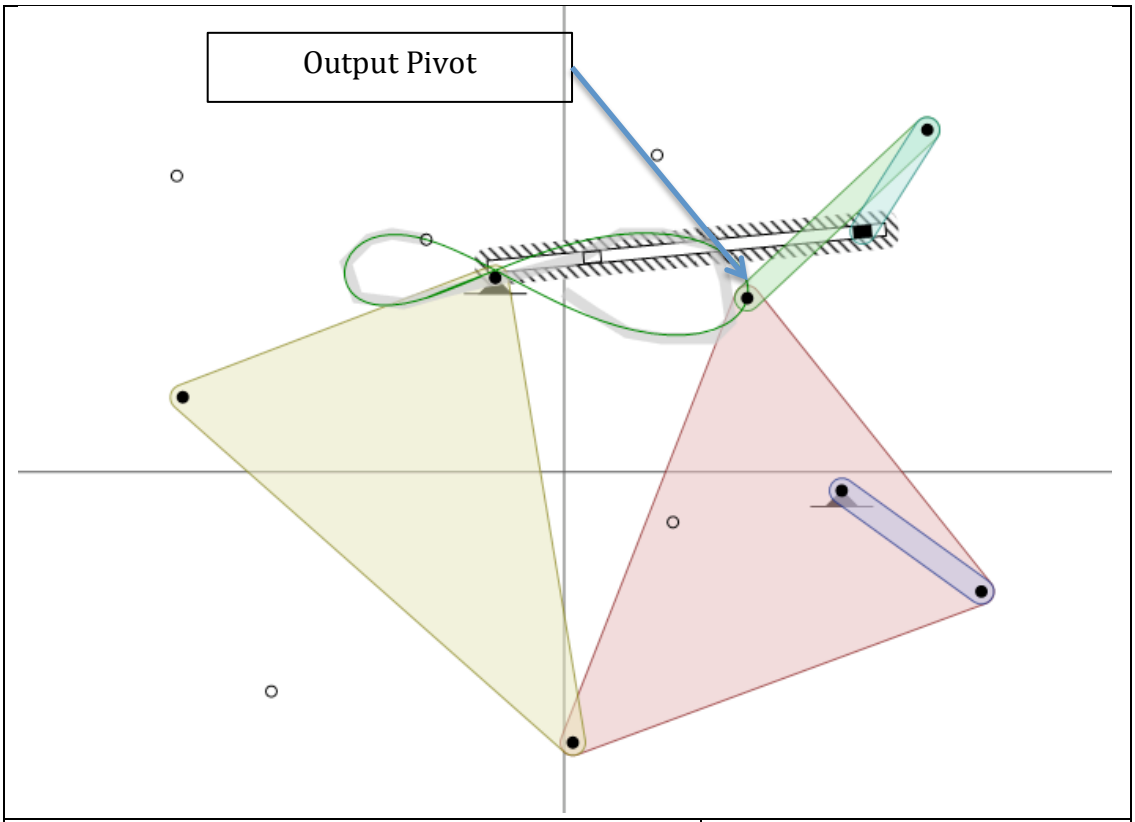
<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>URL: http://goo.gl/zqoCy6</td><td rowspan="2">Objective function value: 16.56 | 0.83 (average distance error)</td></tr>
<tr><td>Configuration: 2-4-4-1-revolute-prismatic</td></tr>
</table>

Table 7-3 continued.



URL: http://goo.gl/WxOqhN	Objective function value: 14.00	0.7 (average distance error)
Configuration: 3-6-7-2-revolute-prismatic	

Table 7-4: Results to Problem #4

Problem 4: (-27,1), (-21.857, -3.214), (-16.7, -7.428), (-6.428, -15.857), (-1.285, -20.071), (3.857, -24.285), (9, -28.5), (15, -29.9), (20, -30), (27.2, -25), (29.2, -20), (28, -10), (22.7,2), (15,10.6), (5,16.5), (-10,19.6), (-22,17), (-28,11), (-29,5)
Result: Four-bar Mechanism

Table 7-4 continued.

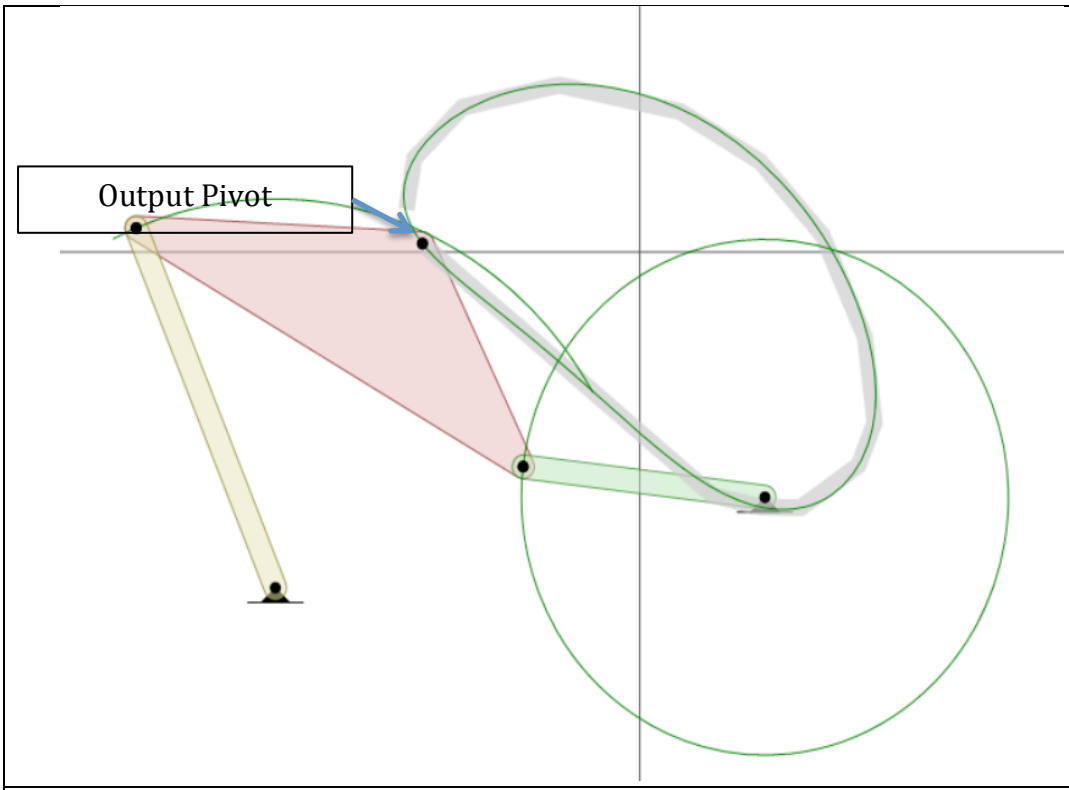

URL: http://goo.gl/pObY62
Objective function value: 3.0	**0.15** (average distance error)
Best result from literature: **0.4154**

Table 7-4 continued.

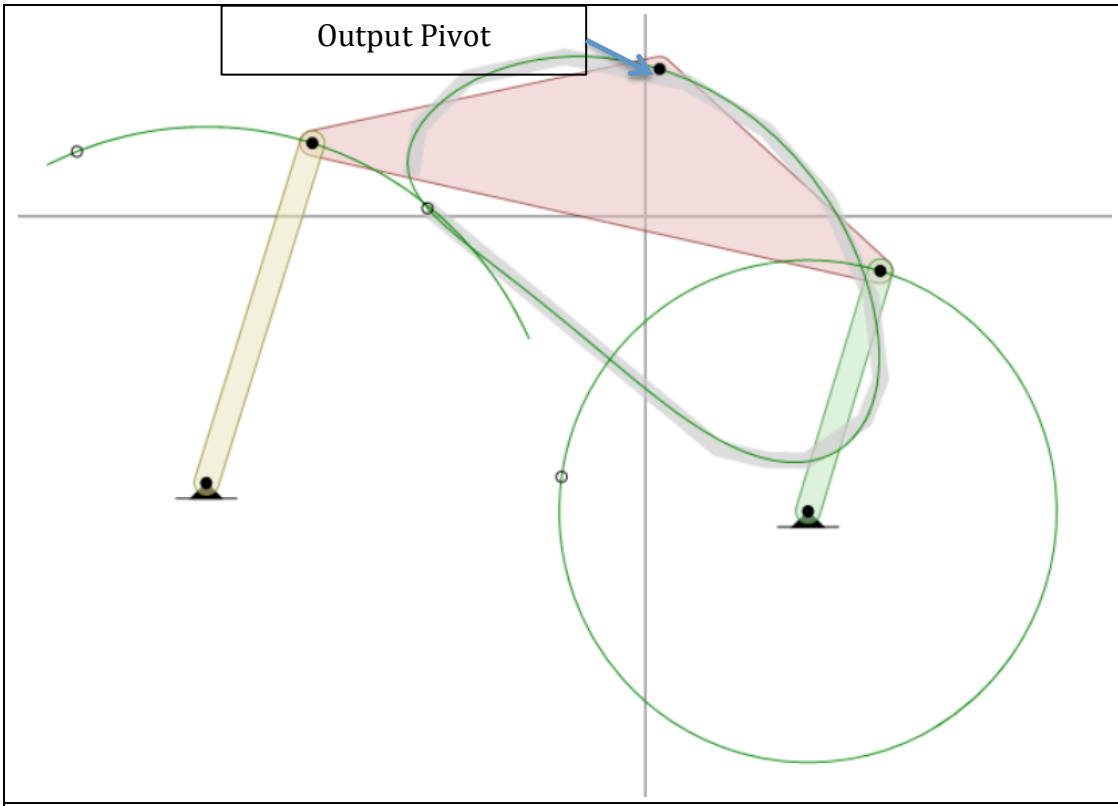
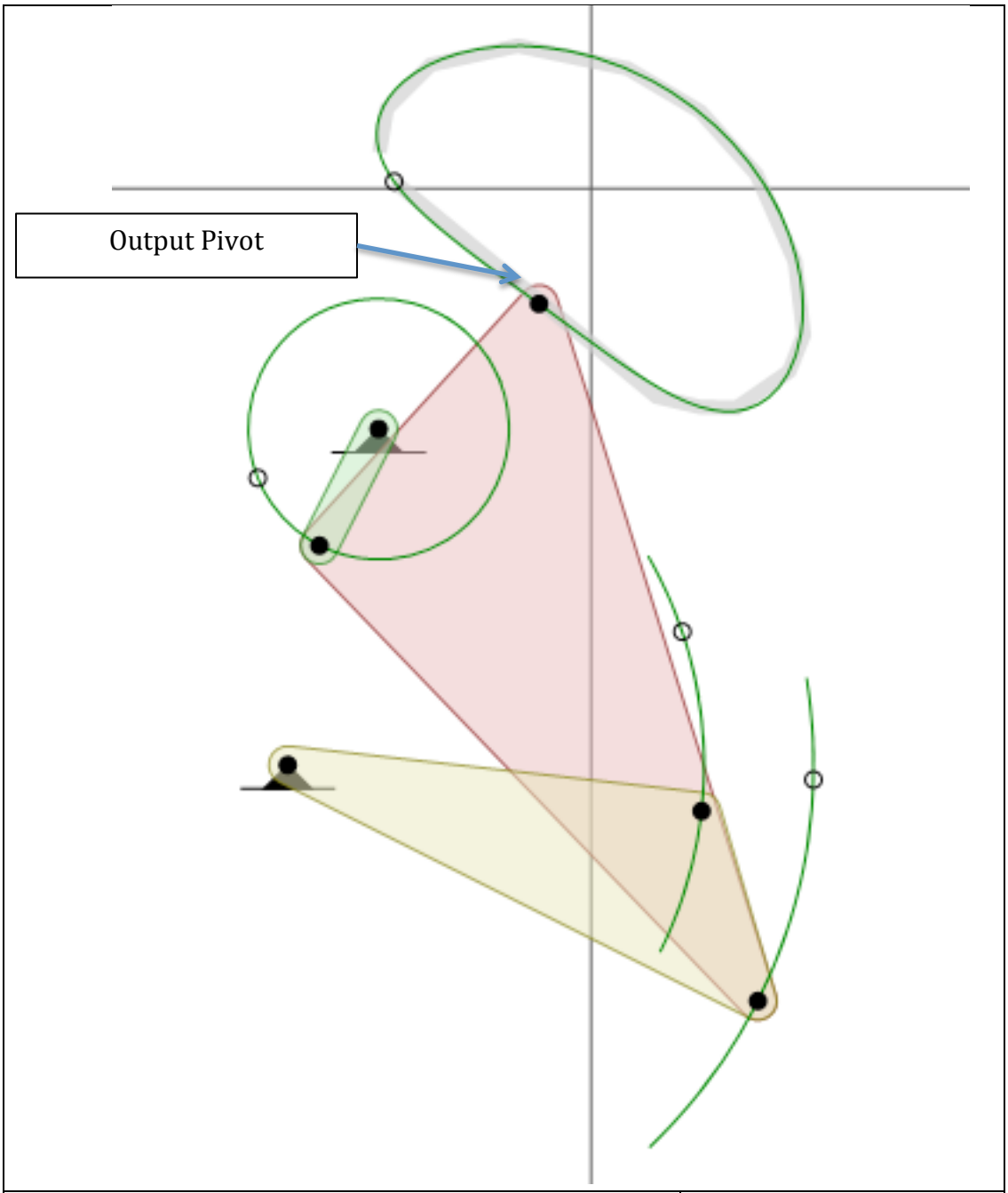
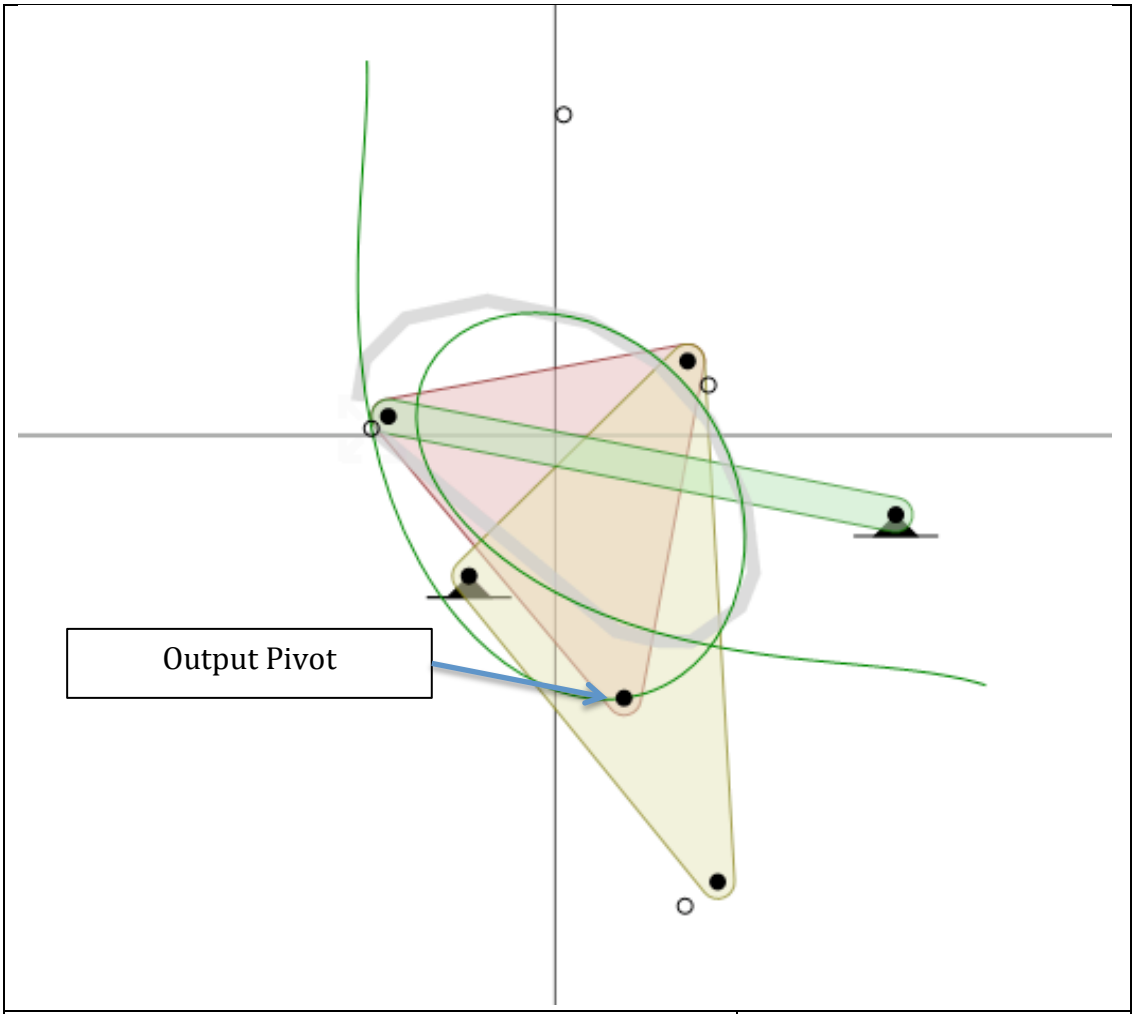

URL: http://goo.gl/dwmFEa
Objective function value: 3.78 | **0.19** (average distance error)
Best result from literature: **0.4154**
Results Combining search and optimization given below:

Table 7-4 continued.



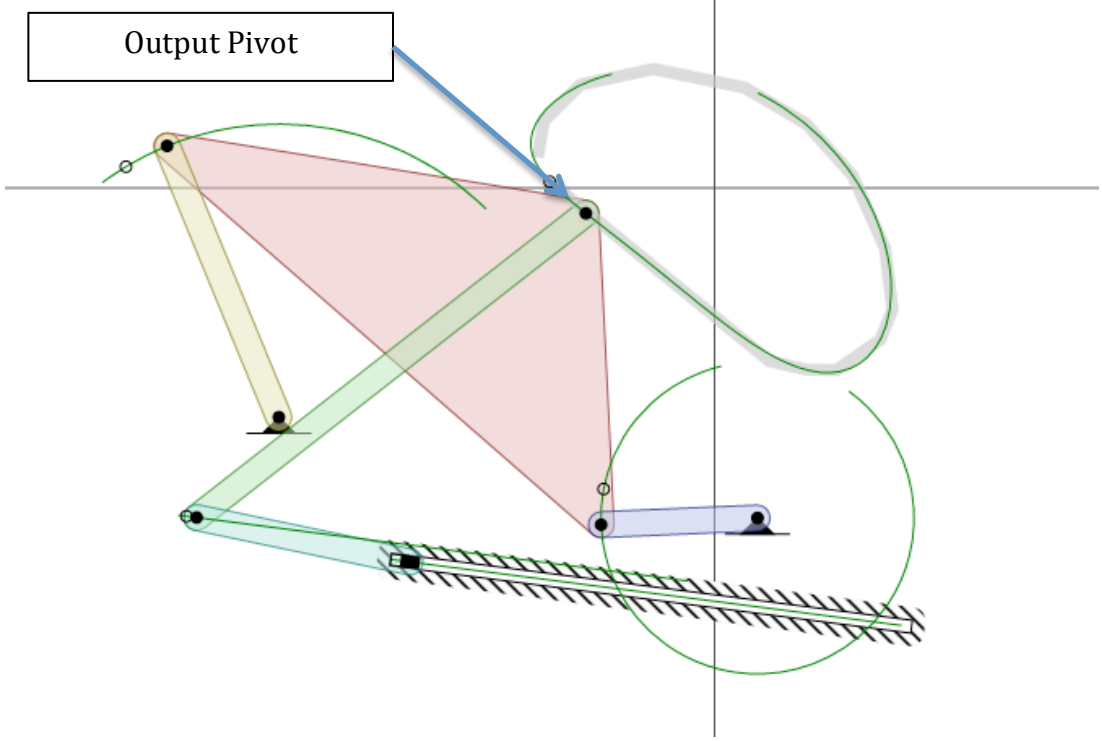
URL: http://goo.gl/axXi9r	Objective function value: 5.99 | 0.30 (average distance error)
Configuration: 2-4-4-2-revolute-no prismatic	

Table 7-4 continued.



URL: http://goo.gl/LHc2J5	Objective function value: 21.33 | 1.07(average distance error)
Configuration: 2-4-4-2-revolute-no prismatic	

Table 7-4 continued.

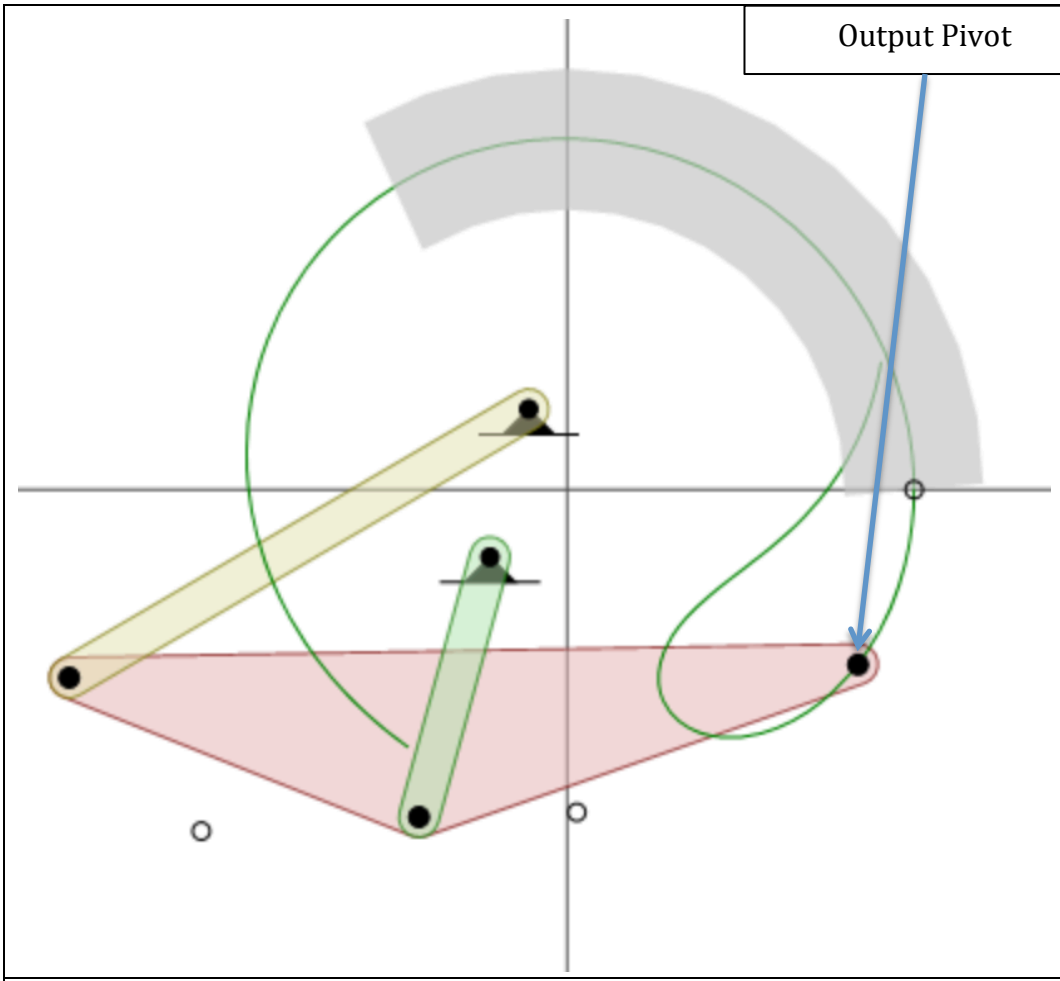


URL: http://goo.gl/Tzx6du	Objective function value: 4.23 | 0.21 (average distance error)
Configuration: 3-6-7-1-revolute-prismatic	

Table 7-5: Results to Problem #5

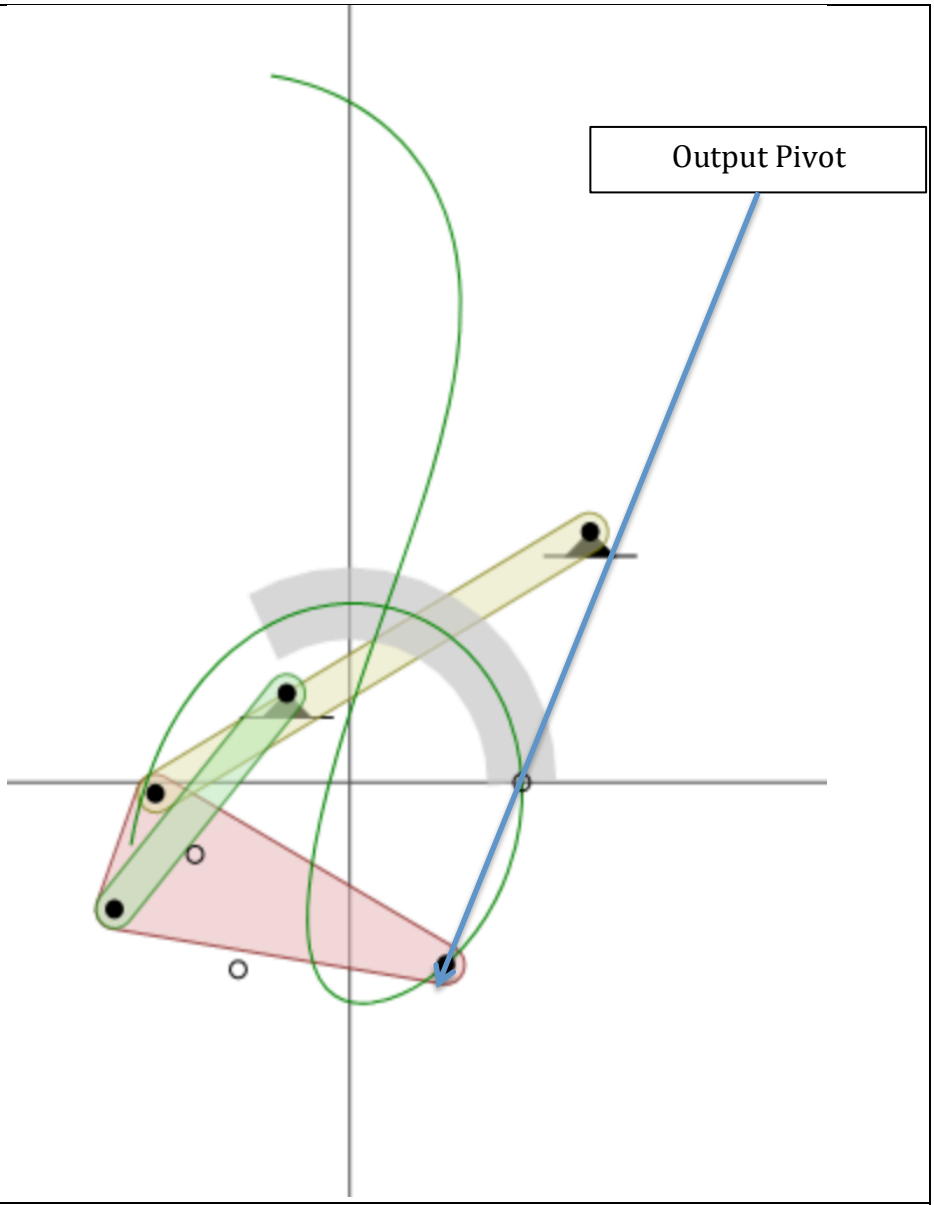
Problem 5: (5, 0), (4.9240, 0.8682), (4.6985, 1.7101), (4.3301, 2.500), (3.8302, 3.2139), (3.2129, 3.8302), (2.5, 4.3301), (1.7101, 4.6985), (0.8682, 4.9240), (0, 5), (-0.8682, 4.9240), (-1.7101, 4.6985), (-2.5, 4.3301)
Result: Four-bar Mechanism

Table 7-5 continued.



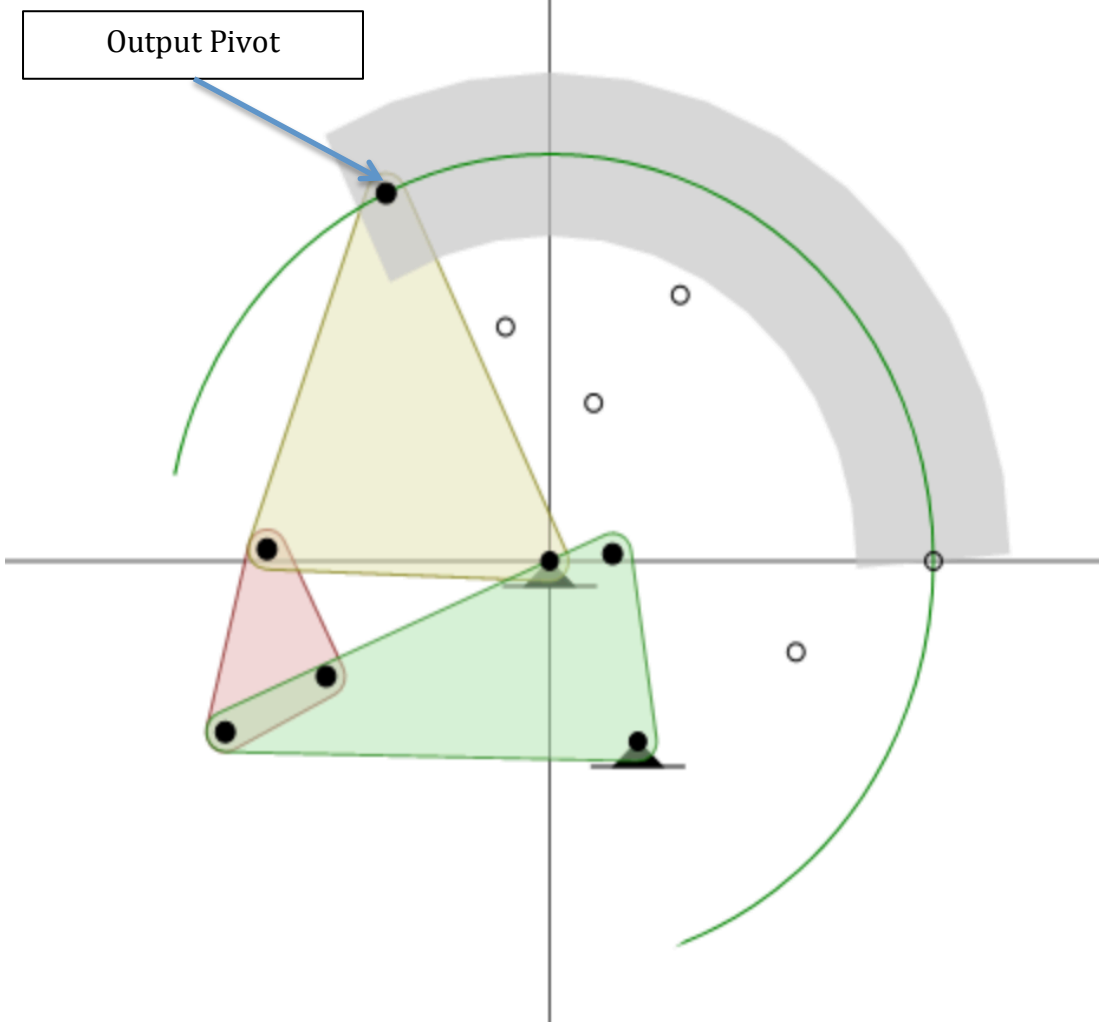
URL: http://goo.gl/LKK4Wj
Objective function value: 0.09 | **0.0007** (as per sum of the squares of distances)
Best result from literature: **0.0154**

Table 7-5 continued.



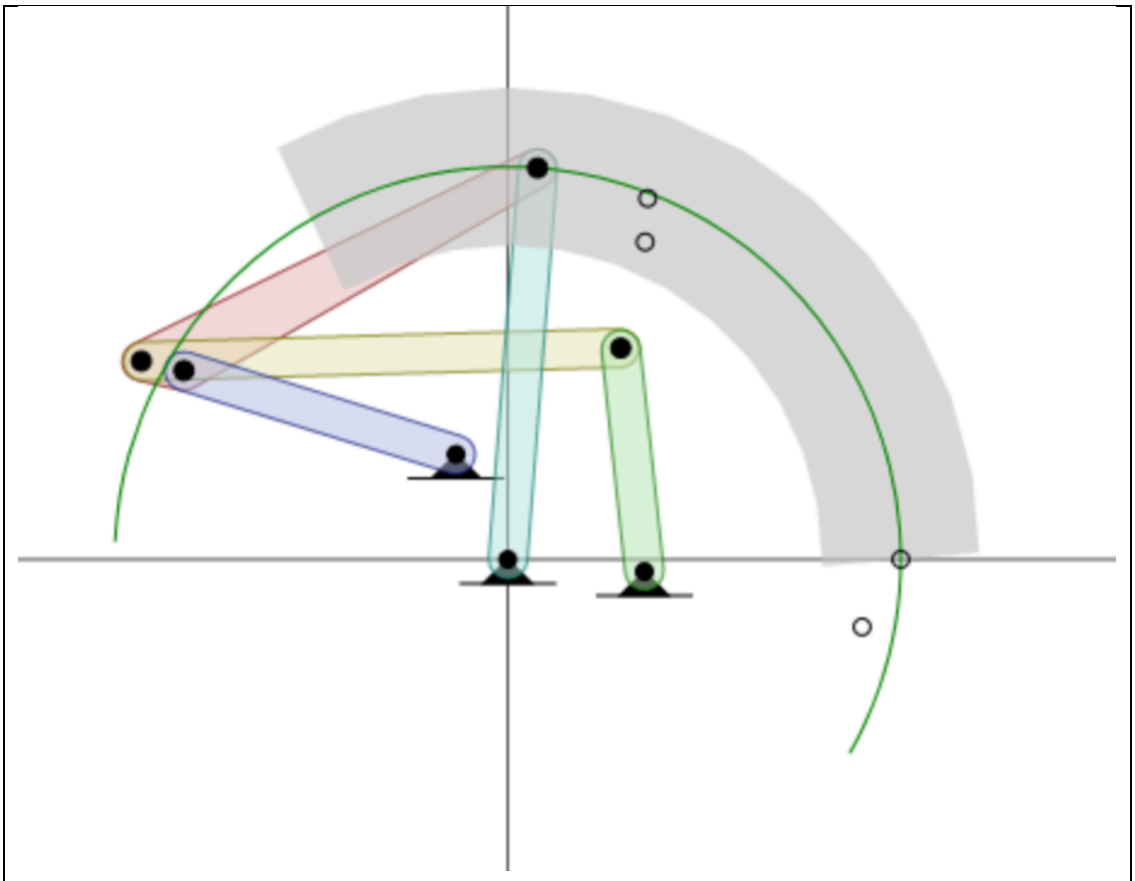
URL: http://goo.gl/Wsa5XN
Objective function value: 0.27 I **0.006** (as per sum of the squares of distances)
Best result from literature: **0.0154**
Results Combining search and optimization given below:

Table 7-5 continued.



URL: http://goo.gl/PJOsdD	Objective function value: 0.02 | 3E-5 (as per sum of the squares of distances)
Configuration: 3-6-7-1-revolute-no prismatic	

Table 7-5 continued.

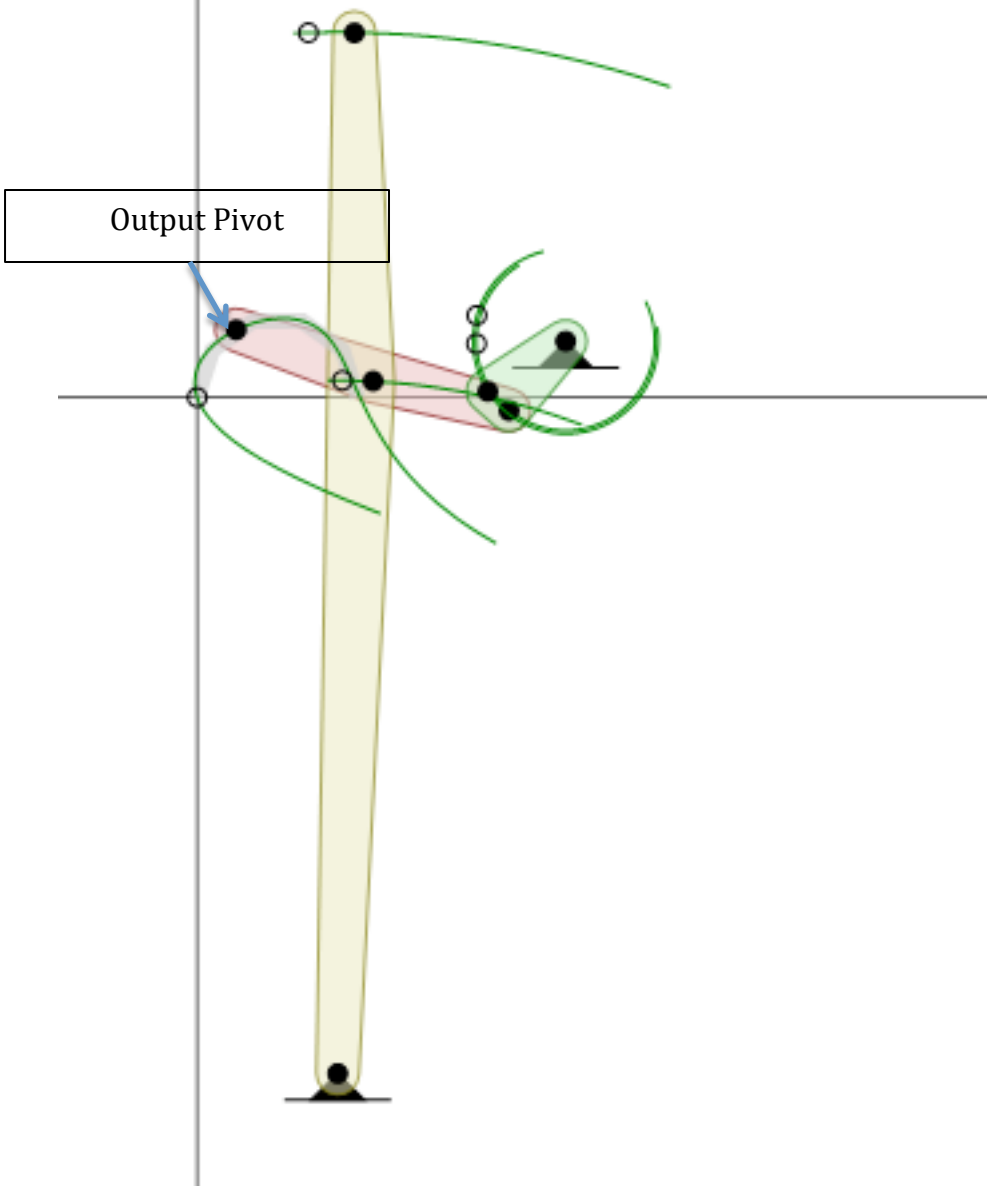


URL: http://goo.gl/TIxRXw	Objective function value: 0.142	0.002 (as per sum of the squares of distances)
Configuration: 3-6-7-1-revolute-no prismatic	

Table 7-6: Results to Problem #6

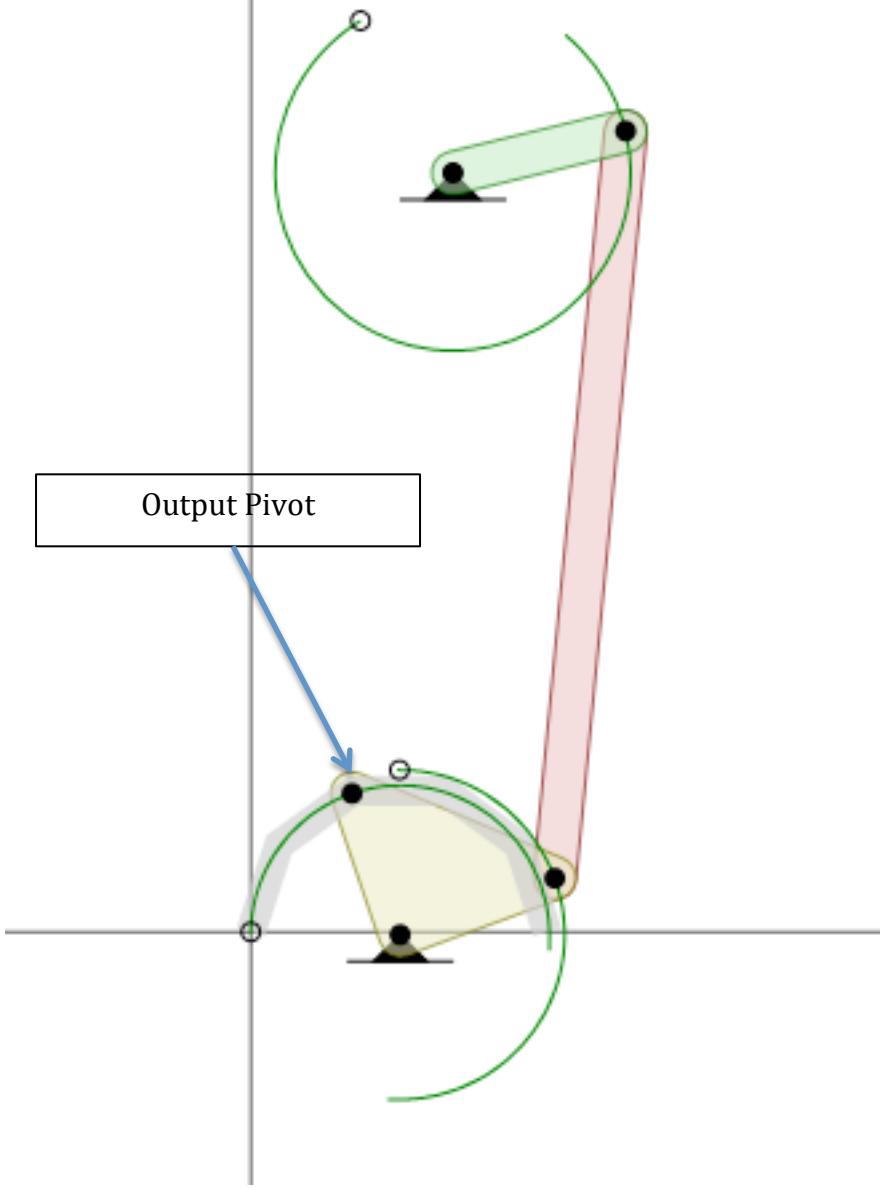
Problem 6: (0, 0), (1.9098, 5.8779), (6.9098, 9.5106), (13.09, 9.5106), (18.09, 5.877), (20, 0) Time: (π/ 6, π / 3, π / 2, 2 * π / 3, 5 * π / 6, π)
Result: Four-bar Mechanism

Table 7-6 continued.



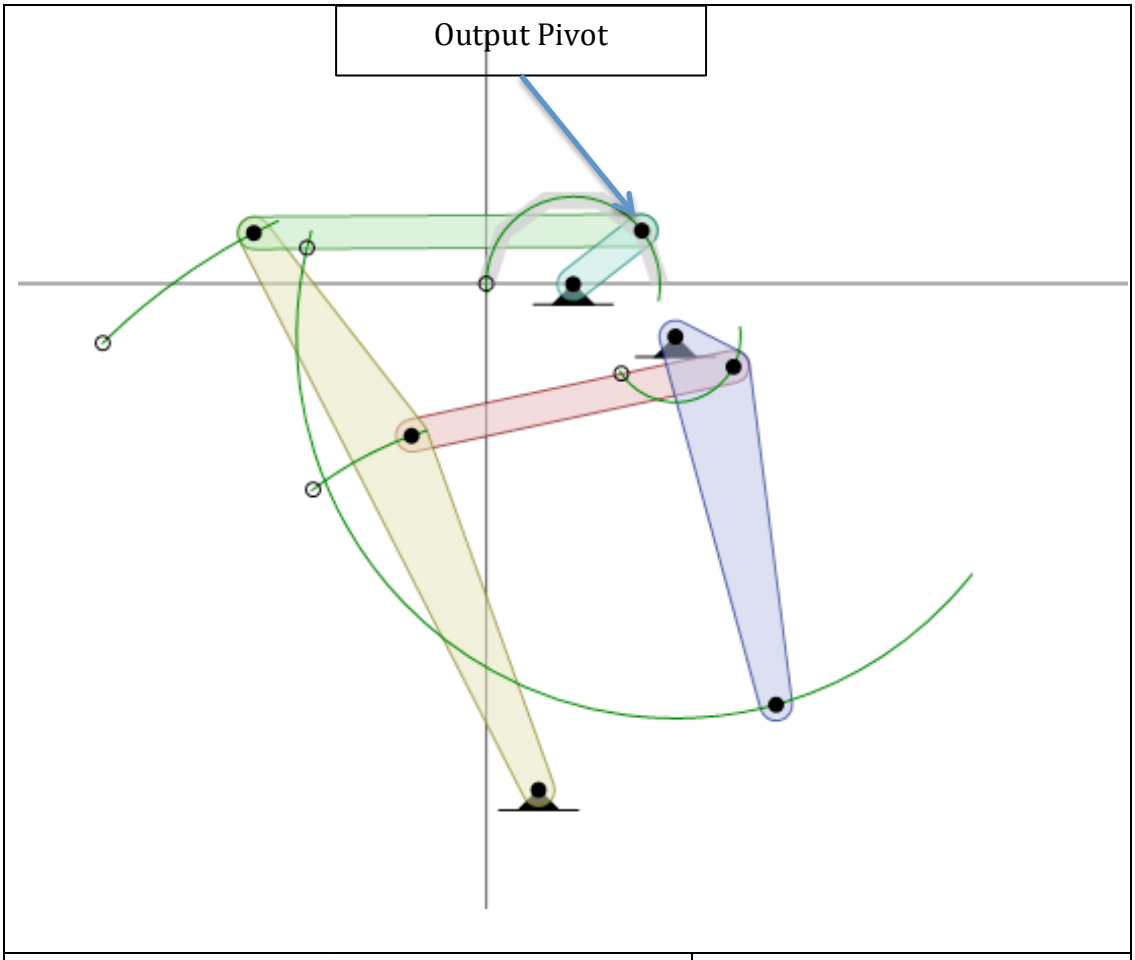
URL: http://goo.gl/FjPnh9
Objective function value: 1.184 | **0.25** (as per sum of the squares of distances)
Best results from literature: **1.2162 | 5.5207**
Results Combining search and optimization given below:

Table 7-6 continued.



URL: http://goo.gl/95cUuf	Objective function value: 1.23 | 0.26 (as per sum of the squares of distances)
Configuration: 2-4-4-1-revolute-no prismatic	

Table 7-6 continued.



URL: http://goo.gl/RFd5ie	Objective function value: 0.09 | 0.001 (as per sum of the squares of distances)
Configuration: 3-6-7-2-revolute-prismatic	

Table 7-6 continued.

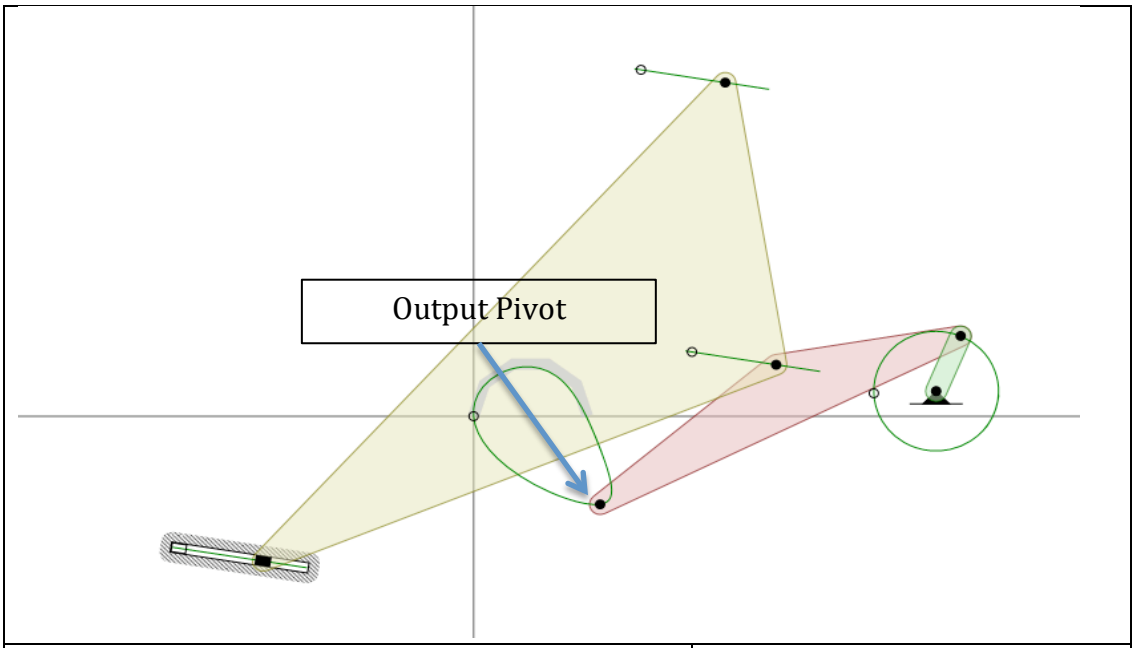
<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>URL: http://goo.gl/qvPYUJ</td><td rowspan="2">Objective function value: 0.87 | 0.13 (as per sum of the squares of distances)</td></tr>
<tr><td>Configuration: 2-4-4-2-revolute-prismatic</td></tr>
</table>

Table 7-7: Results to Problem #7

Problem 7: (0.5, 1.1), (0.4, 1.1), (0.3, 1.1), (0.2, 1.0), (0.1, 0.9), (0.005, 0.75), (0.02, 0.6), (0.0, 0.5), (0.0, 0.4), (0.03, 0.3), (0.1, 0.25), (0.15, 0.2), (0.2, 0.3), (0.3, 0.4), (0.4, 0.5), (0.5, 0.7), (0.6, 0.9), (0.6, 1.0) Time: (0,21 * π / 180, 42 * π / 180, 63 * π / 180, 84 * π / 180, 105 * π / 180, 126 * π / 180, 147 * π / 180, 168 * π / 180, 189* π / 180, 210* π / 180, 231* π / 180, 252* π / 180, 273* π / 180, 294* π / 180, 315* π / 180, 336* π / 180, 357 * π / 180)
Result: Four-bar mechanism

Table 7-7 continued.

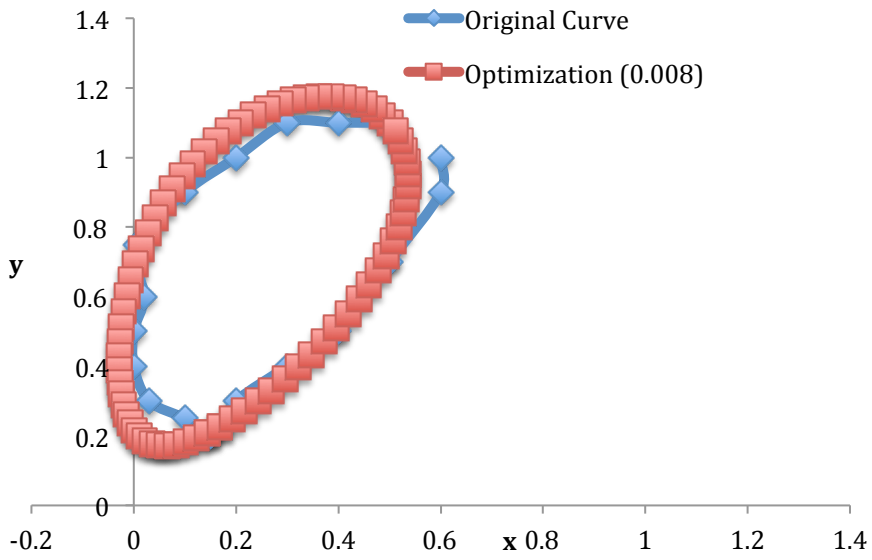
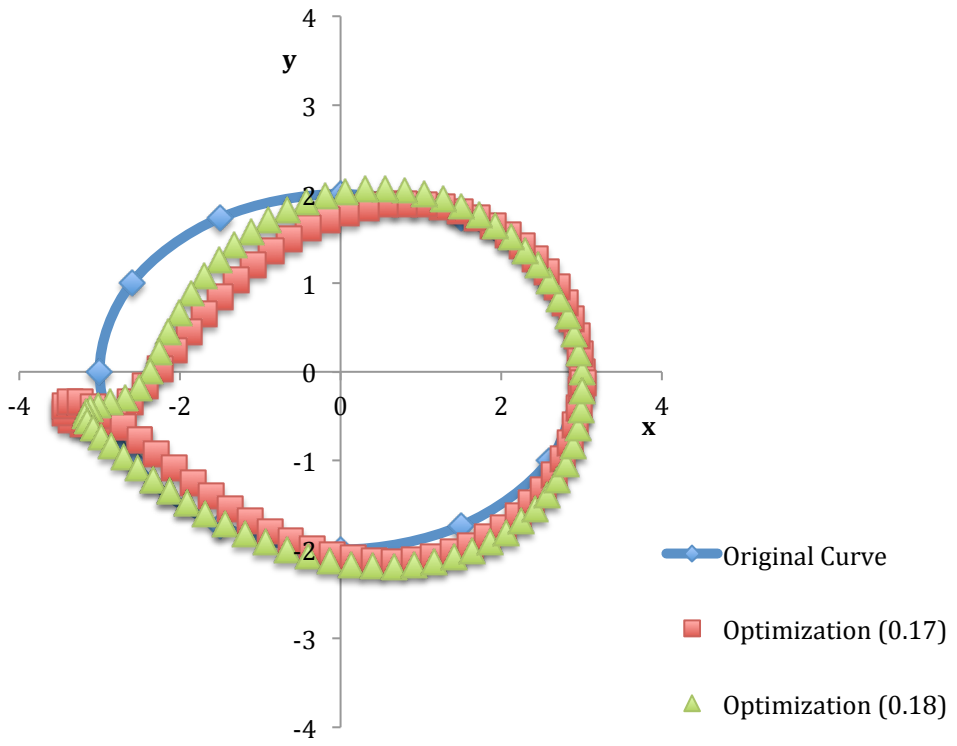

URL: http://goo.gl/FliZXK
Objective function value: 0.36l **0.008** (as per sum of the squares of distances)
Best results from literature: **0.0196 l 0.043**
Results Combining search and optimization given below:
Configuration: 2-4-4-1-revolute-no prismatic l URL: http://goo.gl/SLvuQk Objective function value: 0.64 l 0.024 (as per sum of the squares of distances) Configuration: 3-6-7-1-revolute-no prismatic l URL: http://goo.gl/lgQdKK Objective function value: 0.69 l 0.028 (as per sum of the squares of distances)

Table 7-8: Results to Problem #8

Problem 8: x(t)=3 cos(t), y(t)=2 sin(t), where t is time
Result: Four-bar Mechanism (the benchmark is solved using a four-bar slider crank mechanism while the result shown below is using a four-bar mechanism with revolute joints)

Table 7-8 continued.



URL: http://goo.gl/niDoV9 | http://goo.gl/e8MkY6
Objective function value: 1.38 | **0.17** (as per sum of the squares of distances)
Best results from literature: **0.1298**
Results Combining search and optimization given below:
Configuration: 2-4-4-2-revolute-no prismatic | URL: http://goo.gl/cAE5ZW Objective function value: 1.41| 0.17 (as per sum of the squares of distances) Configuration: 2-4-4-1-revolute-no prismatic | URL: http://goo.gl/nO8Lmx Objective function value: 1.61| 0.23 (as per sum of the squares of distances) Configuration: 3-6-7-1-revolute-no prismatic | URL: http://goo.gl/xuk1tq Objective function value: 2.64 | 0.61 (as per sum of the squares of distances)

Table 7-9: Results to Problem #9

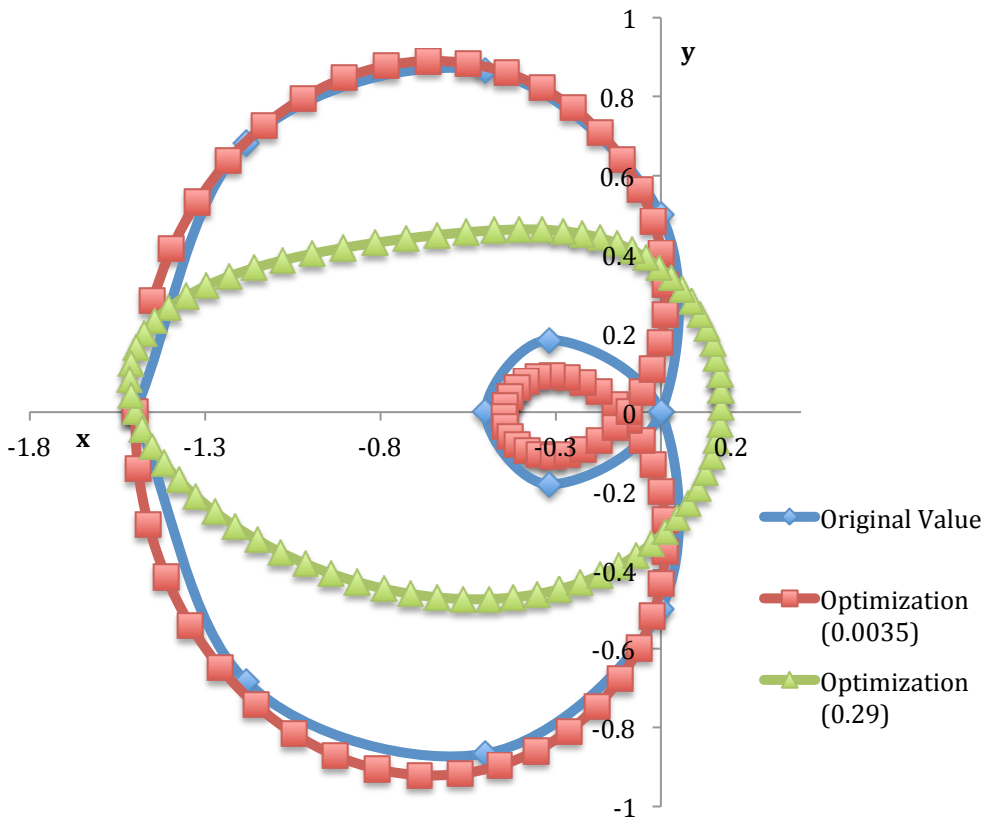
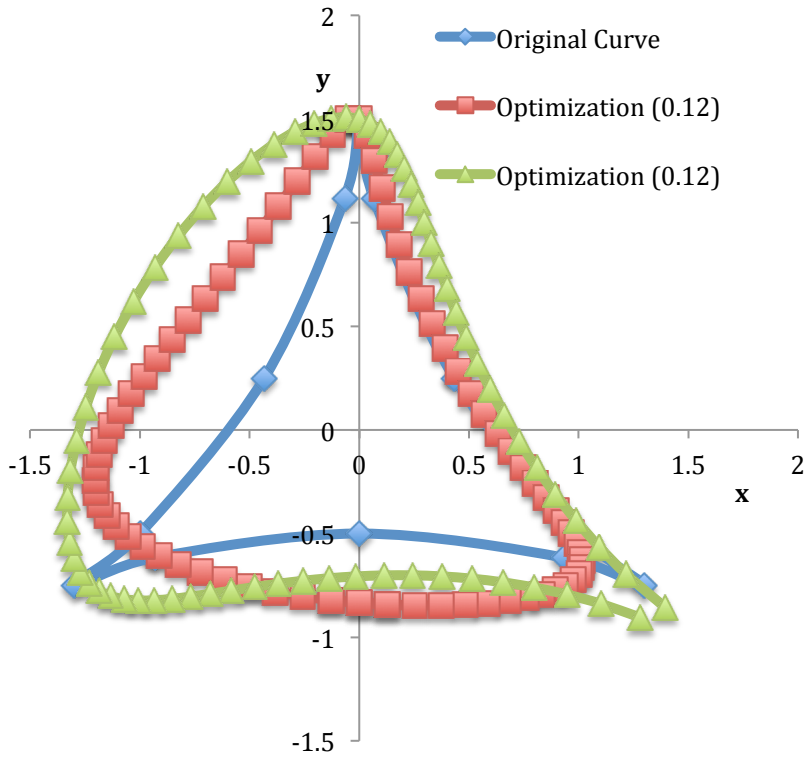
Problem 9: x(t)=-cos(t)*(0.5+cos(t)), y(t)=- sin(t)(0.5_cos(t)), t is time
Result: Four-bar Mechanism (the benchmark is solved using a six-bar mechanism while the result shown below is using a four-bar mechanism with revolute joints)

URL: http://goo.gl/hfcCqc | http://goo.gl/KGVJav
Objective function value: 0.02 | **0.0035** (as per sum of the squares of distances)
Best result from literature: **8E-5**
Results Combining search and optimization given below:
Configuration: 3-6-7-1-revolute-no prismatic | URL: http://goo.gl/VP28pW Objective function value: 1.37 | 0.16 (as per sum of the squares of distances) Configuration: 3-6-7-2-revolute-prismatic | URL: http://goo.gl/cq9r4Z Objective function value: 2.04 | 0.36 (as per sum of the squares of distances) Configuration: 3-6-7-1-revolute-prismatic | URL: http://goo.gl/tFrRq6 Objective function value: 2.28 | 0.45 (as per sum of the squares of distances)

Table 7-10: Results to Problem #10

Problem 10: x(t)=0.5*(2*sin(t)-sin(2t)), y(t)=0.5 *(2*cos(t)+cos(2t)), t is time
Result: Four-bar Mechanism (the benchmark is solved using a six-bar mechanism while the result shown below is using a four-bar mechanism with revolute joints)

URL: http://goo.gl/9Mlfxm | http://goo.gl/F2qsjK
Objective function value: 1.17| **0.12** (as per sum of the squares of distances)
Best result from literature: **1.139**
Results Combining search and optimization given below:
Configuration: 3-6-7-1-revolute-prismatic | URL: http://goo.gl/7EIlmM Objective function value: 2.08 | 0.38 (as per sum of the squares of distances) Configuration: 2-6-7-2-revolute-prismatic | URL: http://goo.gl/o8uXtp Objective function value: 0.81 | 0.06 (as per sum of the squares of distances) Configuration: 2-4-4-1-revolute-no prismatic | URL: http://goo.gl/NuD40d Objective function value: 0.95 | 0.08 (as per sum of the squares of distances)

A summary of our results to the benchmark problems is listed in Table 7-11.

Table 7-11 Summary of results on benchmark problems

Problem	Best Result from Literature (Objective Function Value)	Result from this hybrid Implementation for a four-bar mechanism	Result for Other Mechanisms from Design Space
1	0.0002	0.00007	0
2	0.0047	0.0013	0.0005
3	0.98	0.52	0.7
4	0.4154	0.15	0.21
5	0.0154	0.0007	3E-5
6	1.2162	0.26	0.001
7	0.0196	0.008	2E-5
8	0.1298	0.17	0.61
9	8E-5	0.0035	0.16
10	1.139	0.12	0.06

As shown in Table 7-11, the hybrid implementation is able to generate better results using a four-bar mechanism as well as higher order mechanisms (the best results for which are displayed in the last column on the right) for most of the problems except problem #8 and #9, which we feel is due to scaling issues in the problem (described in Chapter 8). It should be pointed out to the reader that the same algorithm was used on all problems with automatic parameter setting based on the desired path. Also shown in Table 7-1 to Table 7-10 are snapshots of results from the design space when the topology and parameters are synthesized simultaneously for each problem. The results include four-bar and six-bar mechanisms with revolute and prismatic joints. Higher order mechanisms are not shown due to computational time constraints with the facilities using which all these computations were carried out. In some mechanisms where the dimensions are very small compared to other problems, you may notice that even though the resulting objective function is a very low value (close to zero), the generated curve does not exactly match the requirements set by the user. This is one of the topics of

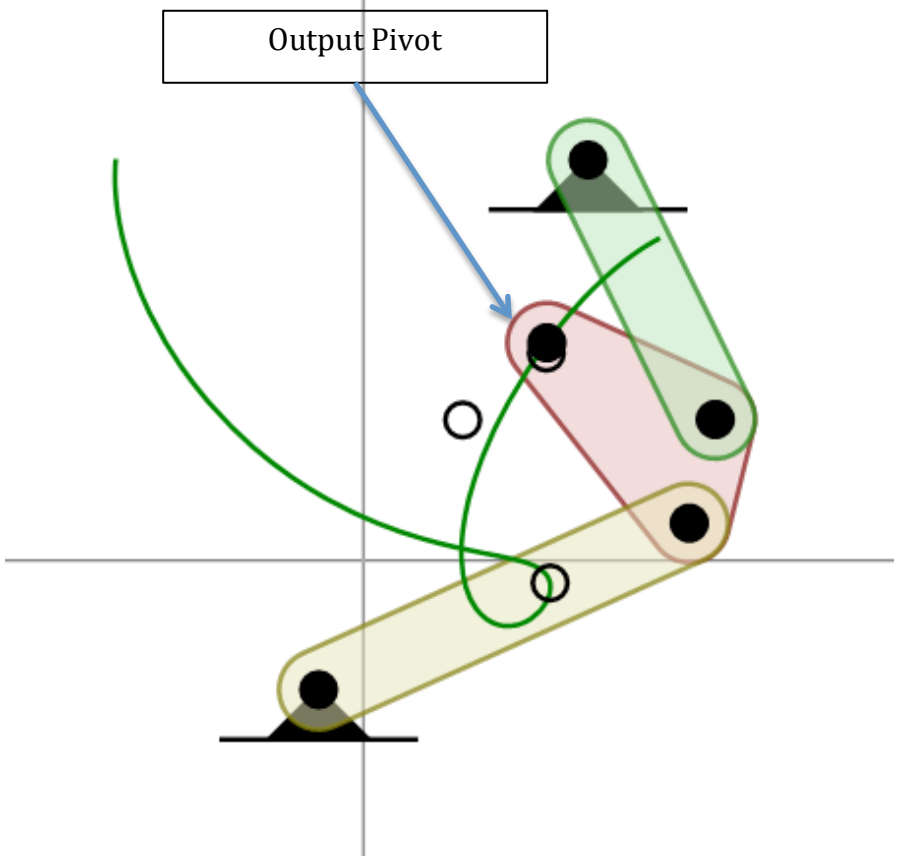
discussion in the next chapter where if the method is not scale sensitive, erroneous results can be obtained and does not bode well in the long-term usage of such methods in design.

7.2 SOLUTIONS TO CHALLENGE PROBLEMS

7.2.1 Challenge Problem #1

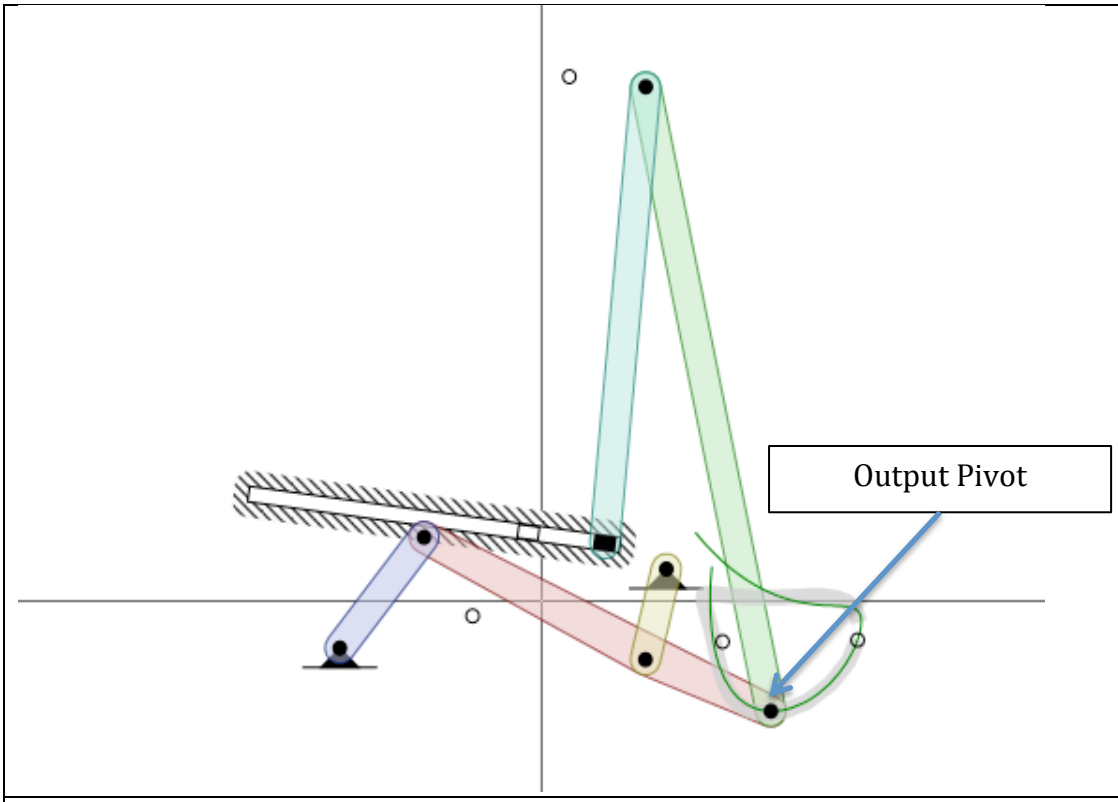
The solutions to challenge problem 1 (refer Chapter 3 for data) are presented in Table 7-12 below. The first solution is obtained using a four-bar mechanism consisting of only revolute joints in the actual scale specified in the problem description. The remaining two solutions are obtained for the curve whose scale is increased by a factor of 10. The best solutions for the second case are obtained using 6 bar mechanisms with both revolute and prismatic joints and are shown in Table 7-13.

Table 7-12: Results to challenge problem #1



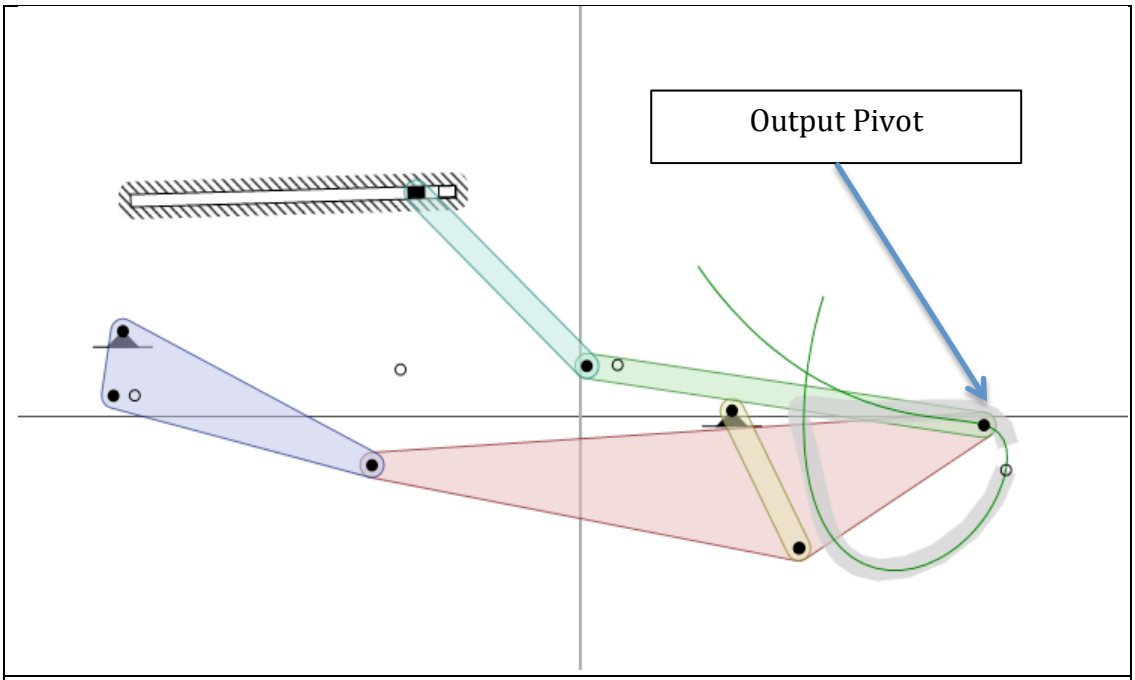
URL: http://goo.gl/1tnC4l
Objective function value: 2.80
Configuration: 2-4-4-1-revolute-no prismatic

Table 7-13: Results to challenge problem #1 using a different scale



URL: http://goo.gl/1W7wwX
Objective function value: 11.79	4.05 (as per sum of the squares of distances)
Configuration: 3-6-7-2 -revolute-prismatic

Table 7-13 continued.



URL: http://goo.gl/0sm9O7
Objective function value: 10.94	3.49 (as per sum of the squares of distances)
Configuration: 3-6-7-2-revolute-prismatic

From the above results, a four-bar mechanism is not predicted for the scale variant of the problem in Table 7-13. You may also notice that the solutions obtained are not quite close to the required goal. One possible reason is that the number of data points on the desired path is much higher than other problems. This increases the computational time for calculating the objective function value since 36 different coordinates have to be checked for each iteration of the algorithm within which there are several more function evaluations carried out. Since the computational resource available to us does not permit longer computational times, we are unable to check for better results using higher order systems or for that matter even using four-bar mechanism. Also, the mechanism is constrained to follow the path exactly due to the presence of more data points and this increases the complexity of the search process. But since the original application is part

of a conveyor (refer Chapter 3), the important section in that path is the straight-line section. Suppose the desired path is changed to a straight-line as against the original, the result produced is given below in Figure 7-2. This shows that it is possible to obtain different results that may be better just by virtue of changing the desired path.

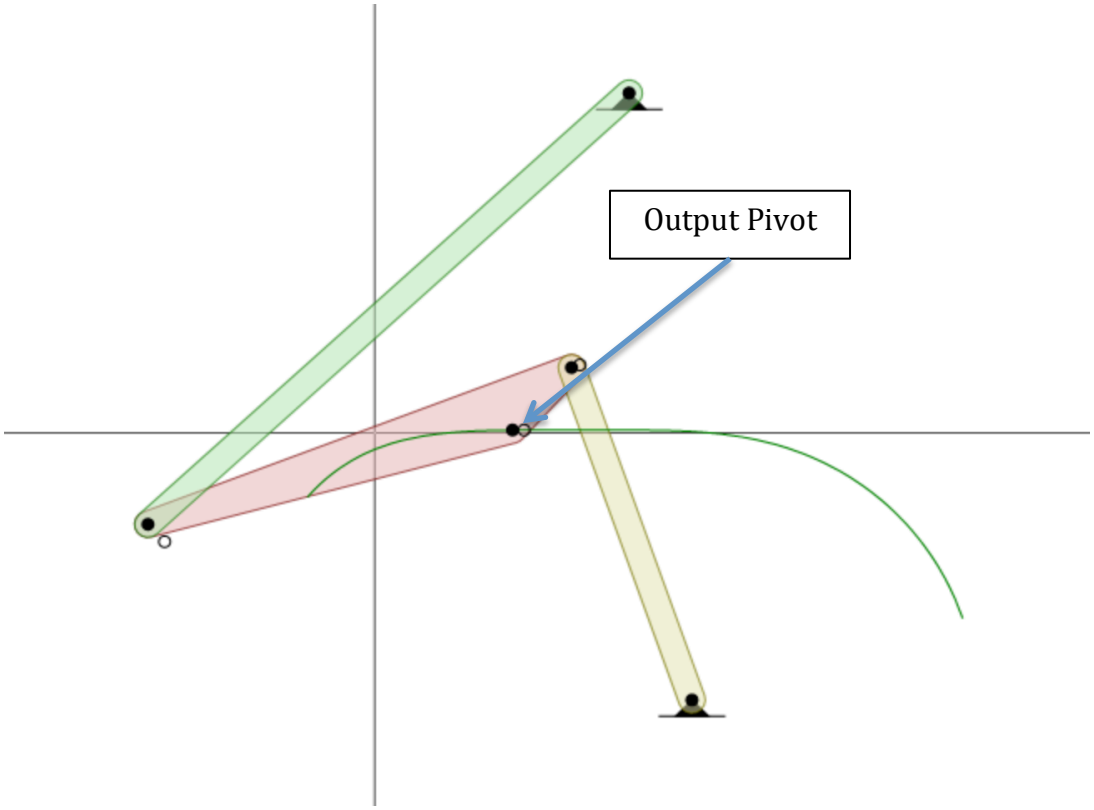


Figure 7-2 Modified challenge problem #1 (URL: http://goo.gl/65svrI)

7.2.2 Challenge Problem #2

As mentioned in the previous chapter, this challenge problem has been approached as a path-tracing single-input multi-output problem. The results presented here are only for the case 1 wherein rule #1 from rule set #4 is applied to generate the appropriate candidate (refer to Chapter 4 for discussion). Table 7-14 and Table 7-15 present two results for this case. Table 7-14 also displays the path traced by the four joints in the actual biological animal.

Table 7-14 Results to challenge problem #2

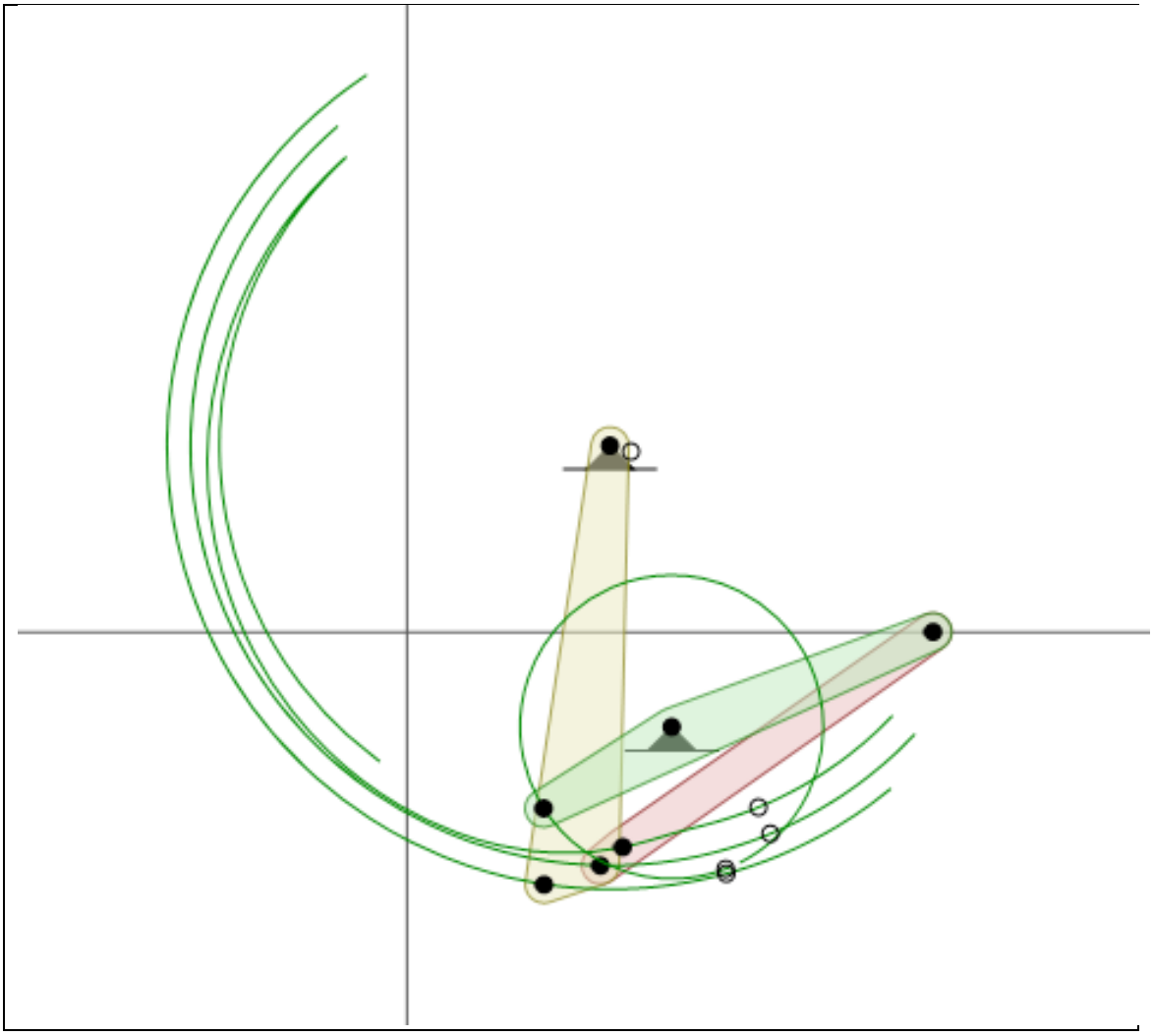


Table 7-14 continued.

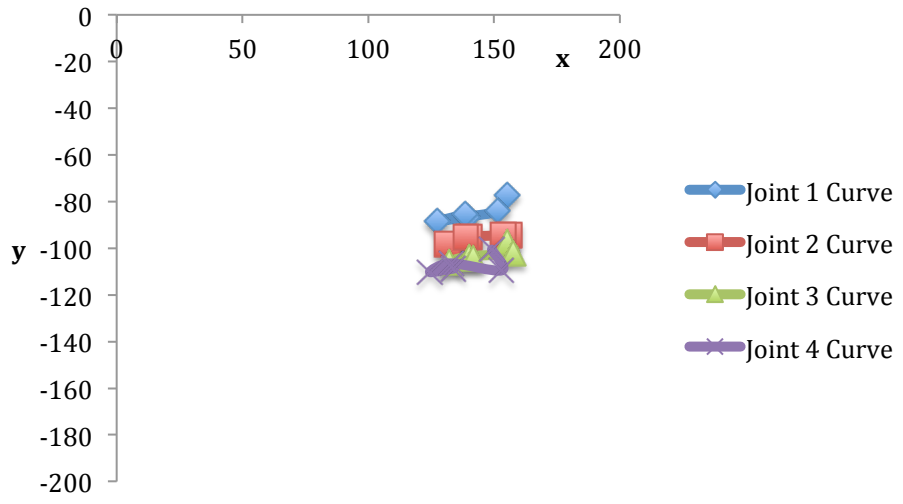
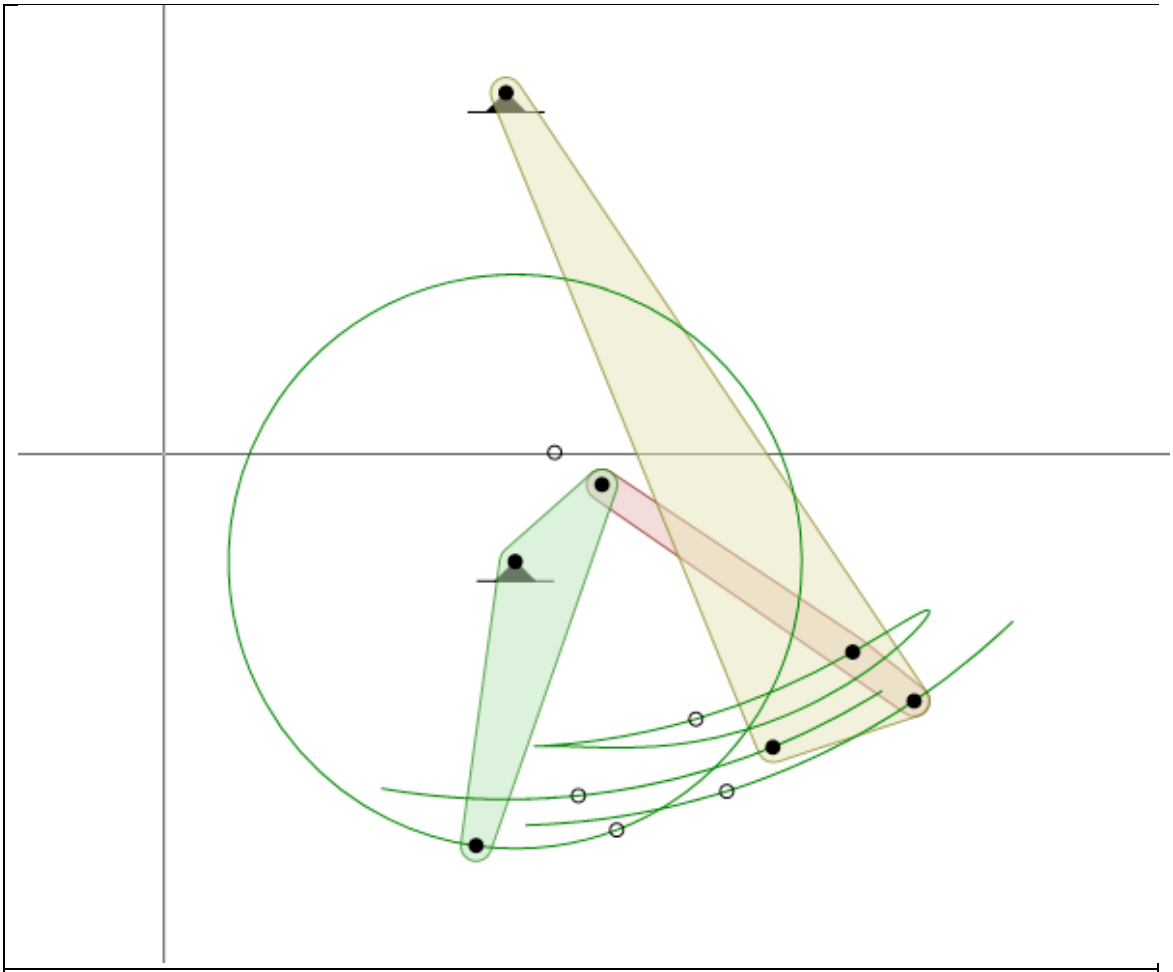
Desired Curve: 
URL: http://goo.gl/jzZlz5
Objective function value: 6.46

Table 7-15 Results to challenge problem #2



URL: http://goo.gl/U4q7Ja
Objective function value: 7.38

As you may see from the results and comparing those that with the desired path, we can find that from a pure position synthesis point of view (leaving aside computational constraints), we are able to obtain an average objective function value of around 7. It should be noted that the given positions are absolute values from the reference image of

the coconut crab (refer to Chapter 3). But in reality, it is important to obtain relative positions of the joints so that a more meaningful result can be obtained.

The results after applying rule #2 from rule set #4 have not been displayed since the computational time required for the particular mechanism using our approach exceeded the permissible limits of the facilities at our end.

7.2.3 Challenge Problem #3

The results to the challenge problem #3 are shown in the tables below. This problem requires tracing the logo of the University of Texas at Austin. Table 7-16 shows the result for the single-input multi-output scenario. Each figure shows the section of the curve traced by a particular pivot (also highlighted by showing on the desired curve).

Table 7-16 Results to challenge problem #3

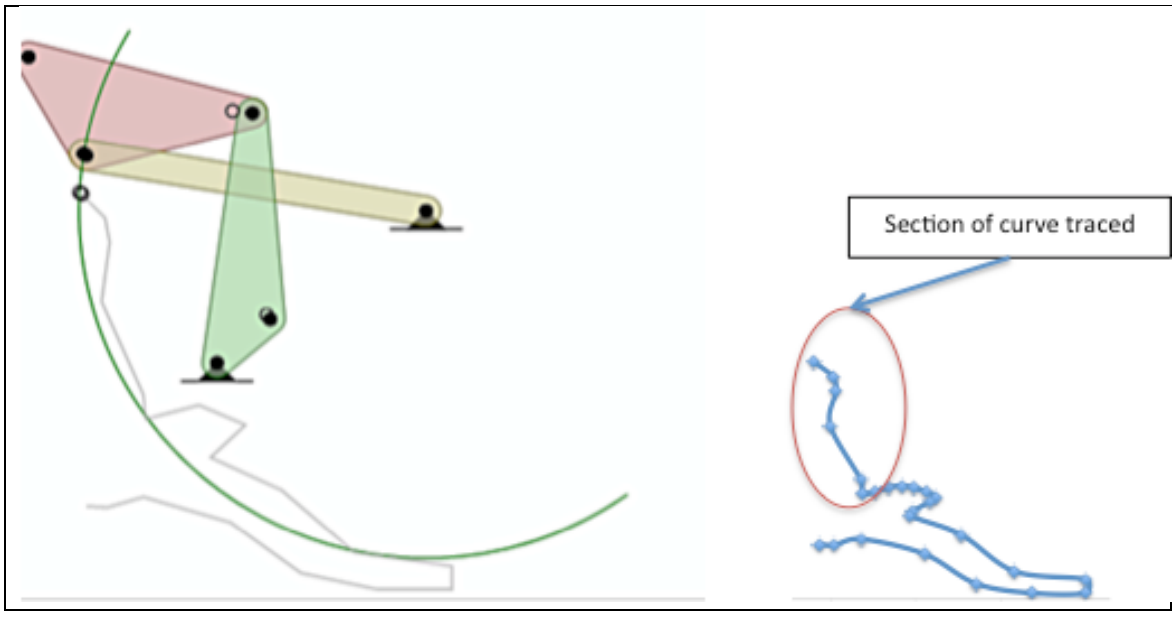


Table 7-16 continued.

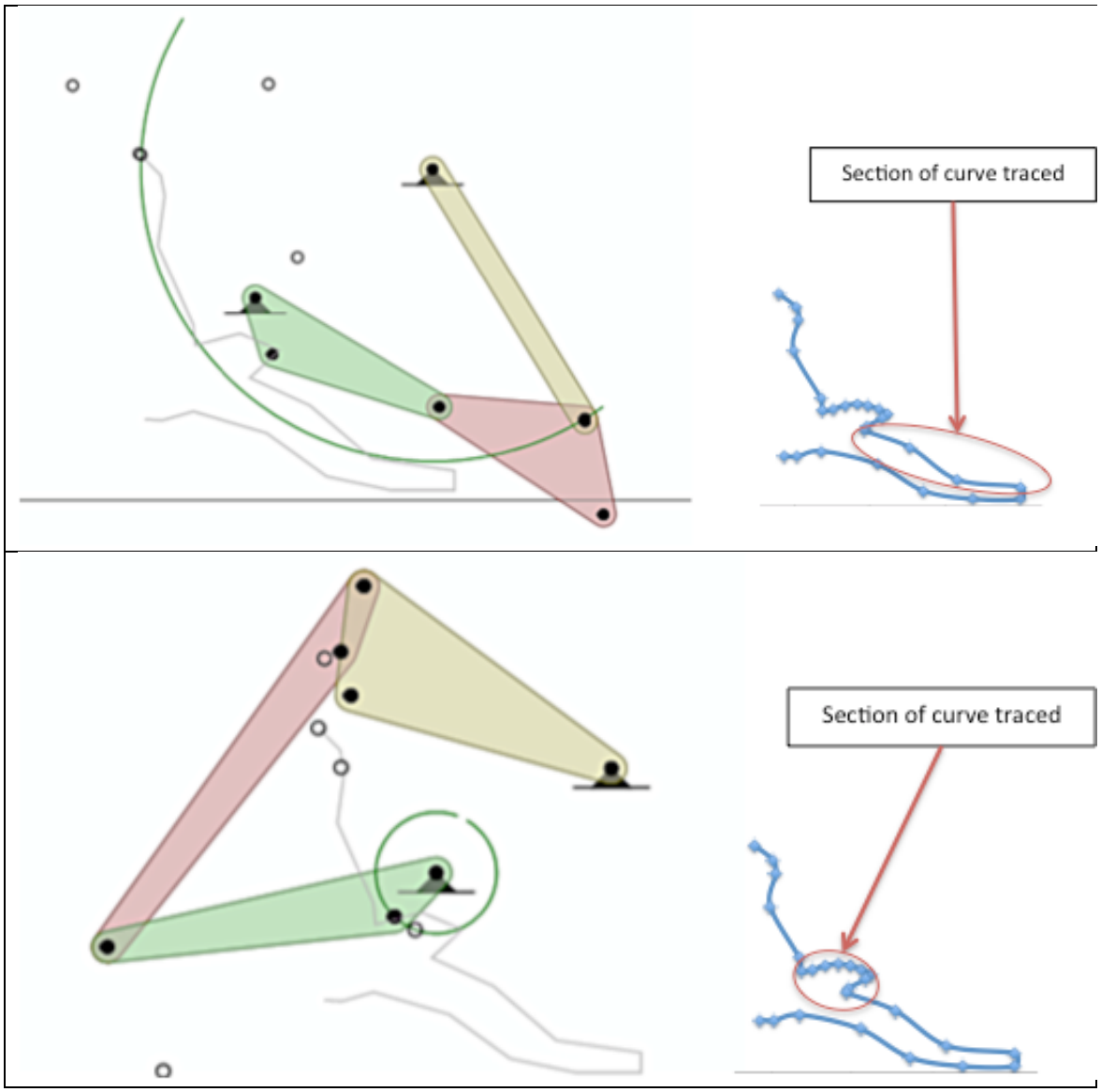
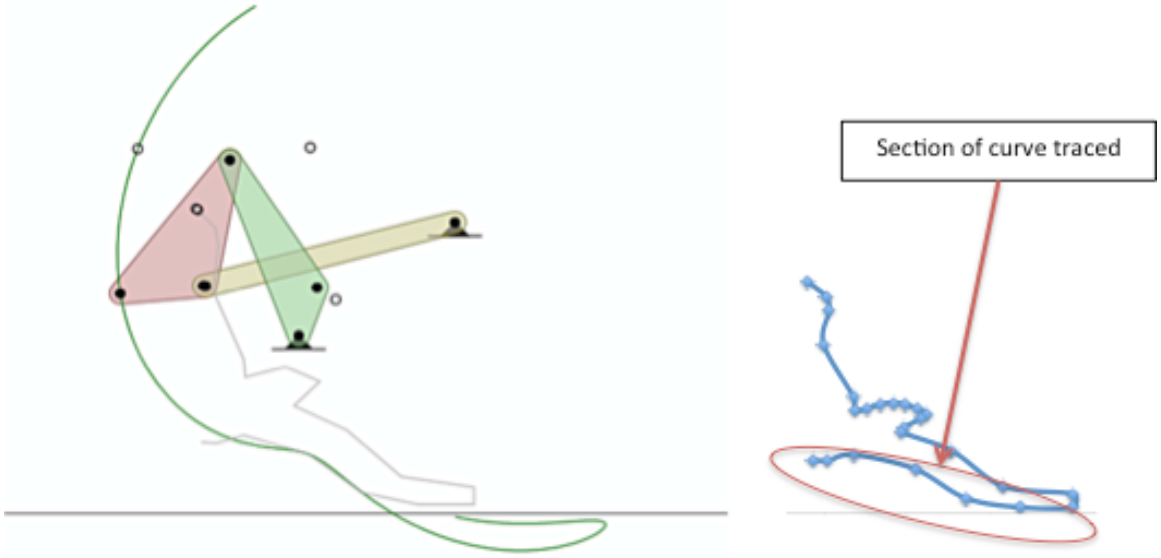


Table 7-16 continued.

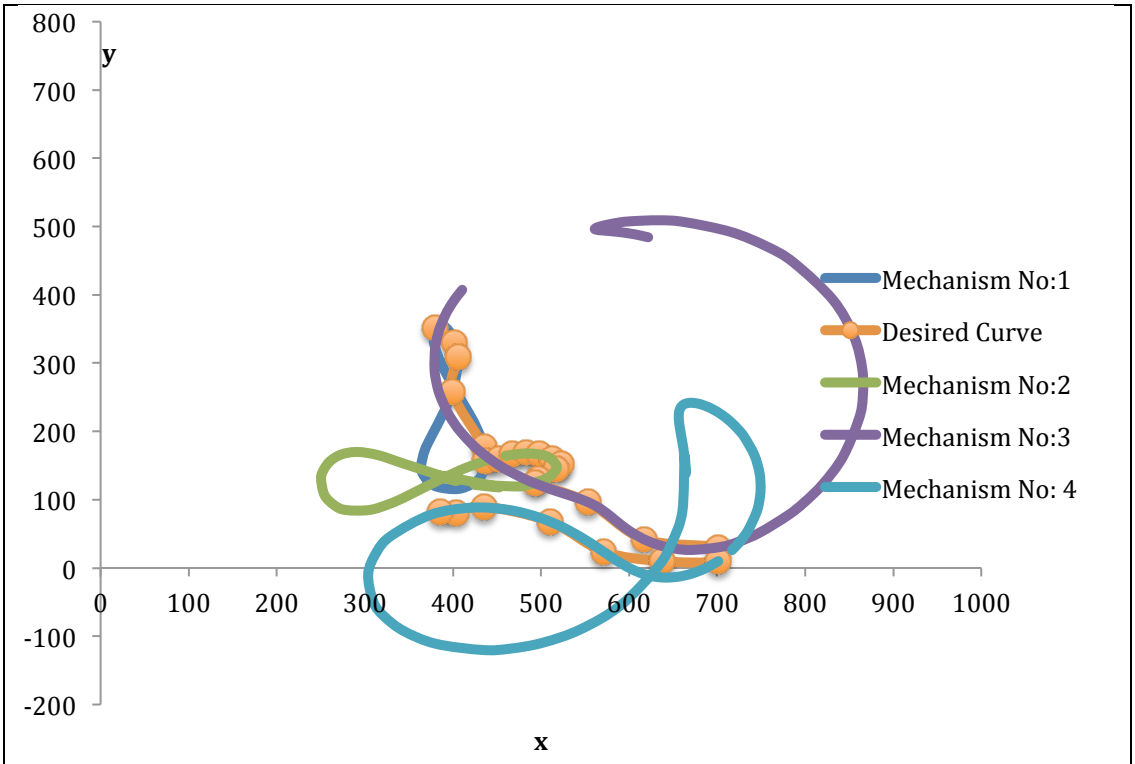

URL: http://goo.gl/D0E2Jb
Configuration: 2-4-4-3-revolute-no prismatic
Objective function value: 79.09 (sum of distances)

From the above result, it can be seen that there are certain segments of the overall curve that are traced better than the rest. Only a four-bar mechanism is shown in this result as due to computational time limitations, we were unable to produce results with a six or higher-bar mechanisms that may have been a better synthesis candidate. Also, during optimization, only the node with "output" label (ref Chapter 4) is assigned a point from the desired path and is not subject to optimization while the positions of nodes with labels namely "output1", "output2" and "output3" are determined by the optimization. Assigning a point for these graph nodes may significantly improve the result obtained in such single-input multi-output mechanisms since the number of variables being optimized is significantly reduced. It can also be seen from the result in Table 7-16 that since there are certain sections of the curve that have fewer points, the generated mechanism is able to match a few points but not able to produce the exact curve. This

highlights the need for an optimum number of points in the desired path such that the trend shown in the desired path can be obtain during synthesis.

The other approach adopted here is the scenario where different mechanisms trace different sections of the curve. Due to restrictions in the time of computation available to us, we were unable to completely automate this process. Instead, we synthesized mechanisms for individual curves separately and manually selected different mechanisms to produce the following results. This way, we are able to show that the technique is promising and alternate ways to improve the usage of computational resources can be explored to produce better results. The following tables (Table 7-17, Table 7-18 and Table 7-19) display the results using this approach. The links for the mechanisms that trace different sections of the curve are presented. The dotted line is the desired curve and the mechanisms are able to cover most sections of the desired curve. All the mechanisms obtained are four-bar mechanisms with revolute joints. Though a few six bar mechanisms were also obtained, they were not selected since their error was higher than what was obtained using a four-bar mechanism.

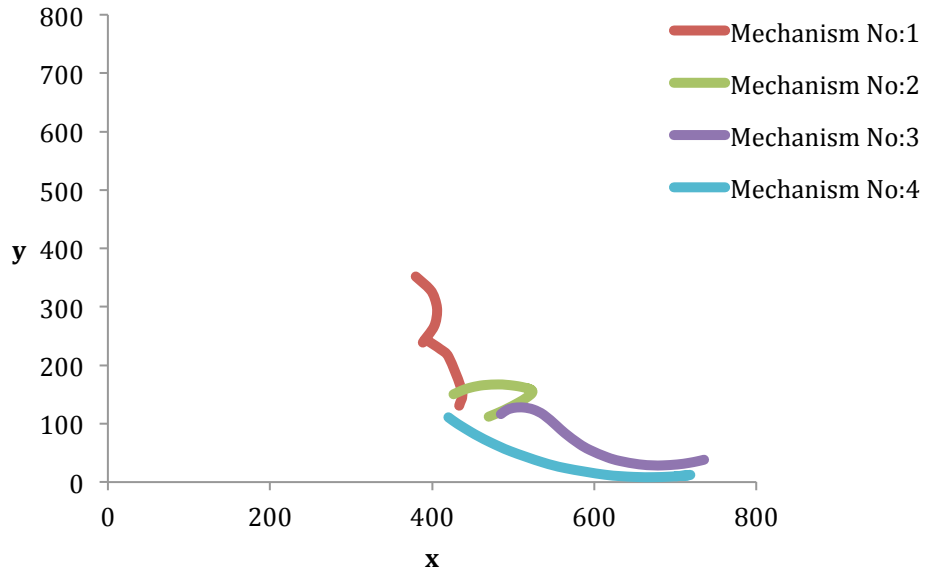
Table 7-17 Results to challenge problem #3



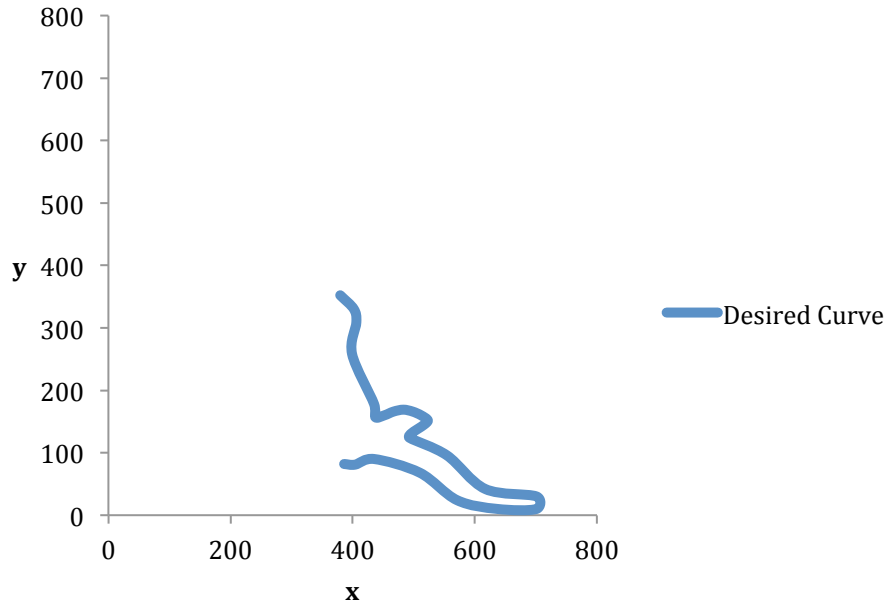
Mechanism 1	Mechanism 2	Mechanism 3	Mechanism 4
URL: http://goo.gl/iJcvrV	URL: http://goo.gl/r03ry2	URL: http://goo.gl/XQGJsN	URL: http://goo.gl/ojvYnE

Table 7-17 continued.

Overall Result:

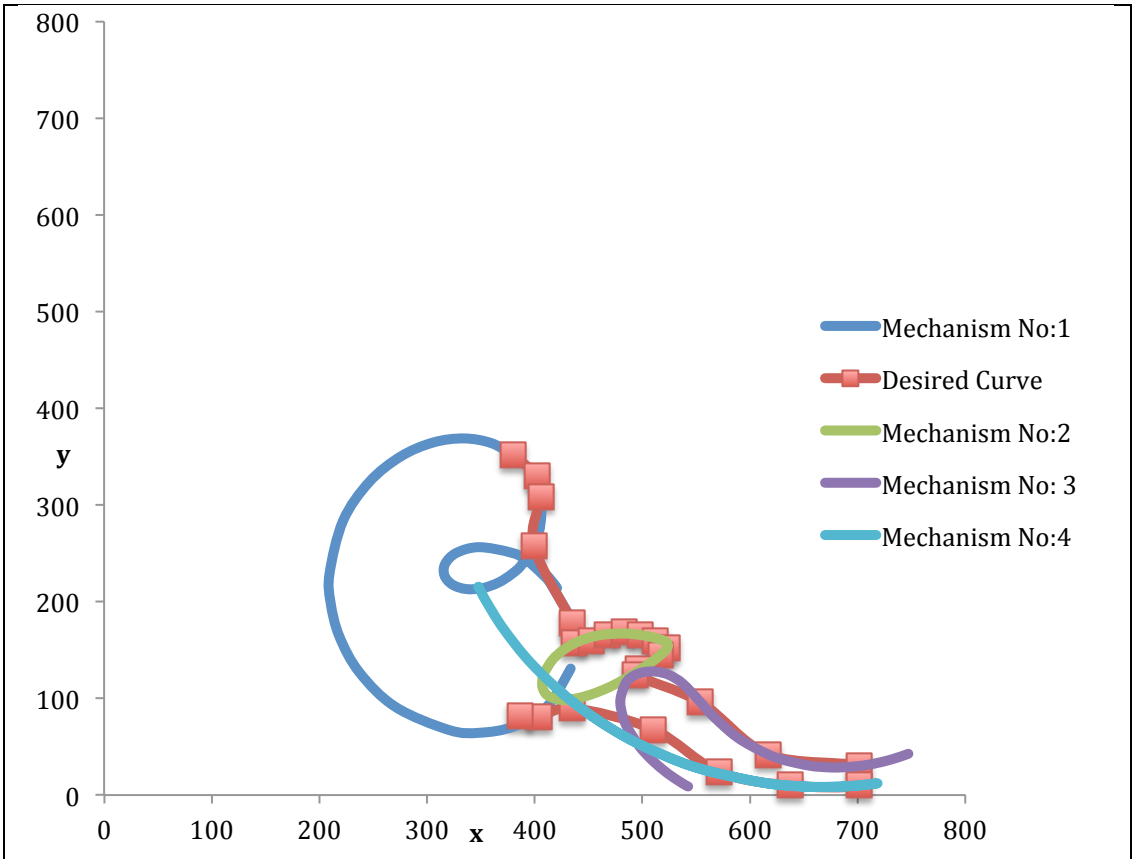


Desired Result:



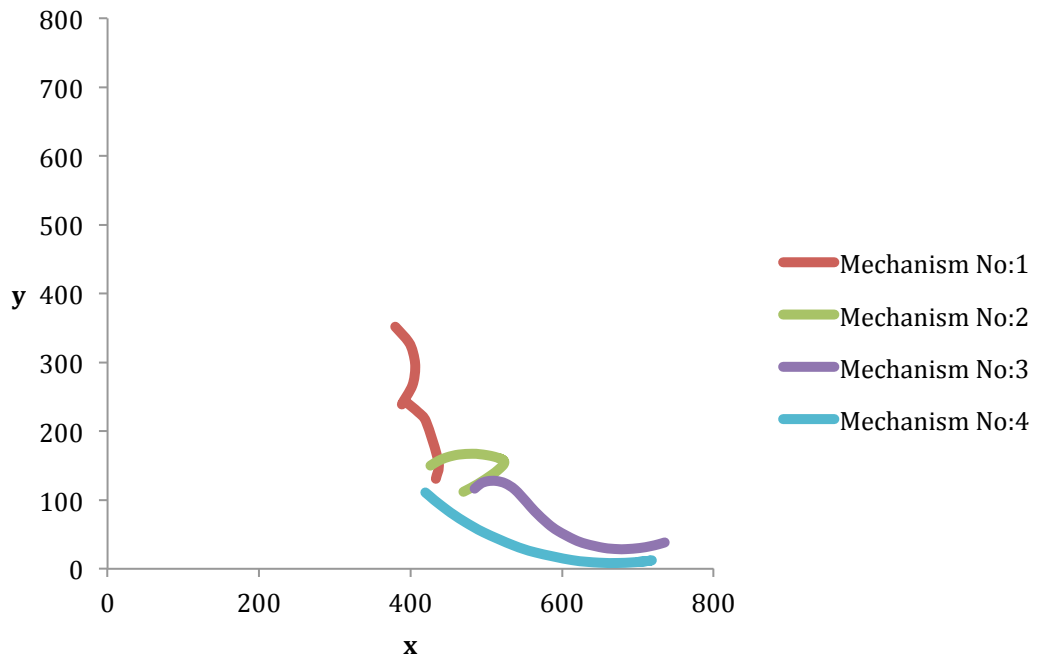
Objective Function Value: 5.76

Table 7-18 Results to challenge problem #3



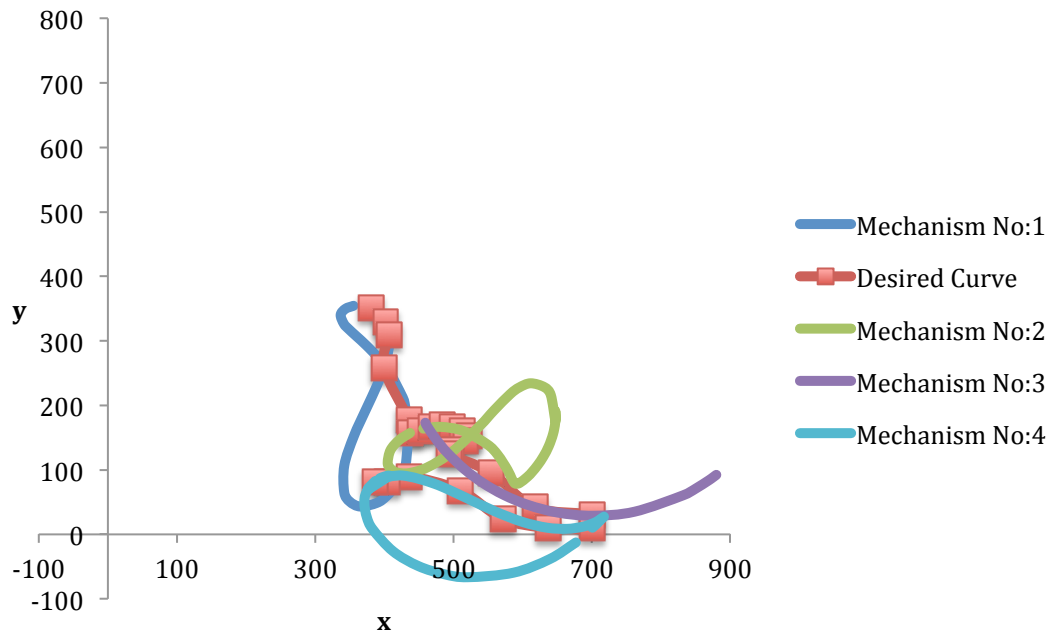
Mechanism 1	Mechanism 2	Mechanism 3	Mechanism 4
URL: http://goo.gl/c4rsVQ	URL: http://goo.gl/pasFtE	URL: http://goo.gl/I5EIIE	URL: http://goo.gl/NoH7Xl

Table 7-18 continued.



Objective Function Value: 8.55

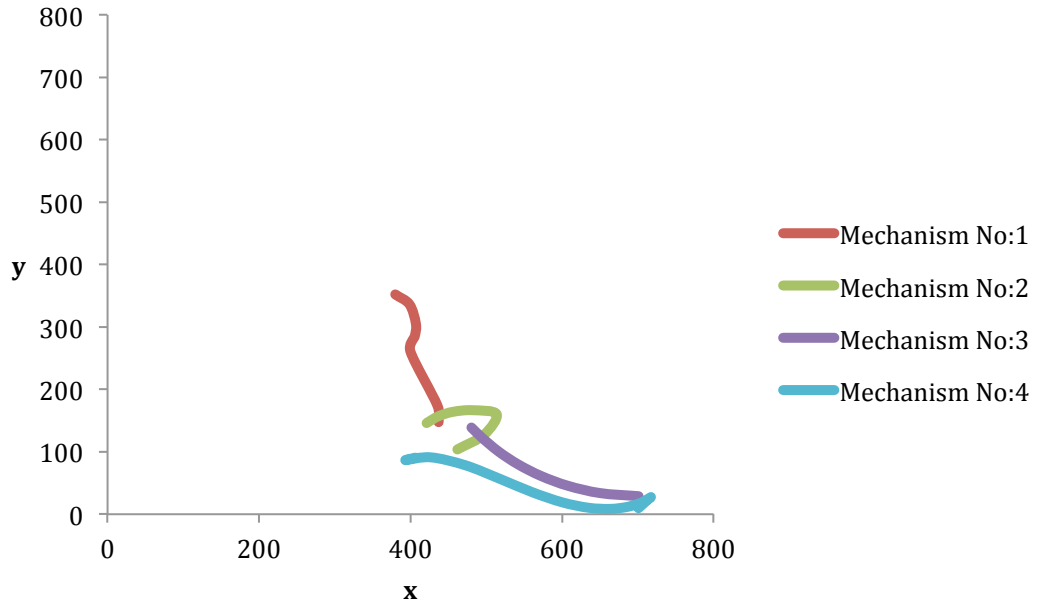
Table 7-19 Results to challenge problem #3



Mechanism 1	Mechanism 2	Mechanism 3	Mechanism 4
URL: http://goo.gl/CQZ3y0	URL: http://goo.gl/xBVRW3	URL: http://goo.gl/k7isZF	URL: http://goo.gl/WoXY82

Table 7-19 continued.

Overall Result:



Objective Function Value: 8.86

You may notice from the three results that by using multiple mechanisms, we are able to produce a result that is very close to the desired curve. Our view is that if the number of points that describe the desired path is increased, better results may be obtained but this is subject to increased computational expense. It should be also pointed out to the reader that we have employed the technique of increasing the number of points on the desired path for the second section of the curve in the above results for this challenge problem. If the constraints are changed, for instance, that the mechanisms should not interfere with one another, then the synthesized result will be totally different. All these different possibilities are discussed in the next chapter.

7.3 CONCLUSION

The results obtained using our method for the benchmark problems are better than the existing results from the literature for most of the problems. In addition, alternate mechanisms for those benchmark problems that were generated by combining tree-search and optimization are also presented. Three challenge problems have been attempted and their results are shared. With better computational resources and employing better coding schemes to take advantage of parallelization, the algorithm may yield better results.

Chapter 8: Discussion

In this chapter, we will discuss the results presented in the previous chapter. Section 8.1 will focus on the insights gathered during algorithm development and during various experiments performed to understand the search space. Section 8.2 will discuss the results of the benchmark problems, which will be followed by comments on the challenge problems in section 8.3. Computation time has been a major constraint in this research and section 8.4 will discuss some of the activities carried out in that area. Conclusions will be presented in section 8.5.

8.1 Algorithms and Search Space

The results to the benchmark problems that are presented in this dissertation are better than those in the literature for most problems. But towards the end of this section, we will describe how these results are not guaranteed all the time. This is a function of the different constraints and the size and shape of the search space. In order to understand how different parameters affect the solutions obtained, several experiments were performed on various aspects of the problem and the details are presented in the subsections below.

8.1.1 Constraints and Problem Definitions

First let us consider the constraints used during the dimensional synthesis of planar mechanisms. In the literature, several constraints are used such as Grashof's criterion and length constraints between different pivots in the four-bar mechanism. Through these constraints, the size of the search space is significantly reduced although it is not necessarily easier to navigate. In our implementation, we are using a bounding box constraint where the links may take up any length but the complete position kinematics of the mechanism should lie within a specified region. In addition, we also specify the maximum and minimum limits for each joint position in the mechanism to enable us to

generate the starting vector using the Latin Hypercube Sampling technique. But once the best starting vector has been determined, the particles in Particle Swarm Optimization (or vertices of the simplex of Nelder-Mead simplex algorithm) will take up other positions during the course of swarm movement yielding better solutions to a problem. For instance, the initial set of particles are generated within an initial box, but due to the swarm movement, the final solution is enclosed within a second box while still satisfying the bounding box constraint. If one of the particles is assigned a position that exceeds the bounding box, then the particle is reinitialized and the process is continued.

So it is possible that the initial maximum and minimum bounds that were assigned before generating the starting vector is no longer enforced during optimization. The only parameter that is enforced is the mechanism's bounding box. This allows our search space to not be reduced unlike what is enforced through constraints in other methods. Additionally this relaxation could also be beneficial in enabling to better synthesize the mechanisms. This is like formulating a less constrained problem to an over constrained problem. In addition to the bounding box constraint, we are using another constraint to space the "input" grounded pivot at a slight distance away from the other grounded pivots. We will explain the reason for this constraint using the example given below in Figure 8-1 whose URL is http://goo.gl/nvYxYF.

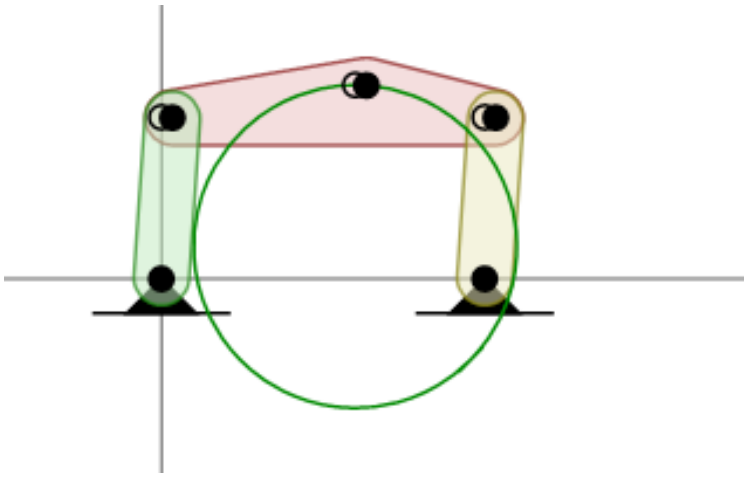


Figure 8-1: Four-bar mechanism used to explain "input spacing" constraint

This is a four-bar mechanism where input pivot is aligned with the two axes at (0,0) and the other grounded pivot is at (10,0). The ternary coupler pivot that traces the circle is at (6,6), the joint between the coupler and the input is at (0,5) and finally the joint between the coupler and follower is at (10,5). The arbitrary circular path generated by pivots (6,6) will serve as the optional goal and the set of random neighbors are generated around this known vector and used as the starting vector in our search process. An example of the neighborhood position will be (-1,0) instead of (0,0); (11,0) instead of (10,0) while keeping the other positions the same. One would imagine that since the vector is only slightly perturbed from the actual solution, the optimization would easily find the original mechanism as the solution. Therefore, 28 random neighborhood positions differing by no more than 2 units in all positions were generated to check if the optimization algorithm is able to produce the original result. Out of the 28 starting vectors, only 11 starting vectors produced objective function values of less than 0.5 (i.e., with a value of 0.5 sum of distances from the desired circle). The results are shown below in the Figure 8-2 (the coupler point (6,6) is not included) and none of the results (each result is termed as a series) ever produce the same original joint positions. This is possible considering the nature of the desired curve and that there are infinite solution possibilities. But what is intriguing is the location of the grounded pivots that are shown using a dotted circle in the same figure. Most of the mechanisms have their grounded joints very close to each other. This led us to introduce the "input spacing" criterion so that the grounds are spaced apart.

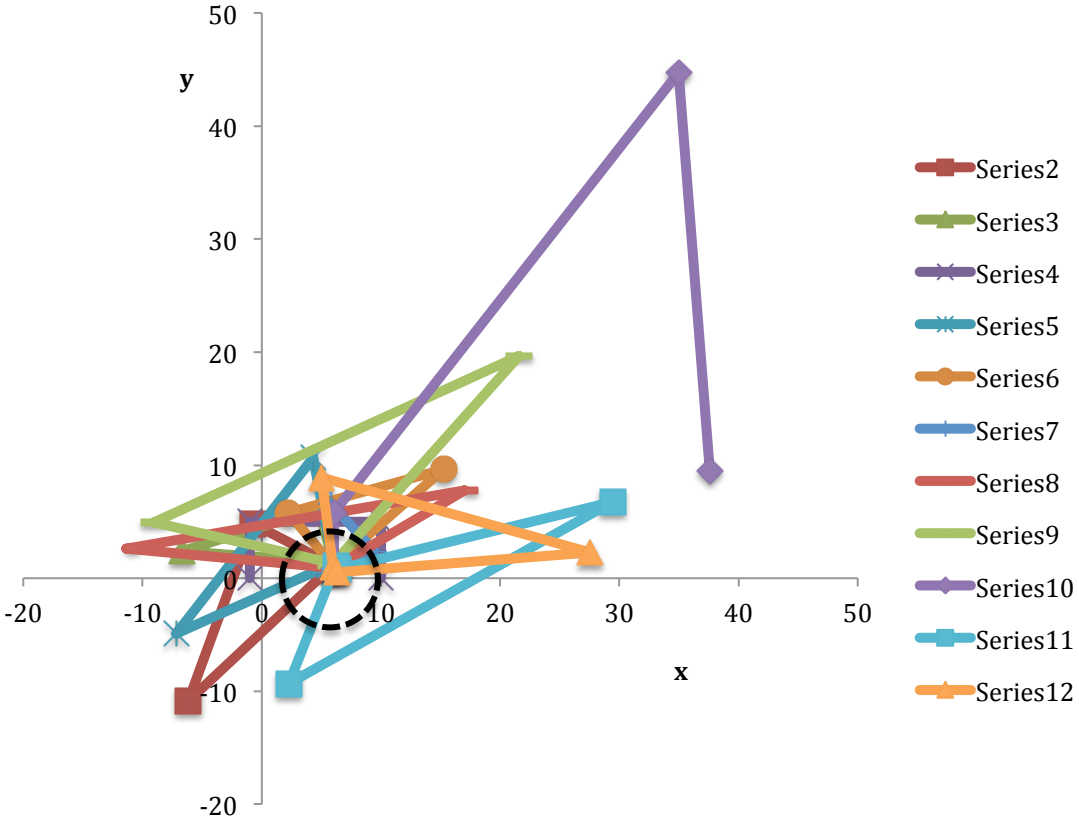


Figure 8-2: Results of applying optimization algorithm to random neighborhood points

The input spacing is also an inequality constraint and for test purposes we have been using values of 1 or 2 units between ground joints. This could be increased for problems with large dimensional scales. The results for the benchmark problems as well as the challenge problems do not have grounded joints close to one-another primarily due to this constraint. It is also clear from this discussion we are obtaining near-optimal solutions for only 40% of the trials. This trend was spotted while running separate instances of Nelder-Mead algorithm and Particle Swarm Optimization as well as on the hybrid implementation presented in this dissertation. This will be dealt further in section 8.1.2.

An important aspect in our technique is the removal of the "output" labeled pivot from optimization consideration. This not only reduces the number of dimensions in the problem but also significantly speeds up the computation. The thinking behind this was that since the desired path has to follow all the points anyway, we might as well assign one of those points to the output location. Like the moving bounding box, this reduction alleviates difficulties in the search without reducing the generality of the resulting solutions. In terms of other constraints such as the bounding box specification, the norms adopted in Chapter 6 are valid for a generic class of problems.

8.1.2 Search Space

In the previous subsection, it was mentioned that the lack of additional constraints makes the search space huge. Also, in the experiment carried out in the previous subsection (related to Figure 8-2), only about 40% of the starting vectors resulted in a near-optimal solution. This may be due to the presence of discontinuities and/or local minima in the search space. This is also validated by a review of the solutions obtained for three benchmark problems (#1, #2 and #3) at the seventh level in the search tree as shown in Table 8-1. As can be seen from the table, of the 24 candidates evaluated, the optimization is able to produce near optimal solution on all the candidates only for the first benchmark problem (sample results in Table 7-1) whereas the second benchmark problem (sample results in Table 7-2) has none and the third (sample results in Table 7-3) has only one solution. These results are at one instant in time and at a different instant, entirely different results may be predicted. To understand this variability, more studies on the search space are conducted.

Table 8-1 Number of solutions generated for three different benchmark problems at level 7 in the tree-search

	Benchmark Problem 1	Benchmark Problem 2	Benchmark Problem 3
No of Solutions	24	24	24
Near Optimal Solutions / Better than Existing Results	24	0	1

To better understand the complexities of the space, consider the example used in Figure 6-4, where an arbitrary four bar mechanism consisting of revolute joints and a ternary coupler link is shown. The input joint is located at A(21,3) and the other joints are B(25,16), C(12,24), D(2,7) and E(15,5). The curve produced by E is set as the goal and the maximum width and height of the bounding box are each set at 50. A set of 25 random neighborhood positions is generated around the original solution for the pivots A, B, C and D. The pivot E is not part of the optimization since the output is always assigned a coordinate (x, y) from the desired path. Due to this, the number of variables in the problem is eight (each (x, y) position correspond to 2 variables). The objective function with respect to each of these random neighborhood positions is calculated and shown in Figure 8-3 below.

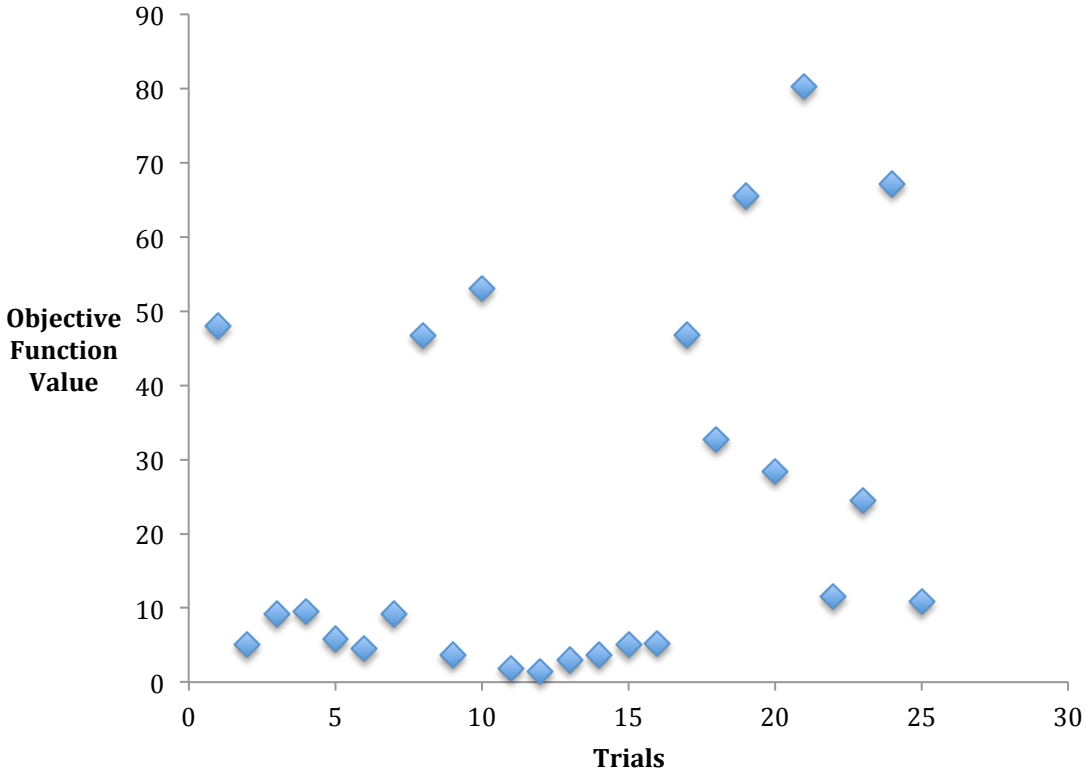


Figure 8-3: Objective function values for different neighborhood positions for the four-bar in Figure 6-4

The above figure (Figure 8-3) shows the objective function values for different random neighborhood positions of the pivots (A, B, C and D). For example, a slight perturbation of point A from (21,3) to (20,3) produces an objective function value of 48 i.e., the sum of the distances between the points traced by the curve using random position and the set of points describing the original curve is 48.

Let us now walk along the unit vector starting from this new position to the original position and beyond and see how the objective function values are changing. This trend is shown in Figure 8-4 below.

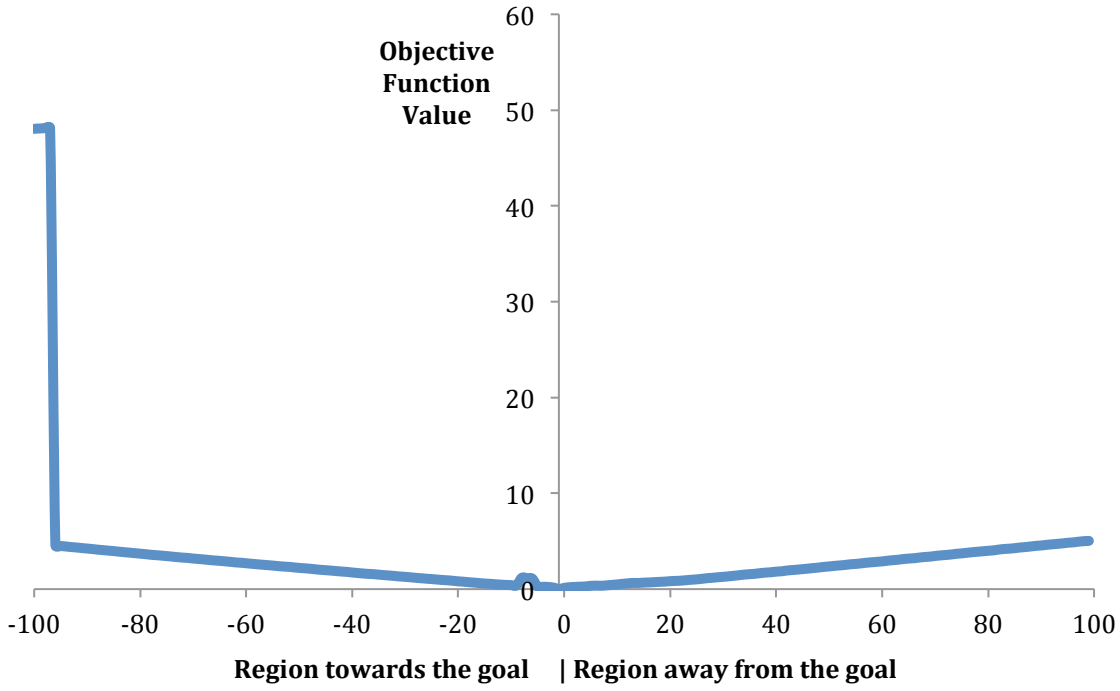


Figure 8-4: Objective function values along a unit vector around the original solution

The plot may be considered to be equivalent to slicing the 8 dimensional space (since there are 4 pivot positions that are optimized and each position is defined by (x, y)) and observing the trend that lies within. It can be seen that the objective function undergoes a drastic drop and as it proceeds towards the actual solution, there is a slight increasing trend in the objective function value (noticed around -10 from the origin 0). The drastic drop is because the Grashof's criterion was not satisfied at the initial position. While moving away from the original solution along the same vector (between trials 100 and 200), we do not find any pronounced inflections in the objective function value. But contrast this with Figure 8-5 below, which is from a different neighborhood position and it can be seen that the region on the right has several inflection points that could affect the algorithms' performance and result in a poor solution with a high objective function value.

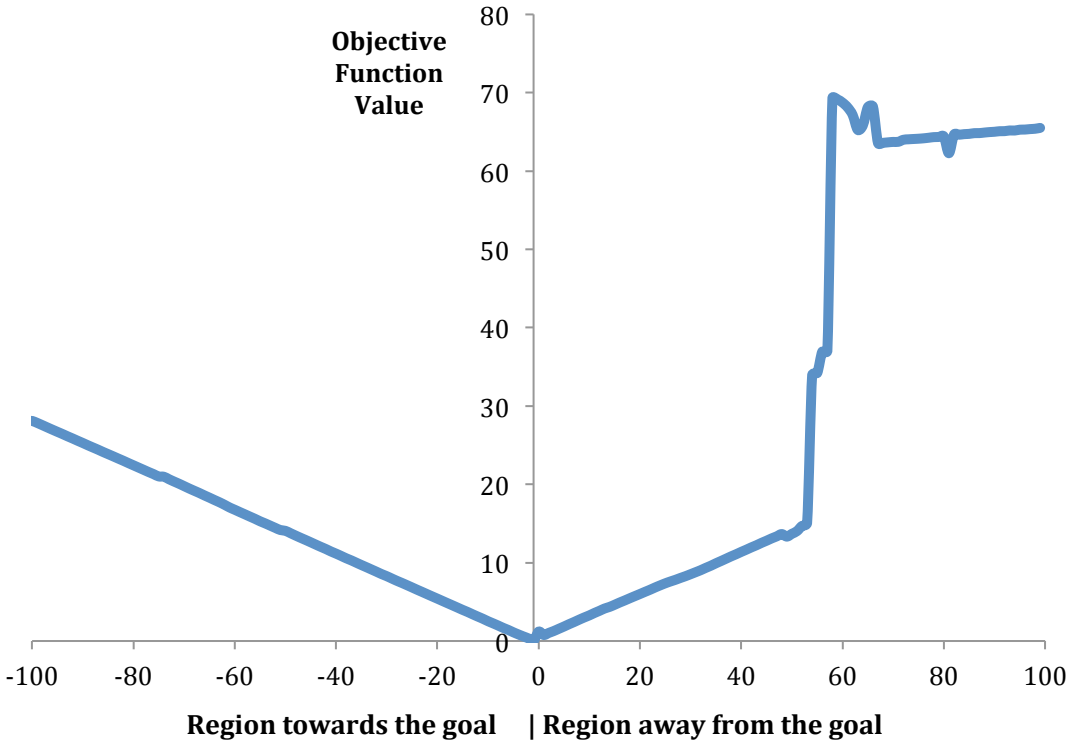


Figure 8-5: Objective function values along a unit vector around the original solution

Now, we will use an optimization algorithm such as Nelder-Mead simplex algorithm to illustrate how this algorithm is able to navigate the search space using the first neighborhood point where A is (20,3). The starting vector is therefore (20,3,25,16,12,24,2,7). For about 100 iterations, the trend produced using Nelder-Mead Algorithm is shown in Figure 8-6 and the vector produced after these iterations is (21.12,3,25.01,16.02,12.02,24.10,2.07,7.06). Though the starting objective function value is 48 as observed in the trend shown in Figure 8-4, we are not including the starting value in our plot in Figure 8-6 so as to present a clearer trend in the objective function value produced during optimization.

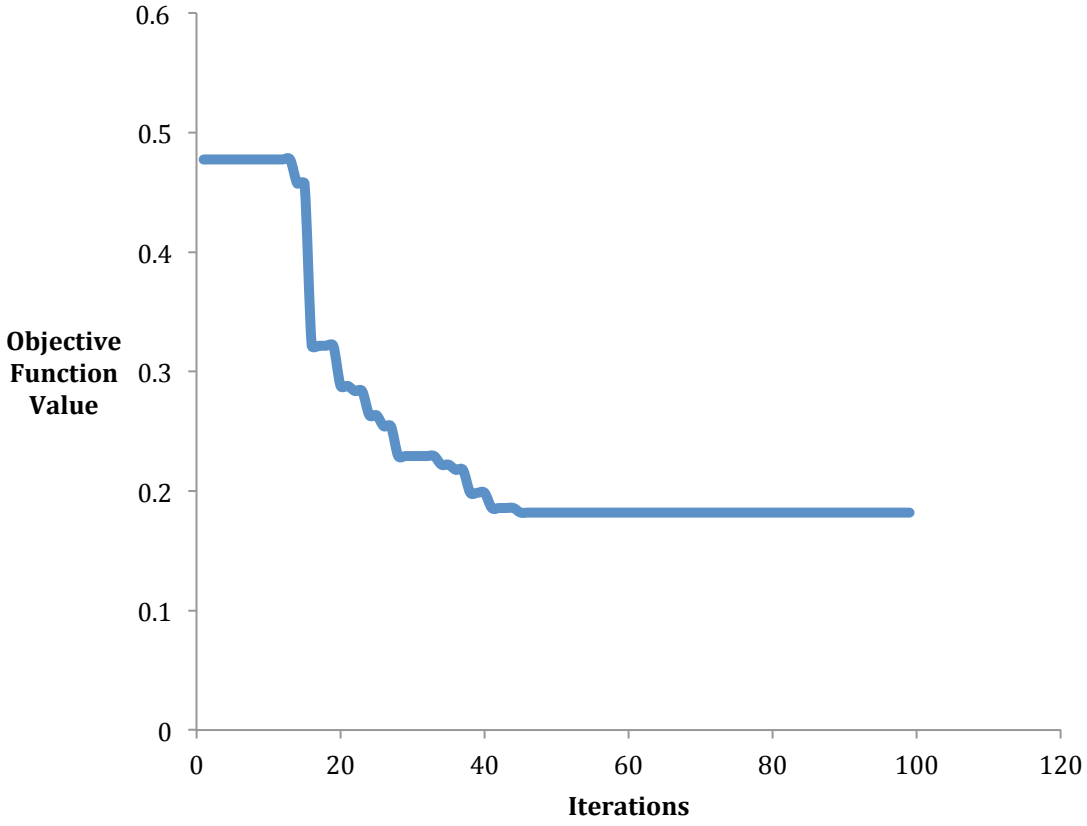


Figure 8-6: Objective function values obtained using Nelder-Mead optimization for a neighborhood point

The algorithm is able to produce a low objective function value but is stagnant around 0.2. Let us examine this space by moving along a unit vector generated based on the output from the algorithm to the actual position. That trend is shown in the figure below.

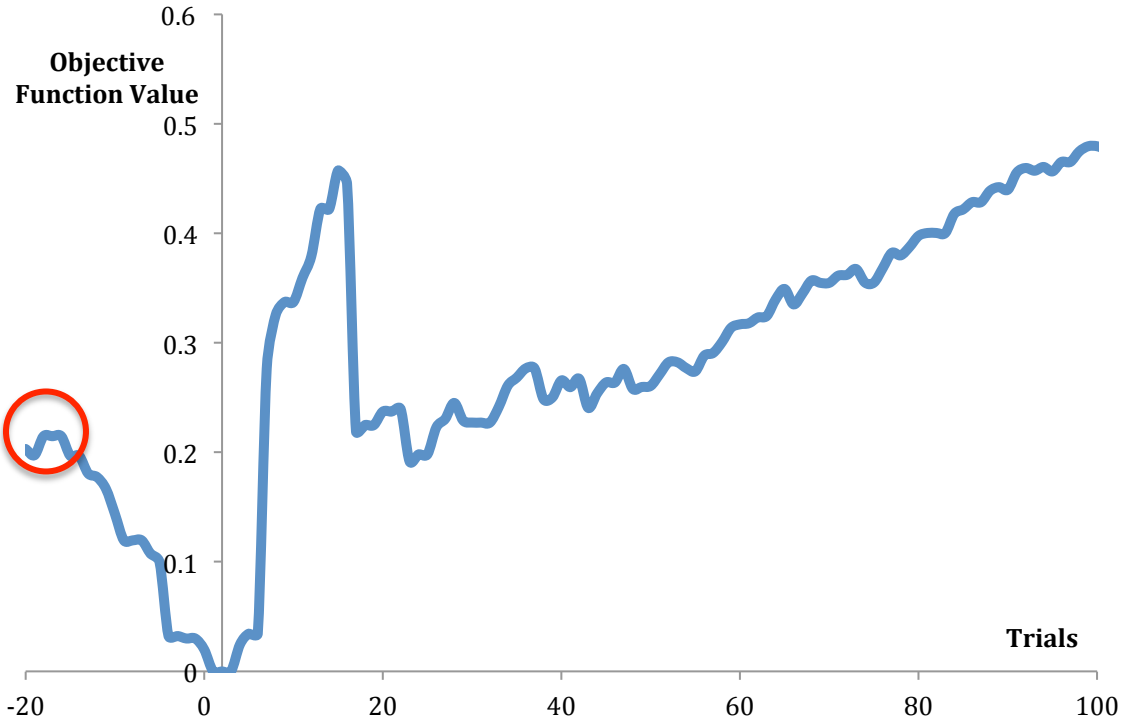


Figure 8-7: Objective function values trend between stagnation point in Figure 8-6 and the original solution and beyond

The trend in Figure 8-7 clearly shows the presence of several points of inflection. Due to these points, the Nelder-Mead operations on the simplex are resulting in poor values around the existing region as indicated by the oval on the figure and this probably explains the stagnation in the objective function value produced in Figure 8-6. This trend is witnessed in several other examples too. Due to the search space being so different in different regions; one region exhibits an almost linear trend (as in Figure 8-4 and Figure 8-5) while in another region (around the region closer to solution), we find the presence of many points of inflection. This example is representative of the search space for a typical problem in the area of planar mechanisms. As shown in the results from Chapter 7, there are a few problems where the solutions are obtained easily (examples like benchmark problem #1) while in others (such as benchmark problem #3), there are not many solutions generated. This is shown in Table 8-1. The investigations on the nature of

the search space reveals that since the space is so different, it is not possible to guarantee solutions each time. In addition to the search space, the nature of the desired path also affects the algorithm output.

8.1.3 Desired Path

The desired Path is usually specified in terms of (x, y) coordinates. The number of points on the desired path affects the performance of the algorithm and the search space. If the number of points specified is too low, then the generated solution does not trace the path but just passes through those points. This can be seen in the solutions to challenge problem #2 and the second segment in challenge problem #3, where the number of points specified are too low for any solution to exactly follow the path. That is why, after initially synthesizing that segment in challenge problem #2 with four points, it was decided to increase the number of points that describe the second longhorn curve without loss of generality. This explains the reason for better results obtained during the multiple mechanism approach for the same curve. But, if too many points describe the desired path, the computational effort to compute the objective function increases in addition to algorithmic complexities in determining a near optimal solution. So finding the optimum number of points on the desired path is required for good performance of the algorithm. A potential method that can be used if a large number of points are specified (say 100) would involve trimming the desired path using the Ramer-Douglas-Peucker approximation technique [76] and then use the vector resulting from the optimization of the approximate path to be used while synthesizing the actual path without trimming. This concept of finding an approximate solution quickly and then refining the same may be computationally efficient.

8.1.4 Algorithm Selection

In the implementation presented, a hybrid approach involving Particle Swarm Optimization and Nelder-Mead simplex algorithm is adopted. This combination was

arrived after several tests. Figure 8-8 presents the improvement effected by the Nelder-Mead simplex algorithm in 100 iterations for the solution produced by Particle Swarm Optimization (after 750 iterations) for benchmark problem #1. But for the same problem, Figure 8-9 shows that the Nelder-Mead algorithm is also able to determine near-optimal solutions on two out of the three trials. In both these methods, there is a stochastic element involved. In the Particle Swarm Optimization, the assignment of positions and velocities of particles is random while in the Nelder-Mead algorithm the starting vector is randomly generated. So the varying nature of the performance is naturally expected. You may also notice from Figure 8-8 that if the solution from Particle Swarm Optimization is already at a minimum (Trial 2), then the Nelder-Mead algorithm does not produce any improvement.

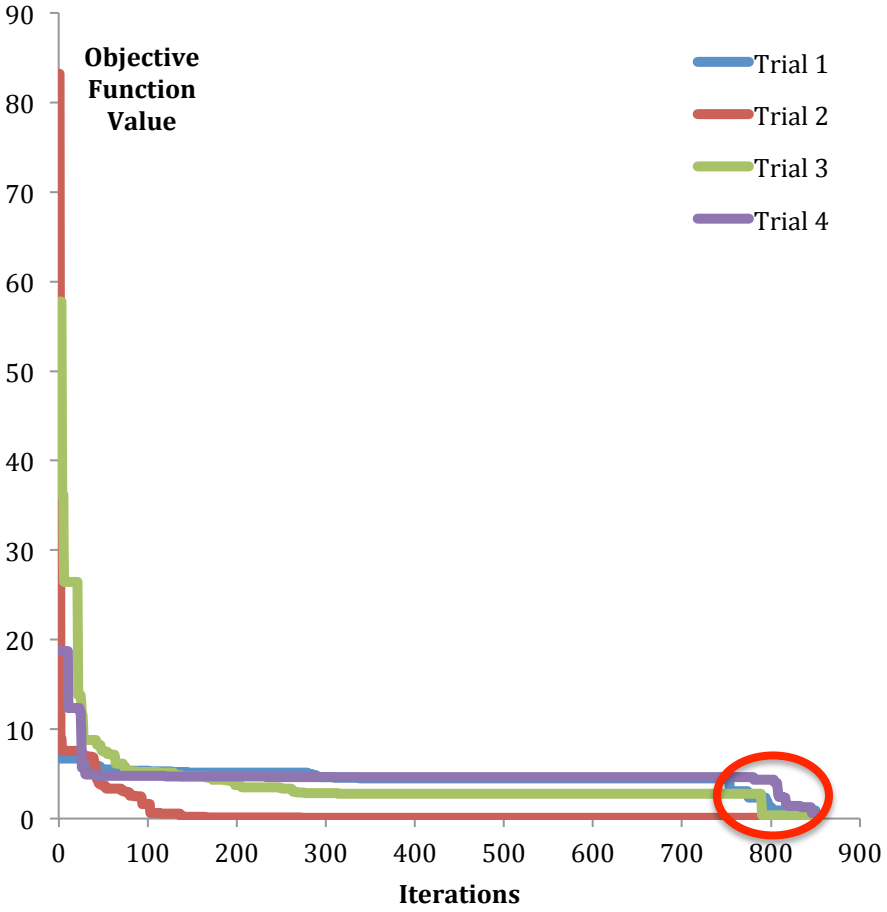


Figure 8-8 Performance of the hybrid algorithm on benchmark problem #1

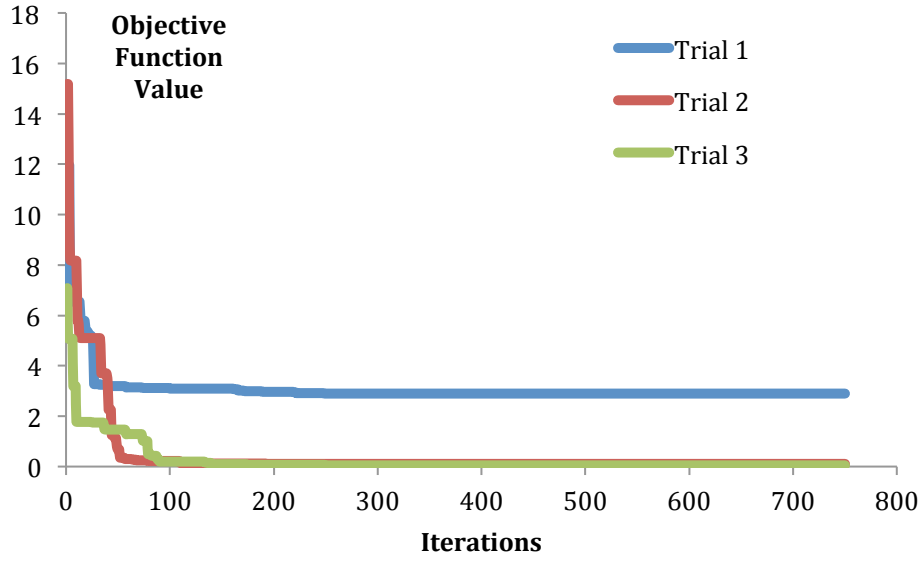


Figure 8-9 Performance of the Nelder-Mead simplex algorithm on benchmark problem #1

This simple trial proves that it is not possible to accurately predict the performance of any optimization method. Based on several experiments that were carried out on different problems, the hybrid algorithm generated more near optimal solutions (based on results in Chapter 7) and hence it was decided to adopt the same.

8.2 Discussion on the Results for Benchmark Problems

Our implementation is able to produce better results than the literature. At the same time, using our integrated search and optimization scheme, we are able to generate alternatives to the four-bar mechanism to solve the benchmark problems. The results in Chapter 7 show that our algorithm is able to synthesize mechanisms with revolute and prismatic joints. At the same time, the results also highlight the fact that the implemented method does not produce near-optimal solutions or solutions better than the literature on all mechanism topologies that are considered for synthesis. This corresponds to the discussion in the previous section that guaranteeing a near-optimal solution every time is

not practical. At the same time, since several candidates are evaluated at the same time, the percentage of feasible candidates is higher in this approach. Since information is scare about the percentage of feasible candidates for other methods in the literature, no comparison with literature is made in this regard

The reader will also notice from the results that there are certain cases where the objective function value predicted are very low but when the actual curve is plotted, there is a marked difference between the actual and the desired curve. For instance, let us consider the benchmark problem #7. Though we have been able to produce good results, there are instances, such as the case shown below in Figure 8-10, where the curve is not traced correctly but still a low objective function value was obtained.

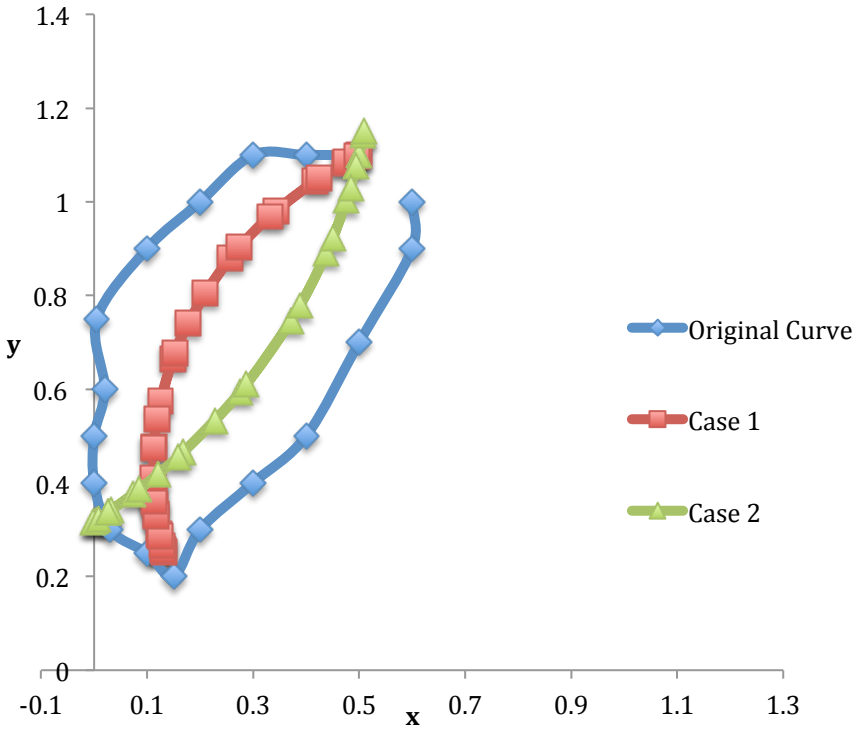


Figure 8-10: Different output curves but still resulting in a low objective function value

This indicates that the method is sensitive to scaling issues. Both the path as well as the resulting mechanism is in a dimensionally smaller unit scale compared to some of

the other problems. Such scale sensitive problems require modifications to the objective function calculation procedure such that these problems are handled better. One possible modification that can be done is to further reduce the convergence criteria for such scale sensitive problems. That way, partially synthesized mechanisms can be avoided from being shown as potential solutions. The other possible option is to adopt the process of exactly matching the points in the desired path, i.e., avoid using the Wilson-Theta approach. This might result in increased computational time. So there is a tradeoff between the input angle increments for analytical solutions versus computational time. Currently, the increments for the input crank are 10° coupled with the Wilson-Theta method for interpolating intermediate positions. This approach has been successful in solving most of the problems but there are instances as pointed out in Figure 8-8. Issues related to scale sensitivity have to been taken up further such that the applicability of this generic method is not affected if a user decides to employ our tool to create mechanisms at the milli- or micro-scales.

8.3 CHALLENGE PROBLEMS

Since this implementation has been able to generate good results for the benchmark problems, the generality of the implementation has also been tested using different challenge problems. The first challenge problem is a path-tracing problem. The result generated is encouraging considering the fact that our implementation has been tested on two different scales with a high number of points describing the desired path. A total of three results are presented and it can be seen that the path followed by all three mechanisms are very similar to each other despite having totally different configurations. Such information can be useful in understanding the limits of mechanisms and the kind of curves that can be generated by them. Studies have been conducted on four-bar mechanisms that predict a sixth-order curve, while the same cannot be said about high order mechanisms. This information can be also used to create a learning system that can understand which linkage combinations can produce a particular section of a curve. For

instance, in this challenge problem, there is a straight-line section followed by an angular section (as part of the conveyer system). Through a learning system, we could modify the linkage on the fly by either adding or removing links based on the kinds of motion being generated. Currently the system generates a topology and then uses an optimization algorithm to synthesize that topology's parameters. Instead, if the linkages are built on the fly based on a learning system, then that would lead this research area in a new direction.

The second challenge problem has only a few points that are part of each joint's desired path. This path problem is solved as a single-input multi-output problem. The results for the first case (where the multiple outputs are randomly assigned) show that precise tracing has not been achieved. This is because of the low number of points in the desired path that causes the synthesis program to generate a mechanism that traces a circular path that lies over the desired path. The second case (where two ternary links are used to represent the four joints that make up the rear leg of the coconut crab) could not be completed due to the computational time limits at our facility. Better results can be obtained by synthesizing one mechanism at a time rather than all the possible candidates at a particular level on the search tree.

The third challenge problem has also been solved using two methods – the first being single-input multi-output case and the second being the multiple mechanism approach. In both cases, the given desired curve has been sub-divided into four since we do not have any information currently or confidence that a single joint in a planar mechanism can trace the complete curve as it is. Hence we selected this approach. It can be seen that the single-input multi-output case does not generate good results. This is primarily due to the fact that only four-bar mechanisms are used. Higher order mechanisms are not generated at the time of writing this dissertation primarily due to computational time constraints. The second approach of using multiple planar mechanisms to solve is able to better trace different sections of the curve as shown in the results. The idea of using multiple mechanisms enables amalgamation of different paths

traced by each mechanism to produce the composite curve. In the results, individual curves were determined separately and manually selected to attain the composite curve. This could be easily automated but again could not be done due to computational time constraints. One aspect in using multiple mechanisms that can be explored is how to generate these multiple mechanisms without intersecting with other mechanisms as well as avoiding a grounded pivot on one of the other curves. This could be specified as additional constraints similar to the "input spacing" constraint. The other aspect is to determine how to create subsections of a complex curve such as the one in challenge problem #3. Currently, this was manually done using a simple first-order check of the inflection points in the curve. But a consistent methodology is required to expand this approach to other problems.

8.4 COMPUTATION TIME

One aspect in our research that has not been mentioned is the time of computation. While synthesizing a standalone four-bar mechanism, quoting the computational time (as in the literature) can provide an indication of the ability of a particular implementation. But in this dissertation, in addition to generating good quality results using four-bar mechanisms, we have been keen on exploring other planar mechanism designs that have rarely been carried out in the literature. In doing so, only an overall assessment of time is possible. For instance, Table 8-2 shows the average time of computation for three different benchmark problems at level #7 in the search tree. This indicates that the optimization algorithm spends an average of 25 minutes in trying to parametrically synthesize a mechanism topology. The table also shows that the time of computation is a function of the desired path as well as the number of valid mechanisms being synthesized. For instance, since more solutions are being synthesized for benchmark problem #1, the time for each candidate is much higher than the other two problems.

Table 8-2 Time of computation for three benchmark problems

	Benchmark Problem 1	Benchmark Problem 2	Benchmark Problem 3
No of Solutions	24	24	24
Near Optimal Solutions / Better than Existing Results	24	0	1
Time (in min)	741	635	465
Time (in min) / solution	30.88	26.46	19.38
Average Time (min)	25.57		

The above table gives an indication of the total time required to synthesize different mechanisms using this technique. Moreover, based on several tests, we concluded that the Particle Swarm Optimization could be allowed to run a maximum of 750 iterations for every potential candidate followed by 100 iterations on the Nelder-Mead algorithm. Additionally, multi-output problems require a much higher time frame to arrive at a solution.

This necessitates incorporating alternative strategies and better memory management and programming to ensure quicker results. One such improvement is to compute the positions for large angle increments of the input link by taking advantage of the Wilson-Theta method. In this dissertation, we have used 10° increments of the input link. This way, the objective functions calculations are much faster than when smaller time increments are used for input rotation. The Wilson-Theta approach has been thoroughly tested and we can confirm that there is only a minimal loss of information in using large increments for input angle. The other is to reduce the number of duplicate candidates that are generated by the grammar rules. Finally, it is important to ensure code parallelization to take advantage of the multi-core CPUs that are currently available.

8.5 CONCLUSION

This chapter highlights some of the limitations as well as the complexities in the search that affect the results being generated for this class of problems. The motivations behind certain solution strategies are also highlighted in this chapter.

Chapter 9: Summary and Future Work

A methodology for automating the design of planar mechanisms is presented in this dissertation. Three major aspects of this research are presented in detail. The first aspect is the graph-grammar based representation scheme used to represent different elements of planar mechanisms. Using this representation scheme, grammar rules are formulated that are used to generate different mechanisms in an exhaustive tree search process The representation scheme as well as the rules formulated is generic due to the small set of rules required to generate all revolute and prismatic joints. Due to the small set of rules, there are duplicate candidates generated. Though this increases the computational resources required while evaluating candidates at every level in the tree, such rules also provide an indication to the designer or a general user about different ways of building a particular mechanism. The second aspect presented in this study is the kinematic analysis required to automatically evaluate a planar 1-degree of freedom mechanism. Graphical methods to evaluate position and velocity and analytical acceleration equation solving method have been formulated in a generic way that can evaluate any mechanism on the fly during the search process. In addition, since generic implementations of advanced methods to solve indeterminate mechanisms are not publicly available, an optimization-based method has been developed to solve the position kinematics of mechanisms with revolute joints. The kinematic analysis developed is also publicly available as an open-source code.

The third and final aspect in this research is the optimization of the generated mechanisms to solve user-defined path problems. Here, after evaluating several different algorithms, a hybrid implementation of Particle-Swarm Optimization and Nelder-Mead optimization has been developed to automate the shape of mechanisms. This hybrid implementation is able to produce better results on most of the benchmark problems without requiring any change in the core algorithm used. The use of the design generator has helped in producing mechanisms (topologies) other than a four-bar mechanism to

solve the different benchmark problems. The hybrid method has also been tested on three challenge problems. Due to the nature of the challenge problems, three different scenarios have been tested in this research namely single input single output, single input multi output and multiple mechanism approach. A discussion on the search space and the constraints for this class of problems are presented where different aspects that influence the results are investigated to understand the difficulties in consistently yielding solutions.

9.1 CONTRIBUTIONS

The overall design automation scheme has been successfully demonstrated through this dissertation. The following are some of the major contributions of this research to the design and mechanism community.

1) Developed a generic graph-grammar based representation and rules system for planar mechanisms consisting of different joints
2) Developed a generic kinematic analysis tool based on graphical and analytical methods for determinate 1-degree of freedom planar mechanisms
3) Developed an optimization based approach to accurately determine the position kinematics of planar indeterminate mechanisms consisting of revolute joints
4) Implemented a modification for Nelder-Mead algorithm to improve its performance for constrained problem class such as planar mechanism synthesis
5) Developed a hybrid implementation of Particle-Swarm Optimization and Nelder-Mead algorithm that is able to produce better results on most of the benchmark problems using four-bar mechanisms
6) Synthesized higher-order mechanisms for benchmark problems by combining grammar rules to generate the mechanisms and evaluating them using the developed algorithm
7) Provided insights into the search space that explains the lack of repeatability and the lower probabilities of algorithms finding the best or near-optimal solutions

8) Demonstrated that a generic tool for automated conceptual design of planar mechanisms can be developed

9.2 FUTURE WORK

In terms of advancing this work, the following are the activities that are being planned.

9.2.1 Representation

Grammar rules for R-P joints are planned so that planar mechanisms consisting of R, P and R-P joints can be created in addition to the current capability of generating mechanisms with just R and P joints. The representation will also be expanded to integrate machine elements like gears, which also will help advance the multiple mechanism approach where by mechanisms generated using our technique can be combined with appropriate gearing automatically to create a more complete device

9.2.2 Kinematic Analysis

The current optimization based method for indeterminate mechanisms will be improved to solve such mechanisms with sliding members. Through this implementation, we can ease the restriction in rules that prismatic joints can only be connected to those mechanisms consisting of an input four-bar loop. Incorporating computations for geared mechanisms as well as robust implementations for R-P joints will be part of future work in the area of kinematic analysis.

9.2.3 Search and Optimization

Improved techniques to detect duplicate mechanisms generated during the search techniques will be incorporated into our system since as shown in this dissertation that the first order isomorphism detection is unable to eliminate duplicate candidates. The other aspect in this research is to test our implementation on other types of problems such as links following a particular motion, mechanisms where there are a combination

requirement i.e., follow a particular path for certain orientations of the input crank and then a particular link will follow a particular motion. Research will also be conducted on the changes that affect the curves generated as a result of adding a link or removing one. This information would be useful to create rules that are adaptive to the optimization process compared to the process demonstrated here where a topology is completely generated before the algorithms synthesize different parameters of that topology.

Appendix A: Additional Finite Position Problems

Figure A-1 shows the model of a double-butterfly linkage, whose pivot positions are listed in Table A-1. OA is the input link of this mechanism. The results of the algorithm are available in Figure A-2. It may be noted that the maximum permissible travel of this input link is 75° beyond which the mechanism encounters a toggle position and the mechanism takes the topology of another kinematically equivalent branch.

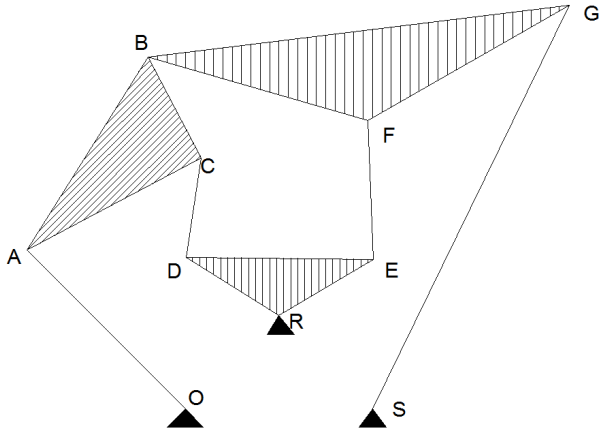


Figure A-1: Double butterfly linkage [36]

Table A-1: Pivot positions of the double butterfly linkage (Figure A-1) for the finite position problem

Pivot	Coordinate
O	(-5.0000, 0.0000) (input CW)
R	(-2.5000, 2.5000)
S	(0.0000, 0.0000)
A	(-9.2400, 4.2400)
B	(-5.9000,9.3000)

Table A-1 continued.

Pivot	Coordinate
C	(-4.5000,6.5000)
D	(-5.0000,4000)
E	(0.0000,4.000)
F	(-0.1340,8.7170)
G	(5.2680,10,7820)

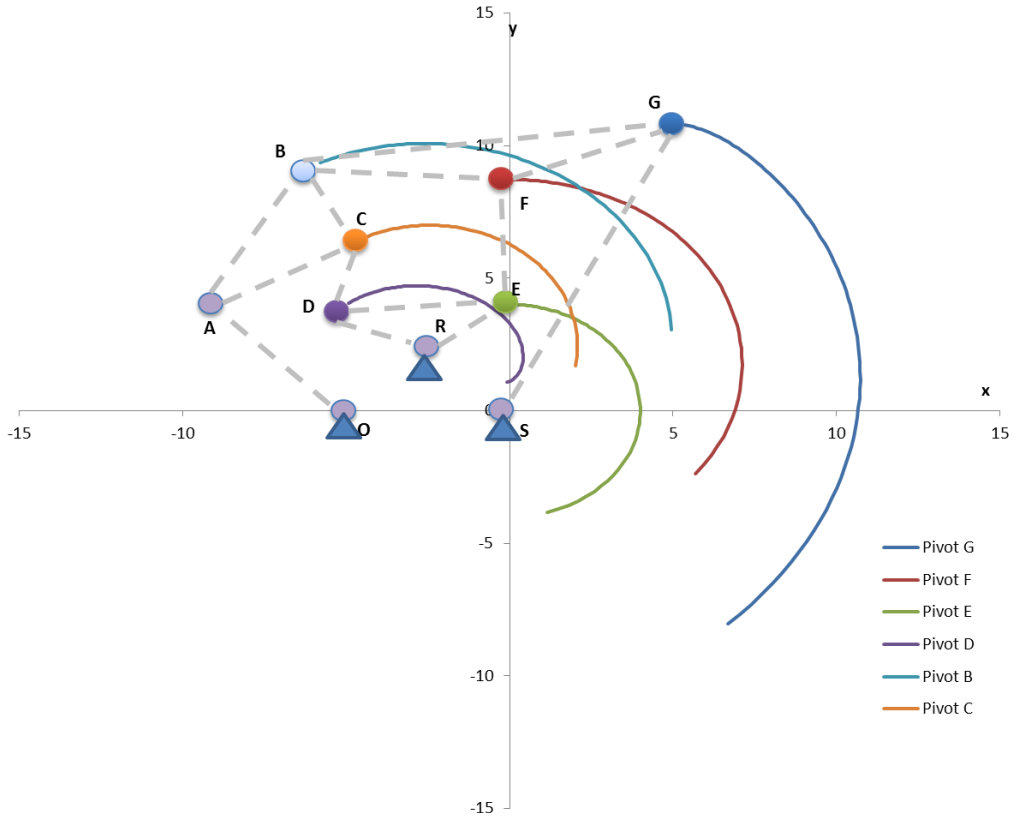


Figure A-2: Path traversed by the pivots (B, C, D, E, F, G) of the double butterfly linkage in Figure A-1

Figure A-3 shows the model of another double-butterfly linkage whose pivot parameters are listed in Table A-2. The ground pivots for this mechanism are A, R and S.

This example has been tested with different input links such as RD, RE and SG. The results of the algorithm with RE as input are shown in Figure A-4. The maximum angle traversed in this configuration is approximately 230°. Since the results with RD as input link is similar that with RE as input, they are not listed here. The results with SG as input are listed in Table A-3 in angle increments of 0.1°. As the maximum angle traversed in this configuration is only 2.9°, no graph is plotted for this case.

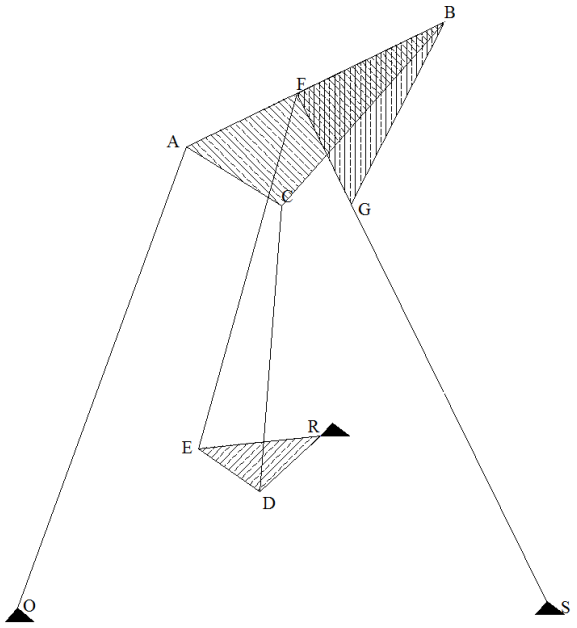


Figure A-3: Double-butterfly linkage – example II [32]

Table A-2: Pivot positions of the double butterfly linkage (Figure A-3) for the finite position problem

Pivot	Coordinate
O	(0.0000, 0.0000)
S	(13.0000, 0.0000)
A	(4.1276, 11.2684)
B	(10.4289, 14.2544)
C	(6.4193, 9.7727)
D	(5.9286, 2.8454)
E	(4.4152, 3.8987)

Table A-2 continued.

Pivot	Coordinate
F	(6.8309, 12.5685)
G	(8.1434, 9.8698)
R	(7.4000, 4.2000)
Inputs: R; S (CW)	

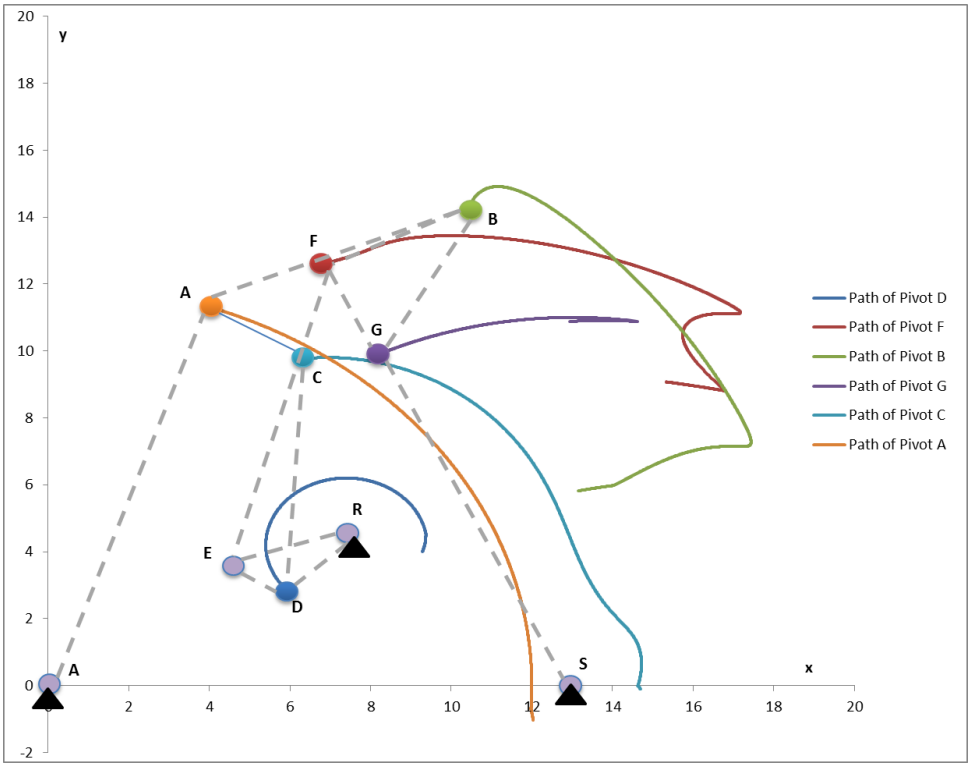


Figure A-4: Path traversed by pivots (A, B, C, D, F, G) with RE as input

Table A-3: Positions of pivots at angle increments of 0.1 ° with SG as input

Angle	Pivot E		Pivot F		Pivot B		Pivot D		Pivot C		Pivot A	
	x	y	x	y	x	y	x	y	x	y	x	y
0	4.42	3.90	6.83	12.57	10.43	14.25	5.93	2.85	6.42	9.77	4.13	11.27
0.1	4.42	3.90	6.84	12.57	10.43	14.27	5.93	2.85	6.44	9.78	4.14	11.26
0.2	4.42	3.91	6.85	12.58	10.43	14.29	5.93	2.85	6.45	9.78	4.15	11.26
0.3	4.42	3.91	6.85	12.58	10.43	14.30	5.93	2.86	6.47	9.78	4.16	11.26
0.4	4.42	3.92	6.86	12.58	10.44	14.32	5.94	2.86	6.49	9.79	4.17	11.25
0.5	4.43	3.93	6.87	12.59	10.44	14.34	5.94	2.87	6.50	9.79	4.19	11.25
0.6	4.43	3.93	6.88	12.59	10.44	14.35	5.94	2.87	6.52	9.79	4.20	11.24
0.7	4.43	3.94	6.89	12.59	10.44	14.37	5.94	2.88	6.54	9.79	4.21	11.24
0.8	4.43	3.94	6.89	12.60	10.44	14.38	5.94	2.88	6.55	9.80	4.22	11.23
0.9	4.44	3.95	6.90	12.60	10.45	14.40	5.94	2.88	6.57	9.80	4.24	11.23
1	4.44	3.95	6.91	12.60	10.45	14.41	5.95	2.89	6.59	9.80	4.25	11.22
1.1	4.44	3.96	6.92	12.61	10.45	14.43	5.95	2.89	6.61	9.81	4.26	11.22
1.2	4.44	3.96	6.93	12.61	10.45	14.44	5.95	2.90	6.62	9.81	4.27	11.21
1.3	4.45	3.97	6.94	12.61	10.46	14.46	5.95	2.90	6.64	9.81	4.29	11.21
1.4	4.45	3.97	6.95	12.62	10.46	14.47	5.95	2.91	6.66	9.81	4.30	11.20
1.5	4.45	3.98	6.95	12.62	10.46	14.49	5.95	2.91	6.68	9.82	4.31	11.20
1.6	4.45	3.98	6.96	12.62	10.46	14.50	5.96	2.91	6.69	9.82	4.33	11.19
1.7	4.46	3.99	6.97	12.63	10.47	14.52	5.96	2.92	6.71	9.82	4.34	11.19
1.8	4.46	3.99	6.98	12.63	10.47	14.53	5.96	2.92	6.73	9.82	4.35	11.18
1.9	4.46	4.00	6.99	12.63	10.47	14.55	5.96	2.93	6.75	9.83	4.37	11.18
2	4.46	4.00	7.00	12.64	10.48	14.56	5.96	2.93	6.77	9.83	4.38	11.17
2.1	4.47	4.01	7.01	12.64	10.48	14.58	5.97	2.93	6.78	9.83	4.40	11.17
2.2	4.47	4.01	7.02	12.64	10.48	14.59	5.97	2.94	6.80	9.83	4.41	11.16
2.3	4.47	4.02	7.03	12.65	10.48	14.60	5.97	2.94	6.82	9.84	4.42	11.16
2.4	4.47	4.02	7.04	12.65	10.49	14.62	5.97	2.95	6.84	9.84	4.44	11.15
2.5	4.47	4.03	7.05	12.65	10.49	14.63	5.97	2.95	6.86	9.84	4.45	11.14
2.6	4.48	4.03	7.06	12.66	10.49	14.65	5.97	2.96	6.88	9.84	4.47	11.14
2.7	4.48	4.04	7.07	12.66	10.50	14.66	5.98	2.96	6.90	9.84	4.48	11.13
2.8	4.48	4.04	7.07	12.66	10.50	14.67	5.98	2.96	6.92	9.84	4.50	11.13
2.9	4.48	4.05	7.08	12.66	10.50	14.69	5.98	2.97	6.94	9.85	4.51	11.12

Figure A-5 shows the model of a ten-bar mechanism, whose pivot positions are displayed in Table A-4 where O, Q, R and S are the ground pivots and SI is the input link. The results of the algorithm are displayed in Table A-5. The maximum angle traversed by the input link in this configuration is 9°. This is another example demonstrating the capability of the method in solving different types of planar mechanisms.

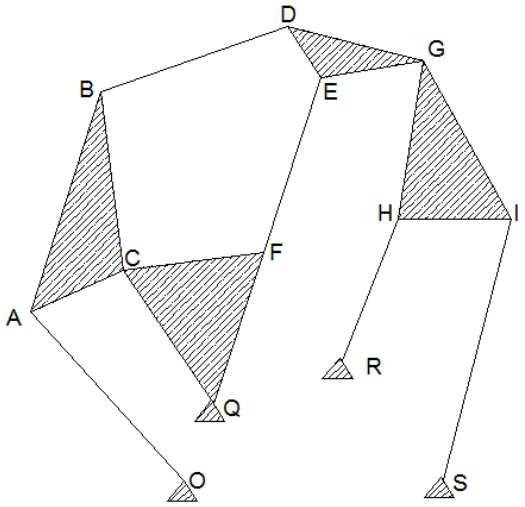


Figure A-5: Ten-bar mechanism [77]

Table A-4 Pivot positions of the ten-bar mechanism for the finite position problem

Pivot	Coordinate
O	(-1.2500, -6.4000)
S	(12.3020, -3.7960)
A	(-1.1860, -4.6000)
B	(0.7330, 1.3560)
C	(2.6280, -1.7200)
D	(0.7010, -4.4430)
E	(3.8870, -3.8080)
F	(8.5290, -1.6950)
G	(5.7100, -0.8030)
H	(10.1200, -2.5930)
I	(12.2380, 1.1040)
R	(8.9500, -4.0900)
Q	(6.1750, -4.7290)
Input Link: SI (CW)	

Table A-5: Positions of pivots (A, B, C, D, E, F, G, and H) at angle increments of 1°

Angle	Pivot H		Pivot G		Pivot D		Pivot E		Pivot F		Pivot C		Pivot B		Pivot A	
	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y
0	10.12	-2.59	5.71	-0.80	0.70	-4.44	3.89	-3.81	8.53	-1.69	2.63	-1.72	0.73	1.36	-1.19	-4.60
1	9.99	-2.48	5.70	-0.43	0.92	-4.37	4.06	-3.54	8.94	-2.07	3.10	-1.24	0.66	1.42	-0.95	-4.62
2	10.03	-2.48	5.77	-0.35	1.06	-4.37	4.18	-3.49	9.14	-2.29	3.38	-1.01	0.70	1.41	-0.71	-4.68
3	10.06	-2.47	5.83	-0.27	1.22	-4.40	4.32	-3.45	9.35	-2.56	3.72	-0.78	0.83	1.39	-0.78	-4.66
4	10.09	-2.46	5.90	-0.20	1.43	-4.48	4.50	-3.42	9.58	-2.95	4.21	-0.52	1.09	1.31	-0.70	-4.69
5	10.13	-2.47	5.98	-0.15	1.81	-4.73	4.79	-3.46	9.88	-3.73	5.17	-0.19	1.78	1.07	-0.57	-4.73
6	10.21	-2.49	6.05	-0.18	1.83	-4.72	4.81	-3.46	9.90	-3.76	5.20	-0.18	1.81	1.07	-0.57	-4.73
7	10.27	-2.51	6.13	-0.19	1.86	-4.72	4.83	-3.45	9.91	-3.78	5.23	-0.18	1.83	1.07	-0.56	-4.73
8	10.32	-2.53	6.20	-0.19	1.91	-4.71	4.86	-3.42	9.92	-3.81	5.27	-0.17	1.87	1.07	-0.55	-4.72
9	10.37	-2.54	6.27	-0.18	1.96	-4.71	4.88	-3.39	9.93	-3.85	5.31	-0.16	1.90	1.06	-0.54	-4.72

Appendix B: Additional Initial Position Problems

The results of the initial position problem for a double-butterfly linkage and a ten-bar linkage are presented in this section. The length parameters for the double-butterfly linkage are taken from the mechanism shown in Figure A-1 and the two solutions for this linkage are shown in Figure B-1 and Figure B-2 along with their pivot coordinates in Table B-1 and Table B-2 respectively. Similarly Figure B-3 and Figure B-4 are the two solutions for a ten-bar linkage (length parameters are taken from mechanism shown in Figure A-5) whose pivot coordinates are displayed in Table B-3 and Table B-4 respectively.

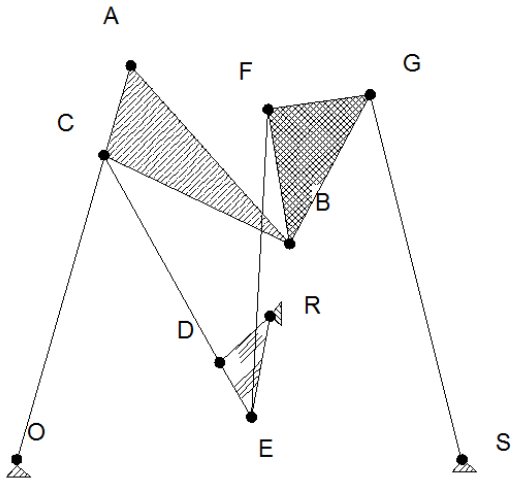


Figure B-1: Initial position problem solution #1 for a double butterfly linkage

Table B-1: Pivot parameters of the double butterfly linkage shown in Figure B-1

Pivot	Coordinate
O	(0.0000, 0.0000)
S	(13.0000, 0.000)
A	(11.5948, 3.0345)
B	(7.2705, -2.4634)
C	(12.4648, 0.4835)

Table B-1 continued.

Pivot	Coordinate
D	(5.9286, 2.8454)
E	(4.4192, 3.9521)
F	(4.1993, -5.0235)
G	(2.3147, -2.6617)
R	(7.4000, 4.2000)

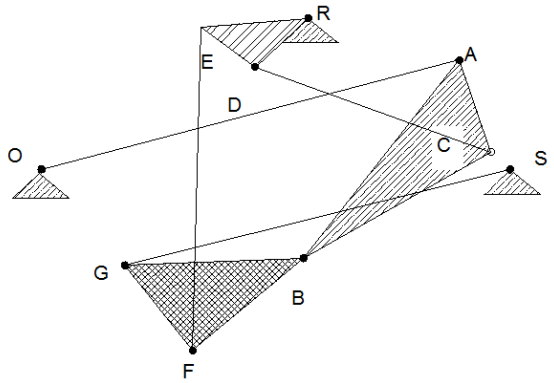


Figure B-2: Initial position problem solution #2 for a double butterfly linkage

Table B-2: Pivot parameters of the double butterfly linkage shown in Figure B-2

Pivot	Coordinate
O	(0.0000, 0.0000)
S	(13.0000, 0.0000)
A	(3.3367, 11.5273)
B	(7.9666, 6.3132)
C	(2.5420, 8.9086)
D	(5.9286, 2.8454)
E	(6.8546, 1.2503)
F	(7.3409, 10.2372)
G	(10.3111, 10.6664)
R	(7.4000, 4.2000)

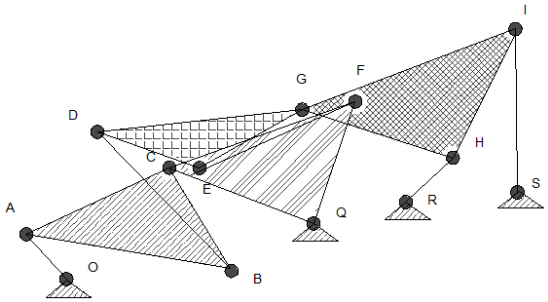


Figure B-3: Initial position problem solution #1 for a ten-bar mechanism

Table B-3: Pivot parameters of the ten-bar mechanism shown in Figure B-3

Pivot	Coordinate
O	(-1.2500, -6.4000)
S	(12.3020, -3.7960)
A	(-2.4556, -5.0559)
B	(3.7091, -6.1585)
C	(1.8439, -3.0581)
D	(-0.3168, -1.9833)
E	(2.7503, -3.0671)
F	(7.4314, -1.0742)
G	(5.8366, -1.3178)
H	(10.3632, -2.7609)
I	(12.2380, 1.1040)
R	(8.9500, -4.0900)
Q	(6.1750, -4.7290)

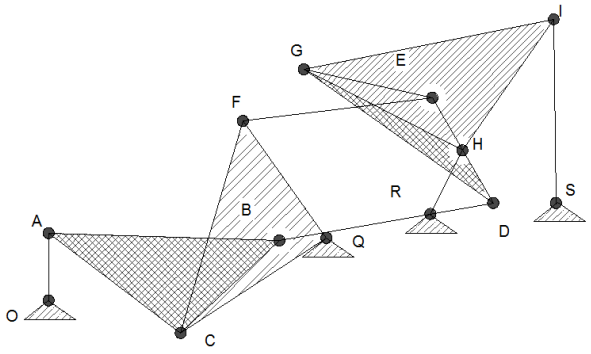


Figure B-4: Initial position problem solution #2 for a ten-bar mechanism

Table B-4: Pivot parameters of the ten-bar mechanism shown in Figure B-4

Pivot	Coordinate
O	(-1.2500, -6.4000)
S	(12.3020, -3.7960)
A	(-1.3395, -4.6011)
B	(4.9148, -4.7995)
C	(2.2797, -7.27128)
D	(10.6262, -3.79518)
E	(8.9997, -0.9829)
F	(3.9378, -1.6078)
G	(5.5679, -0.2241)
H	(9.8043, -2.3931)
I	(12.2380, 1.1040)
R	(8.9500, -4.0900)
Q	(6.1750, -4.7290)

References

[1] Sclater, N., and Chironis, N. P., 2001, Mechanisms and mechanical devices sourcebook, McGraw-Hill.

[2] Waldron, K. J., and Kinzel, G. L., 2004, Kinematics, dynamics, and design of machinery, Wiley.

[3] Norton, R. L., 2004, Design of machinery, McGraw-Hill Professional.

[4] Erdman, A. G., and Sandor, G. N., 1997, Mechanism design, Prentice-Hall.

[5] "3D CAD Design Software SolidWorks" [Online]. Available: http://www.solidworks.com/. [Accessed: 22-Dec-2012].

[6] "Working Model 2D - Home," http://www.design-SimulationcomWM2Dindexphp [Online]. Available: http://www.design-simulation.com/WM2D/index.php. [Accessed: 25-Aug-2009].

[7] "Adams - Overview," http://www.mscsoftware.com/products/adams.cfm [Online]. Available: http://www.mscsoftware.com/products/adams.cfm. [Accessed: 25-Aug-2009].

[8] "ARTAS - Engineering Software" [Online]. Available: http://www.artas.nl/. [Accessed: 25-Aug-2009].

[9] "WATT Mechanism Suite," http://www.heron-Technol. [Online]. Available: http://www.heron-technologies.com/watt/. [Accessed: 25-Aug-2009].

[10] Freudenstein, F., and Maki, E. R., 1979, "The creation of mechanisms according to kinematic structure and function," Environ. Plan. B Plan. Des., **6**(4), pp. 375–391.

[11] Tsai, L.-W., 2001, Mechanism design, CRC Press.

[12] Pucheta, M. A., and Cardona, A., "Type synthesis of planar linkage mechanisms with rotoidal and prismatic joints," Mecánica Comput., **26**, pp. 2703–2730.

[13] Lu, Y., and Leinonen, T., 2005, "Type synthesis of unified planar-spatial mechanisms by systematic linkage and topology matrix-graph technique," Mech. Mach. Theory, **40**(10), pp. 1145–1163.

[14] Ding, H., Huang, P., Zi, B., and Kecskeméthy, A., 2012, "Automatic synthesis of kinematic structures of mechanisms and robots especially for those with complex structures," Appl. Math. Model., **36**(12), pp. 6122–6131.

[15] Mruthyunjaya, T. S., 2003, "Kinematic structure of mechanisms revisited," Mech. Mach. Theory, **38**(4), pp. 279–320.

[16] Sohn, W. J., and Freudenstein, F., 1986, "An Application of Dual Graphs to the Automatic Generation of the Kinematic Structures of Mechanisms," J. Mech. Transm. Autom. Des., **108**(3), pp. 392–398.

[17] Sedlaczek, K., Gaugele, T., and Eberhard, P., 2005, "Topology optimized synthesis of planar kinematic rigid body mechanisms," Multibody Dyn.

[18] Mayourian, M., and Freudenstein, F., 1984, "The Development of an Atlas of the Kinematic Structures of Mechanisms," J. Mech. Transm. Autom. Des., **106**(4), pp. 458–461.

[19] Butcher, E. A., and Hartman, C., 2005, "Efficient enumeration and hierarchical classification of planar simple-jointed kinematic chains: Application to 12- and 14-bar single degree-of-freedom chains," Mech. Mach. Theory, **40**(9), pp. 1030–1050.
[20] Noriega, A., Cadenas, M., and Fernández, R., 2013, "Position Problem in Assur's Groups with Revolute Pairs," New Trends in Mechanism and Machine Science, F. Viadero, and M. Ceccarelli, eds., Springer Netherlands, pp. 141–148.
[21] Shai, O., 2010, "Topological Synthesis of All 2D Mechanisms Through Assur Graphs," ASME Conf. Proc., **2010**(44106), pp. 1727–1738.
[22] Rao, A. ., 1997, "Hamming number technique—I. Further applications," Mech. Mach. Theory, **32**(4), pp. 477–488.
[23] Rao, A. ., 1997, "Hamming number technique—II. Generation of planar kinematic chains," Mech. Mach. Theory, **32**(4), pp. 489–499.
[24] Cabrera, J. A., Simon, A., and Prado, M., 2002, "Optimal synthesis of mechanisms with genetic algorithms," Mech. Mach. Theory, **37**(10), pp. 1165–1177.
[25] Cossalter, V., Doria, A., Pasini, M., and Scattolo, C., 1992, "A simple numerical approach for optimum synthesis of a class of planar mechanisms," Mech. Mach. Theory, **27**(3), pp. 357–366.
[26] Liu, Y., and McPhee, J., 2007, "Automated Kinematic Synthesis of Planar Mechanisms with Revolute Joints," Mech. Based Des. Struct. Mach., **35**(4), pp. 405–445.
[27] Kreyszig, T., 1995, Advanced Engineering Mathematics: With Mathematics Manual, John Wiley & Sons Canada, Limited.
[28] Klein, A. W., 1917, Kinematics of machinery: a text-book on mechanisms and their properties, with many practical applications for engineers and for students in technical schools, McGraw-Hill.
[29] Waldron, K. J., and Sreenivasan, S. V., 1996, "A Study of the Solvability of the Position Problem for Multi-Circuit Mechanisms by Way of Example of the Double Butterfly Linkage," J. Mech. Des., **118**(3), p. 390.
[30] Sommese, A. J., Wampler, C. W., and (II.), C. W. W., 2005, The numerical solution of systems of polynomials arising in engineering and science, World Scientific.
[31] Morgan, A., 2009, Solving Polynominal Systems Using Continuation for Engineering and Scientific Problems, SIAM.
[32] Wampler, C. W., 2001, "Solving the Kinematics of Planar Mechanisms by Dixon Determinant and a Complex-Plane Formulation," J. Mech. Des., **123**(3), p. 382.
[33] Nielsen, J., and Roth, B., 1999, "Solving the Input/Output Problem for Planar Mechanisms," J. Mech. Des., **121**(2), pp. 206–211.
[34] Dixon, A. L., 1909, "The Eliminant of Three Quantics in two Independent Variables:(Second Paper.)," Proc. Lond. Math. Soc., **2**(1), p. 473.
[35] Hernández, A., and Petuya, V., 2004, "Position analysis of planar mechanisms with R-pairs using a geometrical-iterative method," Mech. Mach. Theory, **39**(2), pp. 133–152.

[36] Foster, D. E., and Pennock, G. R., 2003, "A Graphical Method to Find the Secondary Instantaneous Centers of Zero Velocity for the Double Butterfly Linkage," J. Mech. Des., **125**(2), p. 268.
[37] Gea, H. C., and Kwon, J., 2005, "Topological Synthesis for Linkage Mechanism Design Using the Minimum Potential Energy Principle," Volume 2: 31st Design Automation Conference, Parts A and B, Long Beach, California, USA, pp. 931–937.
[38] Chen, W.-J., Chang, C.-J., and Gea, H. C., 2008, "Topology and Dimensional Synthesis of Linkage Mechanism Based on the Constrained Superposition Method," Volume 1: 34th Design Automation Conference, Parts A and B, Brooklyn, New York, USA, pp. 789–797.
[39] Porta, J. M., Ros, L., Creemers, T., and Thomas, F., 2007, "Box Approximations of Planar Linkage Configuration Spaces," J. Mech. Des., **129**(4), pp. 397–405.
[40] Alizade, R. I., Novruzbekov, I. G., and Sandor, G. N., 1975, "Optimization of four-bar function generating mechanisms using penalty functions with inequality and equality constraints," Mech. Mach. Theory, **10**(4), pp. 327–336.
[41] Sancibrian, R., García, P., Viadero, F., and Fernández, A., 2006, "A general procedure based on exact gradient determination in dimensional synthesis of planar mechanisms," Mech. Mach. Theory, **41**(2), pp. 212–229.
[42] Cabrera, J. A., Nadal, F., Mu\ noz, J. P., and Simon, A., 2007, "Multiobjective constrained optimal synthesis of planar mechanisms using a new evolutionary algorithm," Mech. Mach. Theory, **42**(7), pp. 791–806.
[43] Acharyya, S. K., and Mandal, M., 2009, "Performance of EAs for four-bar linkage synthesis," Mech. Mach. Theory, **44**(9), pp. 1784–1794.
[44] Liu, Y., and McPhee, J., 2005, "Automated Type Synthesis of Planar Mechanisms Using Numeric Optimization With Genetic Algorithms," J. Mech. Des., **127**(5), pp. 910–916.
[45] Roston, G. P., and Sturges, R. H., "Genetic algorithm synthesis of four-bar mechanisms," Ann Arbor, **1001**, p. 48105.
[46] Nariman-Zadeh, N., Felezi, M., Jamali, A., and Ganji, M., 2009, "Pareto optimal synthesis of four-bar mechanisms for path generation," Mech. Mach. Theory, **44**(1), pp. 180–191.
[47] Marcelin, J. L., 2004, "A metamodel using neural networks and genetic algorithms for an integrated optimal design of mechanisms," Int. J. Adv. Manuf. Technol., **24**(9), pp. 708–714.
[48] Smaili, A. A., Diab, N. A., and Atallah, N. A., 2005, "Optimum Synthesis of Mechanisms Using Tabu-Gradient Search Algorithm," J. Mech. Des., **127**(5), p. 917.
[49] Sancibrian, R., Viadero, F., García, P., and Fernández, A., 2004, "Gradient-based optimization of path synthesis problems in planar mechanisms," Mech. Mach. Theory, **39**(8), pp. 839–856.

[50] “GraphSynth,” CodePlex [Online]. Available: https://graphsynth.codeplex.com/Wikipage?ProjectName=graphsynth. [Accessed: 13-Apr-2014].
[51] Campbell, M., “A Graph Grammar Methodology for Generative Systems” [Online]. Available: http://repositories.lib.utexas.edu/handle/2152/6258. [Accessed: 30-Oct-2009].
[52] Cabrera, J. A., Simon, A., and Prado, M., 2002, “Optimal synthesis of mechanisms with genetic algorithms,” Mech. Mach. Theory, **37**(10), pp. 1165–1177.
[53] Cabrera, J. A., Ortiz, A., Nadal, F., and Castillo, J. J., 2011, “An evolutionary algorithm for path synthesis of mechanisms,” Mech. Mach. Theory, **46**(2), pp. 127–141.
[54] Hongying, Y., Dewei, T., and Zhixing, W., 2007, “Study on a new computer path synthesis method of a four-bar linkage,” Mech. Mach. Theory, **42**(4), pp. 383–392.
[55] Matekar, S. B., and Gogate, G. R., 2012, “Optimum synthesis of path generating four-bar mechanisms using differential evolution and a modified error function,” Mech. Mach. Theory, **52**, pp. 158–179.
[56] Acharyya, S. K., and Mandal, M., 2009, “Performance of EAs for four-bar linkage synthesis,” Mech. Mach. Theory, **44**(9), pp. 1784–1794.
[57] Kunjur, A., and Krishnamurty, S., 1997, “Genetic algorithms in mechanism synthesis,” J. Appl. Mech. Robot., **4**(2), pp. 18–24.
[58] 2010, Giant Robber crab on Christmas Island [Online]. Available: http://www.youtube.com/watch?v=yqPBVBskD-M&feature=youtube_gdata_player/.
[59] Radhakrishnan, P., and Campbell, M., I., 2010, “A graph grammar scheme for generating and evaluating planar mechanisms,” Des. Comput. Cogn. DCC’10, pp. 663–679.
[60] Diab, N., and Smaili, A., 2008, “Optimum exact/approximate point synthesis of planar mechanisms,” Mech. Mach. Theory, **43**(12), pp. 1610–1624.
[61] Hernández, A., Petuya, V., and Amezua, E., 2001, “A method for the solution of the forward position problems of planar mechanisms with prismatic and revolute joints,” Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci., **216**(4), pp. 395–407.
[62] “Object-Oriented Optimization Toolbox (OOOT)” [Online]. Available: http://ooot.codeplex.com/. [Accessed: 05-Sep-2012].
[63] Goldberg, D. E., and others, 1989, Genetic algorithms in search, optimization, and machine learning, Addison-wesley Reading Menlo Park.
[64] Storn, R., and Price, K., 1997, “Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,” J. Glob. Optim., **11**(4), pp. 341–359.
[65] Kennedy, J., Eberhart, R., and others, 1995, “Particle swarm optimization,” Proceedings of IEEE international conference on neural networks, Perth, Australia, pp. 1942–1948.

[66] Dukkipati, R. V., 2009, MATLAB for Mechanical Engineers, New Age Science, Limited.
[67] Kirkpatrick, S., 1984, "Optimization by simulated annealing: Quantitative studies," J. Stat. Phys., **34**(5-6), pp. 975–986.
[68] Gent, I. P., and Walsh, T., 1993, "Towards an understanding of hill-climbing procedures for SAT," AAAI, Citeseer, pp. 28–33.
[69] Nelder, J. A., and Mead, R., 1965, "A simplex method for function minimization," Comput. J., **7**(4), pp. 308–313.
[70] Blackwell, T., and Branke, J., 2004, "Multi-swarm optimization in dynamic environments," Applications of Evolutionary Computing, Springer, pp. 489–500.
[71] Torczon, V., 1997, "On the Convergence of Pattern Search Algorithms," SIAM J. Optim., **7**(1), pp. 1–25.
[72] Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E., 1998, "Convergence Properties of the Nelder--Mead Simplex Method in Low Dimensions," SIAM J. Optim., **9**(1), pp. 112–147.
[73] Gao, F., and Han, L., 2012, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters," Comput. Optim. Appl., **51**(1), pp. 259–277.
[74] McKay, M. D., Beckman, R. J., and Conover, W. J., 1979, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," Technometrics, **21**(2), pp. 239–245.
[75] Venkataraman, P., 2009, Applied Optimization with MATLAB Programming, John Wiley & Sons.
[76] Hershberger, J. E., and Snoeyink, J., 1992, Speeding up the Douglas-Peucker line-simplification algorithm, University of British Columbia, Department of Computer Science.
[77] Foster, D. E., and Pennock, G. R., 2005, "Graphical Methods to Locate the Secondary Instant Centers of Single-Degree-of-Freedom Indeterminate Linkages," J. Mech. Des., **127**(2), p. 249.

Vita

Pradeep Radhakrishnan was born in Coimbatore, India. He obtained his Bachelor of Engineering in Mechanical Engineering from PSG College of Technology in 2006. He worked at M/s. TVS Motor Company in production engineering. In January 2008, he enrolled at the University of Texas at Austin for his Ph.D. in Mechanical Engineering in the area of manufacturing and design where Dr. Matthew I. Campbell supervised him. His interests are in design automation and optimization.

Email: rkprad@yahoo.com

This dissertation was typed by Pradeep Radhakrishnan.