A Genetic Methodology for Configuration Design

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING

OF CARNEGIE MELLON UNIVERSITY

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS

for the degree of

DOCTOR OF PHILOSOPHY

GERALD P. ROSTON

DECEMBER 1994

Acknowledgments

I would like to thank my two advisors, Dr. Robert Sturges, Jr. and Dr. William “Red” Whittaker for guiding my research. Bob has provided key technical insights, has critically read through severals drafts of this work and has made himself readily accessible. Red has provided both the finincial backing for this work and a far-reaching vision of the future of robotics and this work. My thesis owes much to this unique combination of advisors.

I would also like to thank the other members of my dissertation committee, Dr. Dwight Baumann, Dr. Jonathan Cagan and Mr. John Wiss for their technical counsel and critical reviews of this work. I would also like to thank Dr. Subhas Desa for starting me down the road that has led to this thesis.

I would not have been able to have done this work without the support of the CMU planetary rover group. This group has both generously given of their time and assistance. In addition, it was through the needs of this group that the need for this work first became apparent. In particular, I would like to thank Dr. Eric Krotkov, Dr. Reid Simmons and Mr. Kevin Dowling for their support.

Much of this work was done under NASA contracts to CMU. I wish to thank Mr. Dave Lavery of NASA HQ for supporting our efforts and continuing to see to the growth of the NASA planetary robotics efforts. I would also like to thank Dr. Nicholas Colella of LLNL and Mr. John Garvey for their contributions to this work.

I would like to extend a special thanks to Mr. Martin C. Martin for his patience and help. When it became apparent to me that I would be using genetic techniques for the solution of this problem, I sought out Martin’s help since he is quite knowledgable in these areas. Without his assistance, it is doubtful that this work would have come to fruition.

Other people who have helped along the way include: Mr. Dimitrios Apostolopoulos, Dr. Hans Moravec, Dr. D.J. Laowattana, Ms. Lynn Tinsley (and the rest of the ES&S Library staff), Dr. John J. Uicker of the University of Wisconsin, Mr. David Wettergreen and others. In addition, I thank Ms. Mary Jo Dowling for providing the picture on page 57, my office mates Messrs. Deepak Bapna, John Murphy and Mark Ollis for providing the type of assistance only office mates can provide and Messrs. Ed Mutschler, Henning Pangels, Hagen Schempf and Gary Shaffer for letting me abuse their Sparc 20s.

Of course, none of this would have been possible without my parents, Marjorie and Robert Roston. They both encouraged me as a child and supported my efforts as an adult to reach this goal. Most of all, I would like to thank my life’s companion, Ms. Lorraine M. Thompson for her help and unwavering support throughout the long years of this work.

Abstract

In an increasingly competitive world, the ability to efficiently produce viable artifact design alternatives is necessary for organizations to succeed. For millennia, engineers have been using design methodologies to assist in the configuration of new artifacts. However, as these artifacts have grown in complexity, the need for more capable design methodologies has increased. This thesis presents a design methodology to aid the designer of complex artifacts by generating viable artifact design alternatives for further consideration. This methodology, called Genetic Design (GD), uses formal grammars for artifact description and representation, evaluates the artifacts automatically and manipulates the representations with genetic programming-like operations.

Human designers and optimization codes are very good at improving the performance of existing artifacts. However, due to economic constraints imposed while designing new artifacts, human designers tend to limit the range of alternative configurations considered. GD can explore a wide breadth of the available design space, though at shallow depth, and present viable alternatives to the human designer. The combination of GD's ability to explore the design space and the human engineer's ability to optimize existing configurations promotes the production of viable, new design concepts by avoiding the inefficiencies associated with trial and error methods.

GD represents an attempt to devise a design methodology that can be used across a broad range of application areas. This thesis explores two applications of GD. In the first application, GD is used to determine optimal dimensions and controllers for an abstracted model of a frame-walking robot. In the second application, GD is used for the simultaneous type, number and dimension synthesis of planar mechanisms.

Contents

List of Tables

List of Figures

1 Introduction

➤ This thesis presents a new design methodology called Genetic Design (GD). This methodology is a combination of formal grammars for artifact description and representation, automatic artifact evaluation and genetic programming-like representation manipulations. The combination of these three features yields a powerful, new methodology that is more efficient than brute search techniques. The primary limitations of this methodology appear to be the ability to represent artifacts with formal grammars and the availability of computing resources.

GD represents an attempt to devise a design methodology that can be used across a broad range of application areas. Unlike previous design methodologies, which can be grossly categorized into one of two types — descriptive or prescriptive — GD is both. This combination of characteristics suggests that GD may be applicable to real-world problems. This thesis explores two applications of GD and later, suggests a course of future work for continuing the development of this methodology.

This introductory chapter first presents the author's motivation for pursuing this line of research. Then, related work from several fields are examined. Next, a formal problem statement is presented. The last sub-section outlines the approach taken and the organization of this thesis.

1.1 Motivation

➤ The recent motivation for this work comes from the author's experience in the field of rover (mobile robotic vehicles) design. It is the author's opinion that a rigorous evaluation of alternatives is not used in the design of the overwhelming majority of rovers, but rather, the designs seem to be based on modifications of past experiences, prejudices and other constraints. In addition, even the best studies of rover design are limited in scope, considering only a very few of the large number of possible alternative configurations or some limited subset of the rover's systems. As the tasks assumed by mobile robots grow in complexity, the need to design better mobile robots will increase. To improve the design of these systems, which in some cases have no precedent from which to work, a new methodology for their design is required.

Rovers are a modern instantiation of land transportation systems, systems that engineers have been designing for millennia. The earliest systems utilized animals, or a combination of animals, with human contrived attachments. In the last century, these devices have become dominated by self-contained, wheeled machines that share certain similarities. Recently, there is renewed interest in vehicle design. One of the drivers of this renewed interest is the emergence of field robots, from their previous role of laboratory vehicles and academic curios, to their new roles of highly capable systems designed to carry out increasingly complex tasks in difficult environments. To perform these tasks, these field robots – rovers – must be capable of achieving certain levels of performance and must insure a specified probability of success in carrying out a mission.

To achieve optimal performance[1], the design of a land transportation system[2] must be a function of the terrain that the vehicle is expected to traverse. A vehicle that performs well in one setting may achieve only mediocre performance in another, different setting. The single unifying thread that ties all such problems together is the concept of the locus of vehicle-terrain interaction points.

As a vehicle traverses a terrain, it leaves behind a record of its passage in the form of contact points with the terrain: for wheeled vehicles this record is a continuous ribbon, for legged vehicles a series of points. An observer seeing this record might be able to make some intelligent guesses as to the form of the vehicle that created these tracks. This naturally leads to the idea that to design a vehicle to traverse a terrain, all possible tracks across the terrain should be examined to extract vehicle designs from subsets of these tracks. This set of vehicle designs is then analyzed to determine that which is best for a particular mission. The benefit of using this approach is that it allows the possibility of generating previously undiscovered vehicle configurations. The difficulty with this approach is that even with a constrained vehicle configuration space, the number of loci of terrain-vehicle interaction points grows combinatorially as the area being studied increases in size.

A more practical approach is to start with the vehicle and determine its interactions with the terrain. Using this idea, the problem of determining optimal vehicle configurations has three parts. The first is to develop a means to represent vehicles. This representation must be amenable to computer manipulation and must allow a means to represent a very wide variety of configurations. The second is to develop a way to evaluate the performance of the vehicles. This performance evaluation must clearly express the designer's intent, because this is the sole means of expressing the desired results. The third is to use a scheme that intelligently searches the vehicle configuration space to find vehicles that perform well. The potential size of the vehicle configuration space demands an intelligent search scheme for efficient implementation.

One tool for representing vehicles in a simple, concise manner is a formal grammar. The constructive nature of formal grammars allows both the means to generate a wide range of vehicle configurations and the ability to generate novel new configurations that were previously undiscovered. Using a grammar also allows bounding the range of parameter values to fall within predefined ranges, based on mission, engineering or computational constraints. A grammar provides a means to represent a vehicle that is easily understood by both humans and computers. In addition, the use of a grammar guarantees syntactically correct vehicles. To simplify the computer's task of parsing a vehicle, the grammar can be restructured in a functional form.

---

1. In this work, optimal is defined to be some user-specified criteria, for example, power consumption.

2. The scope of this work is limited to vehicles that move on a planetary surface in a non-negligible gravity field.

The evaluation of a vehicle must be done in sufficient detail so the search procedure can make appropriate selections. The first step in formulating a performance metric is to determine those criteria that need to be evaluated. Then, a method for calculating these criteria based on the vehicle description is developed. For vehicles, the calculations necessary for evaluation are non-trivial and might require using specialized software packages. The results of the evaluation must reflect the designer's intent, with those vehicles that are better scoring higher than those which are average. The evaluation must also detect those vehicles that are non-viable although syntactically correct.

Intelligently searching a space is a euphemism for optimization. Numerous optimization schemes exist - the oldest being calculus based and the newest being genetic based. For the vehicle optimization problem, genetic optimization schemes appear to be appropriate. More specifically, since the vehicles are variable length structures that can defined by a grammar, a modification of current genetic computing techniques can be effectively used to search the vehicle configuration space. Unlike classical methods, genetic methods only require representations of the structures being optimized and an evaluation function to determine the quality of the structures.

Although the solution outlined above was motivated by the desire to produce optimal vehicle configurations, there is nothing restricting this methodology to vehicles. Rather, the methodology presented is a new, powerful procedure for designing certain classes of objects. To use this methodology, hereafter referred to as genetic design (GD), the designer must do two things: codify the elements of the design problem in the form of a formal grammar and formulate objective functions that can be automatically calculated. The class of objects that can be designed depends primarily on the ability of the designer to cast the problem in the form of a grammar since any evaluation, whether performed by computer or not, requires the existence of an evaluation function.

This ability of the GD methodology to aid in the design of artifacts across a broad range of applications ties back to the author's long-standing interest in designing systems that are optimized for their application. In some relatively simple domains, some methods have been developed that yield good results. However, for more complex systems, such methodologies do not exist. This work presents the culmination of a decade of pondering the question of formulating a methodology for designing complex systems and making numerous "false" steps towards its solution. The word "false" is quoted for two reasons: first, each of those previous steps provided further insight into the problem and second, the implied assumption that the current approach is the "true" answer. The work presented in this thesis, although impractical to implement (except for simple cases or only a portion of a design) given the limitations of current computing technology, does appear to provide a means to designing optimal systems.

One of the central contributions of this work is the idea that a system and its controller should be designed simultaneously. This notion is appealing for two reasons. The first reason is one of practicality - by designing the controller with the system, a means for evaluating the system is provided. Without this, the GD methodology would be limited to evaluating only those systems that the designer had preconceived, and this would not allow for the generation of new configurations. The second is one of "beauty" - by designing the controller with the system, the system being designed and the means of design both more closely parallel the biological world, the world that provides the underlying paradigm for genetic computation. In the biological world, successful systems are those in which there is a match between the mechanical and computational capabilities of the system. A next, logical continuation of this work is to include the ability to model and evolve sensors along with the mechanism and controller. Given sufficiently accurate models and powerful computers, the generative capabilities of the GD methodology may produce fascinating results.[3]

1.2 Survey of related work

The work presented in this thesis was motivated by the desire to design rovers systematically, thus reviewing the literature on the design of rovers is necessary. Since rovers are little more than vehicles with the addition of computers, it is also necessary to review the literature on the design of vehicles. In the process of formulating a procedure for designing vehicles, it became apparent that a methodology for designing certain classes of systems would not only provide a means for designing vehicles, but also a means for designing a wide variety of artifacts. Therefore, it is also necessary to study the literature on design methodologies. It is also instructive to review the literature in the areas of genetic algorithms (GA) and genetic programming (GP) to learn of recent applications of these methods. The following four sub-sections review the literature in these areas and a fifth subsection reviews some other salient literature.

1.2.1 Review of rover design literature

The first rovers were built in the early 1970s. Since then, hundreds more have been developed, and thousands of designs exist on paper. However, even with all of the available rovers and the papers that they generated, there does not yet exist a scientific methodology for configuring these machines. In this section, existing techniques and previous work will be examined.

When configuring a rover to traverse a particular terrain, the first step is to determine the type of rover to employ. Each different rover configuration has inherent characteristics that distinguish it from other configurations. For different terrains, certain characteristics

3. In the book *The Hitchhiker's Guide to the Galaxy*, by Douglas Adams, the readers are informed that the Earth was built and populated by a species of pan-dimensional beings who were seeking the answer to "life, the universe and everything." Without stretching too far, it is possible to envision (within this science fiction context) the Earth as a giant instantiation of GD. Also see, *The Origin of Species*, by Charles Darwin and *Galapogos* by Kurt Vonnegut.

are more important than others, and the type of rover selected should be chosen accordingly. Most papers about existing rovers are merely descriptive in nature, giving the as-built configuration and some performance measures, but not explaining the process of why the particular configuration was selected. Even fewer papers discuss the attributes of different types of robots.

The vast majority of rovers that have been built use either wheels or legs for locomotion. Legged systems fall outside the normal experience of many vehicle designers and are not discussed in the body of vehicle design literature. The remainder of this section will focus on walking rovers. Section 1.2.2, which focuses on vehicles, will incorporate the literature on wheeled rovers since, from a locomotion perspective, they are no different from other wheeled machines.

Two papers compare different types of mechanisms for use as robotic legs, [15] and [86]. Although purportedly for robotic application, these papers are discussions of the kinematics of mechanisms. In chapter 3 of his book, [101], Todd gives a list of leg properties and important attributes for leg design. Although a comparative study is not presented, these lists present a number of different leg configurations, configurations that the GD methodology should be capable of generating given an appropriate grammar.

Several papers present studies on choosing an appropriate type of robot from a number of candidate types, [23], [43], [62] and [72]. Neither [43] or [72] provide an analytic explanation for their choice of rover configuration.

In 1988, NASA's Jet Propulsion Laboratory (JPL) commissioned reports from Martin Marietta Corporation (MMC) and FMC Incorporated detailing all aspects of a proposed Mars Rover/Sample Return (MRSR) mission. These reports are two of the most comprehensive reports about robot design found in the literature. Although these reports use a single figure of merit[4], an inappropriate method for selecting one configuration from a set of configurations, to determine the "best" rover to use for a particular mission and even though many of the selection criteria used were not strictly related to rover mobility, the vast amount of information in these reports may be useful in developing appropriate metrics for future vehicle design studies.

The MMC report, [62], places great emphasis on the mobility platform development. Taxonomies of different concepts were developed and utilized, page 4-16, (but never fully exploited) and a detailed description of the scoring of the capabilities was presented, page 8-13ff. However, by developing a single figure of merit, a design may be chosen which does not reflect the best technical approach due to the inclusion of other considerations. Some of the shortcomings of this report include using specific configurations for selecting the type of robot

4. More on this in Section 4.4.

to employ; using a relative, not absolute, scale for evaluating the capabilities of the robot types and a bias towards the machine previously developed by MMC, a walking beam vehicle.

The conclusions reached in the FMC report, [23], pages 49ff and 149ff, are intuitively obvious, and little formalization was presented to prove these statements. An extensive list of possible alternative robot configurations was generated, but most were eliminated without the benefit of a detailed examination. The assignment of weights to different capabilities to determine the figure of merit, page 63ff, are seemingly arbitrary as no scientific method of selecting the values is discussed.

Bares' Ph.D. thesis, [5], presents the strengths and weaknesses of three similar robot configurations. He does not claim that one configuration is innately superior to the others, but rather claims that for his particular application, one of the configurations may be superior to the others. By presenting a table showing the relative strengths and weaknesses of the different configurations, another engineer might chose one of the other configurations for his particular application. Bares' work clearly illuminates the difficulty of selecting between different configurations when multiple objectives exist. Bares' work can be augmented by providing a framework for generalization to other types of rovers, considering a broader range of capabilities, providing a better quantification of the capabilities, addressing the issue of scale and providing a computer tool to perform the comparison.

Only one paper was found which directly addresses the issue of configuration design, [93]. This paper shows how the dimensions of the Ohio State Adaptive Suspension Vehicle (ASV) were determined. The method used was to model typical terrain features, such as steps and ditches, and to use geometry to determine the kinematic dimensions of the ASV to enable it to negotiate these terrain features. This paper also shows how modifying certain kinematic parameters affects mobility. However, this paper only addresses one type of rover. Further discussion of the ASV can be found in [96] [101] [42].

Since legged rovers do not remain fixed in one location, a performance evaluation of these machines must include more than simple motions, such as single leg and body moves. Instead, the robot's performance as it traverses terrain that it would encounter during a mission must be evaluated. To perform these analyses, an understanding of the system and its interactions with the terrain as well as the capability of simulating the system are needed. Although there are many papers about dynamics, few deal with multi-legged walking machines. These machines are difficult to analyze because they are redundant, closed chain mechanisms. Bekker [6] [7] spent many years studying the effects of terrain-vehicle interactions for wheeled vehicles, and much can be learned from his work. Nakamura [69] presents both a model for walking machines as well as some preliminary results of a simulation. This work is limited in that it only considers hard, flat floors. The intent of his work seems to be modelling the walking by changing feedback gain parameters rather than

determining performance measures. Another simulation reported in the literature, [19], is simply a kinematic analysis.

Two thorough examinations of walking machine dynamics are Manko's and Nagy's theses. Manko developed a dynamic model of a walking rover and ran some tests in simulation [61]. More importantly, his work provided a dynamic mechanism model that characterizes the interactions of a closed-chain robot with the terrain. Nagy's work [68] built upon Manko's work by combining the simulation work with experimental data to understand vehicle-terrain interactions. The results of these works need to be incorporated into the evaluation formulations for walking vehicles to ensure some degree of fidelity. Other papers that report on legged vehicle locomotion, including terrain-vehicle interactions include [50] [63] [78].

1.2.2 Review of vehicle design literature

While the original work in the area of vehicle design was performed for off-road vehicles [29] (since roads didn't exist at the time), most modern research is dedicated to on-road vehicles for the simple reason that the world-wide, on-road vehicle manufacturing is a multi-100 billion dollar industry. The American Society of Automotive Engineers holds numerous conferences and publishes hundreds of papers annually, and the overwhelming percentage are dedicated to on-road vehicles. Unfortunately, on-road vehicles are a special case of vehicles, and much of this research does not apply to off-road vehicles. Some sources, such as Gillespie's book [32], provide a thorough understanding of on-road vehicle design and can be used as a basis for further work.

The primary concern with off-road vehicle design is the vehicle-terrain interactions. To design a successful off-road system, these interaction must be properly understood. The seminal research in the area of terrain-vehicle interactions was performed by Bekker [6], [7]. In these books, he quantitatively described the terrain-vehicle interaction, developed heuristics for configuring vehicles and laid the groundwork for future researchers. [1] is a more recent rendering of Bekker's works.

The *Journal of Terramechanics* frequently has articles in the area of terrain-vehicle interactions. The majority of these papers deal with important sub-systems of the vehicle. Example topics include design characteristics of steering systems for earth-moving equipment, climbing ability of four-wheeled vehicles, dynamics analyses of off-road vehicles, vehicle stability, etc. (for instance, [108]). Although the topics covered by these papers are extremely important, none provide a methodology for designing a vehicle. The authors of these papers assume the reader has a vehicle for which further investigation is required. None seem to start with the premise of the designer staring at a blank sheet of paper. What is needed, and what is developed in this thesis, is a means to incorporate this vast accumulation of theory, heuristics and ideas into a design engine; one that can be used to aid an engineer in designing a vehicle.

While much of the vehicle research has been motivated by the needs of the military [54], the expanding need for field robots has caused renewed interest in this are of research. One of the key differentiators between field robots and other vehicles is the control system: field robots are controlled autonomously or remotely where other vehicles have human operators present. Humans are so good at operating vehicles that inaccuracies in modelling the vehicle-terrain interactions are easily compensated for by the operator. For remotely operated vehicles or autonomous vehicles, the capabilities of the human operator are not available, so a much better understanding of the interactions are required to safeguard the rover.

Two of the previously mentioned works, [23] and [62], also have extensive reviews of a few wheeled rover concepts. Although the previous comments about the weaknesses of these reports also hold for wheeled machines, the extensive lists of evaluation criteria and mission level issues presented may provide useful guides for formulating criteria for future vehicle design studies.

To increase their understanding of planetary rover systems, the researchers at MMC built a wheeled rover in addition to the walking beam, see page 6. This work is explained in [77]. This paper clearly specifies the design goals and presents an interesting trade space. The trade space presented is similar to the higher level features (Section 1.4) and is subject to the same difficulties. Furthermore, although a trade space was described, there appears to have been little search throughout the entire space to find both the appropriate configuration and dimensions for the vehicle that was built. With an appropriate grammar, the GD methodology should be capable of generating any vehicle that can be described by the trade space.

The vehicle designed by Martin Marietta was a four wheeled, two cab rover with an active roll joint, parallel to the direction of motion, between the two cabs. With a renewed interest in lunar exploration, Sandia National Labs designed and built a rover for planetary exploration [49]. This vehicle is also a four wheeled, two chassis rover, but for this vehicle, the chassis are joined by a passive pitch axis perpendicular to the direction of motion. Although no rationale for the particular configuration chosen is given in the paper, a good description of the sizing of several key components is presented. At present, a copy of this vehicle is undergoing extensive testing at Carnegie Mellon.

Two of the more recent planetary rover designs have had four wheels and two chassis. Although the means for reaching this design decision is not made clear in either case, one is left wondering if these decisions were made because the utility of this design is self-evident, or if there are other reasons. If the design is the natural optimal design, the GD methodology should also converge to this solution given similar requirements.

Another current wheeled, planetary rover concept is the JPL rocker-bogey family of vehicles. Although not descriptive of the design process, [16] presents a very thorough examination of the performance of this vehicle. All of the equations were developed parametrically, and some attempt was made to explore the space spanned by the parameters to

determine an optimal set. However, this was done primarily by modifying some small number of the parameters to determine those parameters affects on particular criteria. This problem appears to be ideally suited to a GA approach, an approach that could yield some exciting results. Unfortunately, this report was prepared after the vehicle had been built, so the results were not incorporated into the design. Other papers that have examined the locomotion for a Mars rover include [60] [79] [97] [107].

Two other papers on "wheeled" rover design should be mentioned. The first, [89], proposes a solution to the very challenging problem of designing a rover to pass through a very small opening, without exceeding certain terrain contact pressures while at the same time being able to perform excavating type activities. Although the means by which the configuration was selected is not presented, the use of the equations for the terrain-vehicle interactions is informative. For this problem, the proposed solution is a tracked vehicle. The second is [10] which proposes what can only be termed as a hybrid rover for lunar exploration. This vehicle uses tracked locomotion elements at the ends of a Stewart platform, thus allowing the machine to reconfigure itself and giving it some unique capabilities. Although there is additional cost in the added complexity, the greater mobility of the system might out-weigh that cost. This paper is important because it indicates that hybrid solution must be considered, and the methodology proposed in this thesis, because of its generality, can handle such designs.

As these two sections have shown, a vast amount of research in the areas of rover design and vehicle design has been done. However, none of the reviewed research has presented a systematic means for designing these systems. In the next section, design methodology research will be reviewed, with the goal of finding a methodology that can be applied to this problem.

1.2.3 Review of design methodology literature

Research in the field of mechanical design methodologies is one of the most active field of research in mechanical engineering, and related disciplines, today. The work in this field is so vast, that there exists a paper that provides a review of current research in the field - a "tour guide" for the uninitiated [24] [25]. Rather than duplicating the review paper, this brief over-view will highlight several works that place the current work in context and will examine other works that present similar approaches. It is this author's opinion that the current thesis work is more general than the previous works and that many of the previous works can be subsumed by the current thesis work. Thus, it is more important to identify the strengths of the previous works as opposed to trying to find shortcomings.

In their seminal work, Pahl and Beitz [73] introduce a systematic approach to engineering design. This approach divides the process of engineering design into four phases: clarification of the task, conceptual design, embodiment design and detailed design. In addition to providing a systematic approach, the book also provides tools and examples for help-

ing the designer. Within the context of the four-phase model of design, it is important to determine where the current work fits into this model. It is believed that the current work can be applied to all phases except the clarification of task phase, see Chapter 4 of this thesis.

Although Pahl and Beitz's work provides a solid framework in which one can carry out engineering design, engineering design has yet to be defined. The definition offered by Dym [20] is used:

> Engineering design is the systematic, intelligent generation and evaluation of specifications for artifacts whose form and function achieve stated objectives and satisfy specified constraints.

The author feels that this definition comes closest to describing the process of engineering design as implemented by the genetic design methodology. One of the key, underlying assumptions incorporated in this definition is that hierarchies of representation for both form and function exist, and that these representations can be manipulated. In addition, it is necessary that these representations can be translated into other representations, including a final physical instantiation of the artifact represented.

There exist numerous techniques for representing systems and artifacts. One of the most general is the function logic method of value analysis [99]. With this technique, objects, and classes of object, are represented by a hierarchy of noun-verb pairs. Although this technique is capable of expressing a very wide range of concepts, computer implementation is difficult and it lacks some of the formality of other methods. A related methodology, [14], presents the concept of designing from the basic, underlying principals. Although this method was shown to be successful for certain, simple problems, it does not appear that this method can be generalized for more complex systems.

Another approach used is bond-graphs [26]. In this paper, the authors state, "During the design process, a designer transforms an abstract functional description for a device into a physical description that satisfies the functional requirements." (This definition is quite similar to the one given above.) To achieve this goal, the authors propose using bond-graphs, a tool for describing generalized lumped-parameter dynamic systems, to achieve the required transformation. The bond-graph technique works well in the domain for which it is intended, however, it is not clear that this representational domain can be extended. Also, the authors did not propose a methodology for automating the process, a necessary step for producing a range of alternatives from which the designer can chose.

Other means of representation is the use of formal grammars. Some of the earliest work in this area was performed by Stiny [98] who developed the concept of shape grammars. Shape grammars, which are used to describe planar shapes, have been shown to be equivalent to other types of formal grammars [33]. More recently, Mullins and Rinderle [67] [83], Schmidt and Cagan [90], and Reddy and Cagan [81] have used grammars for the design of mechanical systems.

The introduction to the Mullins and Rinderle's paper [67] clearly states several reasons for using formal languages for mechanical design:

- Formal methods facilitate the characterization of the mechanical design problem more precisely.
- The absence of a fixed structure in mechanical decision-making (e.g. hierarchical decomposition) makes formal but less rigid methods attractive.
- Recent work in the representation of geometry as strings or graphs makes methods which operate on strings or graphs useful.
- A grammatical approach to design makes it possible to leverage related work in computer science.

The Mullins/Rinderle paper presents two examples, one using a context-sensitive grammar (CSG) and the other using a context-free grammar (CFG) (see Chapter 2 for definitions). Although the examples shown in the current work use only CFGs, the Mullins/Rinderle paper shows that this is not a necessary restriction. In the concluding remarks of the Mullins/Rinderle paper, the authors state that there are several critical issues in applying grammatical formalism to mechanical designs. One of those issues is the generation of good designs. The current work presents a methodology for using this formalism to achieve the stated critical issue, generating good designs.

Schmidt and Cagan [90] propose an abstraction model for conceptual design. A conceptual design progresses along several levels of abstraction, from a high level, black-box description "convert electrical energy into mechanical energy" to a low level, component technology description "use an electric motor". The authors develop a grammar that can distinguish between different levels of abstraction while being compatible across the different levels. The strings formed by these grammars are manipulated by a recursive annealing process to produce results that optimize a designer- specified objective function. There appear to be two difficulties with this approach that may limit its applicability to more complex problems than the sample problem illustrated in the paper. First, incorporating the required features into the grammar results in a fairly complex grammar that is non-trivial to generate. The authors note that to fully capture the example problem, the grammar given is not sufficient. Second, while the authors specifically state that this is a model of conceptual design only, greater detail of the component technologies, such as dimensions, are required to verify the feasibility of a conceptual design. Since their grammar does not incorporate dimensions, an apparently good design might require components that are not realizable. The authors of this work believe that these details can be accounted for in their model.

Reddy and Cagan's [81] work uses shape grammars and simulated annealing to solve a variety of design problems. Reddy/Cagan's work is important because it shows how a grammatical representation of an artifact and a means of intelligent search can be used to generate optimal designs. The current work extends this work by using a "better" means of intelligent search and expanding the concept to more complex systems. The author believes that the GA/GP method of search presented in this work is superior to simulated annealing

for this type of problem for several reasons. The GA/GP approach is designed specifically to deal with tree representations; GP has been shown to work on a variety of problem with similar representations; and the GA/GP approach is less sensitive to the proper selection of control parameters, see Section 4.5.

In addition to these works, a number of researchers are also trying to understand the design process itself — that is, how people do design, see [102] for example. From the perspective of the current research, the key finding from this body of research is that engineering designers are quite adept at optimizing a given design, but rather poor at generating alternative design options. This finding directly influences the form of the results generated by the current research, Section 1.3.

1.2.4 Review of GA/GP literature related to design applications

Many of the current papers and books related to GA and GP are focused on the implementation of the methods. This is not unexpected due to the relative newness of these techniques. GA literature will be reviewed first, then the GP literature. The work presented in this thesis, while novel, utilizes fairly straight-forward GA/GP techniques. Future work may include implementing some of the theoretical work being done in the GA/GP fields to improve the performance of the method developed here.

The primary source for GA theory is Holland's 1975 book [39]. This book sets forth schemata theory and set the stage for all future developments in the field. This book is highly mathematical in nature and does not provide any "real-world" examples. Another important source is Goldberg's 1989 book [34]. This book does not delve as deeply into the mathematics as Holland's book, but rather presents more of a user's guide to GA. This book provides some examples as well as a listing of, the then current, applications using GAs.

Because of the means of representation, GAs are not well suited to open-ended problems. A good example of this limitation is presented in [3], in which the author uses a GA-type approach, but does not use the traditional bit-string chromosomal representation. Since GAs work best with fixed-length encoding, the GA design research has been limited to problems whose topology remains fixed, but whose parametric values can be changed. Some example application areas include controller parameter estimation, filter design, scheduling, truss and strut optimization, etc. (The proceedings of the annual conference on genetic algorithms typically has papers about new GA applications. [80] [106] [88] [8] [28]) Two recently published articles use GAs to solve mechanical design problems. The first [12] starts with a user specified configuration and applies GAs to select appropriate parts from a catalog. The second [76] designs a transmission based on user requirements from a set of primitive elements. Both works show that genetic approaches to design are viable, though neither is as general as the work presented in this thesis.

Recently, Goldberg introduced the notion of "messy GA" [36]. With this technique, the chromosome size of the members of the population are not constrained to be of the same length. The additional freedom allows the expression of a wider range of objects and permits the creation of new objects. However, using a bit-string to represent an object is not intuitive and bit-string cross-overs (see Section 3.1, Figure 5) yield different results from tree-node cross-overs.

GAs have also been used to aid in the modeling of networks. [84] presents such a system, however, the methodology described applies only to predefined network topologies. This work, see Chapter 6, requires a methodology that can manipulate the topology of the network.

GAs are also used in classifier systems. A classifier system is a type of machine learning algorithm that is formulated from a number of productions, statements typically of the form:

if <condition> then <action>

By setting up a set of these productions and testing them on known cases, the algorithm increases the weight assigned to a production if it is activated during the learning phase. Once the weights have been assigned, new cases can be input to the classifier system and results obtained. [71] presents examples of how this technique can be applied problems with multiple objectives (see Section 4.4 for further discussion of multiple objectives). However, these examples assume that an evaluation of the artifacts exists *a priori*, and the program discovers the rules used to generate the ranking. Although this method does provide a straight forward means for handling multiple objectives, for the general problem of artifact design, *a priori* rankings are not available, so it is not readily apparent how the classifier system approach could be used to solve this problem.

The first paper on genetic programming was published in 1989. Much of the early work was done by the inventor of GP, Koza, and is reported in his book [52]. Numerous examples are given in the book and in [48] and [53]. These include symbolic regression, controller design, path planning, task prioritization, etc. In each of these examples, the genetic representation of the individual, the program, is executed directly. (How the current work differs will be explained in Chapter 4.

In addition to Koza's books, there are three papers that have been important in the development of this work. The first, [65], generalizes Koza's constrained syntactic structures. This generalization yields an new type of GP called strongly-typed GP (STGP), see Section 3.3. The second, [70], is the only paper (known to this author) in which the GP approach has been applied to something other than computer programs. Although the problem presented in the paper is not as rigorously defined as the problem solved in this thesis and although the use of a grammar for defining their models is not explicitly stated, the fact that the GP representational form is used for artifacts lends credence to this thesis work. Finally, the author of [41] and this author were simultaneously working to solve the same

problem, that of representing constants in GPs in a more compact form. By corroborating, both were able to clarify their thoughts.

The recent works of three authors deserve special attention because their works are similar in nature to the work presented in this thesis.

In the first, [58], the authors present a methodology for performing adaptive design using graph representations and GA representation manipulations. The illustrative problem used is the topology of floor plans. Although this work does achieve its stated goal of finding topologies that satisfy certain constraints, there is little indication of how this work can be extended into other domains. In addition, there is no indication of how to incorporate numeric values into this methodology. For example, while the results from the sample problem show that the auditorium should be adjacent to the corridor, there is no way to determine the relative sizes or even to determine if the proposed topology would work in the context of a restricted area. What is interesting is that the means of representation chosen by the author to define floor-plan topologies is similar to the means of representation that the author of this thesis developed to represent planar mechanisms, Chapter 6. This suggests that this representational scheme can be used across a wide variety of problem domains.

In the second work, [56] [57], the author proposes a methodology called *structure evolution* for determining the parameters and structure of an object. This work is based on an adaptation of evolution strategy, a genetic methodology that is related to GA. The author appears to focus much of his attention on issues central to the evolution itself, such as using semi-isolated populations, as opposed to defining a structure into which other problems can be fit. In these papers, the author presents solutions to four problems in four diverse domains. Although good results are obtained for each of the example problem domains, there appear to be some limitations to this work. First, the author does not explain how the problems are represented, although this may be for pragmatic reasons, i.e., page limitations. Second, the degree of structure variation permitted in the various problems appears to be limited. Finally, the author does not present a means for adapting this methodology to other domains. Although this work does present good results, it seems that this work is inherently limited to problems that require a particular means of representation.

In the third work, Sims, [94] [95], the author presents a means for creating “virtual creatures”. The goal of the work presented in [94] was to generate creatures that exhibit certain types of behaviors, such as walking, jumping, etc. The goal of the work presented in [95] was to model the co-evolution of creatures that compete with each-other for a limited resource. These works were performed on a Thinking Machine CM-5 super-computer. One of the outstanding features of these works is the graphical results generated, a “movie” of which is available on the Internet. This work is recognized as being one of the more important works in the field of artificial life.

Sims' work has few limitations. The work uses directed graphs to describe the creatures' genotype, which is used to generate the creatures' physical phenotype as well as its controller. This means of representation, while philosophically similar to the use of a formal grammar, does not allow for the same level of descriptiveness. Furthermore, it does not appear that this means of genotype representation can lead to network-like phenotypes, such as planar mechanisms. The same objection can be raised with regard to the controller - the use of a formal grammar representation can yield controllers that are more complex and hopefully controllers that exhibit more complex and/or interesting behaviors. Sims' work appears to be tightly focused in the one domain in which he worked, the physical simulation of these virtual creatures. In focusing the work, the dynamic simulation of the creatures, the evaluation function, seems to be an integral part of the methodology. The current work does not require the evaluation to be an integral part of the modelling program, nor is it tied to a single domain, thus its future applicability might be greater than for Sims' work. Finally, Sims does not tell a lot about the constraints, design evaluations, etc., thus making it difficult to properly evaluate his work.

1.2.5 Review of other literature

One issue that must be addressed in many design problem is that of scaling — both the absolute size of an artifact and its relative size with respect to their environment. For the vehicle design problem, the issue of scaling arises both during the "design" phase, specifying the sizes of the vehicle elements to allow it to locomote effectively, and during the "evaluation" phase, during which the vehicle must negotiate a feature-rich terrain. For example, from the vehicle's perspective, there are three ways to negotiate a terrain feature: to simply ignore it (the robot is much larger than the feature), to surmount the feature (the robot is of similar size to the feature) or to go around it (the robot is much smaller than the feature). The need for scaling is also presented in Section 6.1.4.11. Several sources report the effects of scaling of biological systems [64] and [74], but only one was found which reports on the effects of scaling in robots, [46]. Although this work makes a number of simplifying assumptions which may limit its practical application for designing rovers, it provides a solid analytical basis. In all cases, dimensional analysis is used to determine invariants. [51] presents a general discussion of dimensional analysis, but does not apply it to particular systems.

Another work of interest is Braitenberg's book [11]. While primarily philosophical in nature, this book present a series of "experiments" in which seemingly simple vehicles with simple controllers and control algorithms exhibit fairly complex behaviors. The series of experiments discussed in this book suggests a path of research that may be followed in developing complex systems. However, since the goal of the experiments is to develop system that appear to be intelligent, it is not clear how this appearance could be measured. Furthermore, Braitenberg suggests that intelligence cannot be measured by looking in from the outside. To quote Braitenberg:

> ... for what I would like to call the “law of uphill analysis and downhill invention.” What I mean is this. It is pleasurable and easy to create little machines that do certain tricks. It is also quite easy to observe the full repertoire of behavior of these machines — even if it goes beyond what we had originally planned, as it often does. But it is much more difficult to start from the outside and to try to guess internal structure just from the observation of behavior. It is actually impossible in theory to determine exactly what the hidden mechanism is without opening the box...

The implications of this will be more fully discussed in Chapter 7.

One final item to be reviewed is Kim’s Ph.D. dissertation [47]. In his dissertation, Kim proposes a design methodology call Task Based Design (TBD), a methodology for designing optimal robot manipulators. Although this methodology is for manipulators, it is designed to solve the same class of problem - the design of complex systems. The TBD methodology is used to determine the optimal parameters, base position and poses for a manipulator to carry out a specified task. Since Kim restricted the scope of his work to include only those manipulator arms comprised solely of revolute joints, he exhaustively searches the space of all possible manipulator configurations to find the optimal one. For low degree-of-freedom arms (the type most likely to be designed), the space is not very large, and can be explored efficiently. Since the space being searched is well defined, Kim uses a GA approach for finding optimal manipulators. Although the TBD methodology has been used successfully, it does not appear that it can be readily extended to other domains because its success is, in part, dependent on the “limited” domain it works in. However, the success of the technique in automatically generating complex systems to optimally achieve a specified task lends support to the notion that the design of complex systems can be automated, at least to some degree. Additionally, it should be possible to use the GD methodology to attempt to solve the same problems that have been solved using TBD. However, since TBD was designed specifically for working within the manipulator domain, one would expect that it would yield superior results than the more general GD, because of the domain specific knowledge encoded in TBD.

In summary, the review of the literature has shown several things: first, although vehicles have been designed for many years, there is as yet no straight-forward means for designing these systems. Most of the vehicle designs reported do not discuss the crucial design decisions that led to a particular configuration. Second, although there are numerous design methodologies, none seem well suited to the task of designing complex systems. Third, evolutionary computational techniques have the capability of being able to intelligently search large, discontinuous spaces. By expanding these techniques, they become well suited for artifact design. Thus, this research will endeavour to build upon the previous works to develop a system that overcomes some of the aforementioned short-comings.

1.3 Problem statement

➤ The original problem for which a solution was sought was to configure a vehicle that successfully and optimally traverses some specified terrain. While working to solve this problem, several things became apparent: the number of vehicle parameters that need to be considered is quite high; the number of possible vehicle configurations is essentially infinite; and there is no single correct solution as the problem has multiple competing objectives. These realizations caused a shift in focus, from trying to configure a vehicle to discovering a means by which a vehicle can be configured. Further reflection indicated that designing vehicles as not significantly different from designing other complex systems, and that with little added effort, a means of designing complex systems could be developed.

Thus, the underlying problem that this thesis sets out to answer is this: Does there exist a methodology for designing complex systems? If such a methodology can be discovered and developed, it can be applied to the original problem of vehicle configuration, yielding the originally sought answers.

Although the wording of the above question appears to be quite simple, it is important to discuss the meaning of each of the key words to properly understand the meaning and scope of this thesis work. The word *methodology* implies a general, context-free technique for doing design. By context-free, the ultimate solution should not be limited to particular sub-domains of engineering, but rather it should be able to work within any domain. The word *designing* is used in its most general sense. Some design methodologies are restricted to particular aspects of the design problem, such as preliminary design or conceptual design. This methodology should be as free from these restrictions as possible, with the ability to be applied to any level of the design problem the engineer needs help with. Finally, *complex systems* means a set of components that interact with each other and their environment in a non-trivial manner, and whose interactions are not easily described by closed-form equations. Section 4.2 will expand upon this definition. This is not to say the methodology developed cannot handle simple systems, however, it must also be able to handle complex systems. As an example, consider the difference between designing a single spur gear pass and designing an automatic transmission.

At first glance, the problem statement and the goals of this research may appear to be so lofty that they are impossible to achieve. However, this research shows that there is a simple methodology that can achieve these goals. This is not to say that finding the answer to the original question of vehicle configuration is trivial. The methodology developed does require domain specific information for each domain in which it is to be used, and this information might be quite complex. What the methodology does provide is a single, context-free framework for utilizing this domain specific knowledge in a manner that can yield new solutions and new insights into the problems for which answers are being sought.

One of the difficulties encountered when developing this new methodology was the means of evaluating candidate designs. If the methodology required the designer to implement a means of evaluation for each possible, candidate design, the methodology would be of little value, since to implement these evaluation routines, the designer would have to first enumerate all possible candidate designs. The secondary question then arose: Is it possible for the design of a complex system to incorporate a means of control into the design itself? By incorporating a controller into the design, the engineer using the methodology is no longer burdened with enumerating all possible configurations *a priori*, but instead must provide a generic means of *evaluation*.

To clarify this last point, consider the original question of configuring a vehicle to traverse a terrain. If a means of vehicular control were not built into the vehicle design, the designer would first enumerate all possible vehicle configurations. This being an impossible task, the designer would be limited to some relatively small number of configurations. For each of these configurations, the designer would then determine how the vehicle moves, i.e., its gait. Given a vehicle and its gait, a wide range of analyses can be performed. However, the designer would be limited to studying only those configurations for which he had enumerated. By simultaneously developing the vehicle and the gait, however, the entire space of possible vehicle configurations can be explored, since no *a priori* enumeration is required.

Using similar reasoning, the concept of developing the controller at the same time as the system should be expanded to include the system's sensors. Experience and experiments have shown that, in general, greater sensor capabilities facilitate greater system functionality, however, sensors are typically expensive. By simultaneously developing the sensor with the system, the "proper" degree of sensing will be incorporated into the design, allowing the required degree of functionality without the burden of excess sensors.

Finally, the products generated by the methodology must be considered. The research into design methods suggests that engineering designers are quite good at optimizing design configurations, but rather poor at generating alternate designs. In addition, the discussion of multi-objective optimization, Section 4.4, suggests that regardless of the technique used, there is always some ambiguity in selecting the single, optimal choice. Both of these considerations indicate that presenting the designer with several good alternatives would be more useful than presenting a single alternative.

1.4 Overview and approach

As described in the previous section, the original plan of research was to develop vehicle configurations. The original approach taken was to enumerate vehicle features, then iterate through this space of features until certain conditions were met. The problems with this approach were two-fold: first, the problem was computationally intractable; and second, combining vehicle features proved to be extremely difficult. To make matters worse, increasing the accuracy or fidelity of the model would make the problem even less tractable.

Like many other solutions to many other problems, the solution to this problem came from an unexpected direction: the author read the book *Artificial Life*, by Stephen Levy. [55] Reading this book directly led to the solution of these two difficult problems. The first problem was solved by using the ideas of genetic computation to intelligently search the vehicle configuration space. The second problem was solved by realizing that vehicle features are actually higher level constructs of more basic features. These more basic features are easier to manipulate and can be assembled in a large number of ways that leads to a large space of possible vehicle configurations. What really helped cement the pieces was the realization that biological vehicles were developed in the same general manner, and that the approach has been proven to work.

The approach taken in formulating this work also mirrors the biological approach[5] in the sense that the first problems to be solved are relatively simple, and that once the methodology has successfully solved some relatively simple problems, more difficult problems will be tackled.

➤ This thesis is divided into four parts. The first is a review of formal grammars, genetic algorithms and genetic programming. The second discusses the new contributions of this work, in particular, the genetic design methodology. The third is an application of GD to two different problem domains. The fourth is a review and summary.

To implement the GD approach as previously out-lined, the first step is to determine a means for clearly expressing the artifacts being designed. The method chosen must be unambiguous, amenable to computer manipulation, be domain independent and be able to express a wide variety of concepts. In Chapter 2, formal grammars are discussed, and it is found that formal grammars provide these attributes. Of the various types of formal grammars, context-free grammars are initially chosen because sentences formed with context-free grammars are typically easier to parse than sentences formed with other, more descriptive, grammars.

The next step is to examine current implementations of genetic computation and to determine if they are directly applicable or if they require modification. This is done in Chapter 3 by first presenting a primer of the basics of genetic computation, then exploring their applicability. What is discovered is that neither traditional GA nor GP exactly fit with the requirements for GD, and that a combination of the two is required. In addition, some extensions to this hybrid GA/GP approach are made.

Chapter 4 uses the results of the previous two chapters to formulate a new design methodology called genetic design. This methodology combines the descriptive qualities of formal grammars with the intelligent search capabilities of genetic computing to design complex systems. In developing this methodology, it is found that there exist essentially

5. Of course, biological systems do not follow a predefined approach.

two classes of design problems, and that one of these classes requires the co-evolution of a controller along with the artifact.

Since the author's primary interest is vehicular design, this is the problem studied. Since the design of a real vehicle is a tremendously complex task, the first problem is a simplified, abstracted problem. Chapter 5 studies the problem of designing a "stepping-stone" vehicle. The stepping-stone vehicle is an abstraction of a multi-legged vehicle crossing a body of water by only stepping on stones. Although this problem is simple, it provides a good, first exercise for the GD methodology because the solutions to this problem are intuitive, and the resulting designs can be checked to insure that they make sense.

Three sets of experiments are run with this model. In the first set, the terrain traversed is periodic. Using this simple terrain model, the optimal results can be predicted. By comparing the results to the predicted values, the correctness of the generated solutions can be determined. In the second set, the terrain is periodic but with superimposed noise. For those terrains with little noise, the results should be similar to the first set of experiments, however, for very noisy terrains, the resulting solutions will no longer be close to the theoretical optimal solutions. In the third set of experiments, the same terrain as in the second set of experiments is used but the ability to generate a vehicle level controller is added. This controller incorporates information about the terrain that is obtained from "sensors". With the addition of the controller, the performance of the vehicles improves.

Chapter 6 uses the GD methodology to solve some planar mechanism synthesis problems. This problem is introduced because it is a well understood problem for which there exists a large body of literature and because planar mechanisms are sometimes used as legs for walking vehicles. Although not addressed in this thesis, these two sets of experiments could be coupled to generate a more realistic walking vehicle model. First, a GA problem is set up to design four-bar mechanisms that move "through" any number of points in the plane. This problem is studied to help develop evaluation functions. Then, a grammar is developed that can describe a class of planar mechanisms. By using the grammar with GD, the type, number and dimensions synthesis problems for certain types of planar mechanism synthesis problems is solved.

To assess the potential of the GD methodology, for each set of experiments, the following items will be observed:

- Do the generated designs accomplish the specified task?
- Are the generated designs the same as, or similar to, the predicted optimal designs?
- How long does it take for the design synthesis to run on a specified computer?
- How does the run time change with increasing problem complexity?

Chapter 7 summarizes the work, presents some conclusions and suggests future work.

2 Introduction to formal grammars

➤ The GD methodology makes use of formal grammars to describe and represent artifacts. The difference between formal grammars and ordinary grammars is that all sentences formed with a formal grammar must adhere to a set of precise grammatical rules. This chapter first motivates the need for formal grammars. Then some of the basic concepts of formal grammars, including some necessary vocabulary for discussing formal languages and the underlying rules that govern language composition are introduced. Finally an example is presented.

To facilitate the communication of ideas between individuals, a common language is used. For example, this dissertation is written in English to express the idea of the genetic design methodology. If properly used, the language conveys the ideas succinctly and unambiguously. To aid communication, numerous "languages" have been developed: the "language" of math, the "language" of economics, etc. These languages are best suited to describing concepts within their domain, but may fail to clearly communicate ideas outside that domain.

The need to precisely describe artifacts suggests that a "language" for their description is needed. A set of line segments drawn on a Cartesian plane (a sketch), is a type of artifact. To describe these sketches, a "standard" language, such as English can be used, however, such a language is ill suited to conveying such a description precisely. The description of a tic-tac-toe board, in English, may be given as:

> Two sets of two parallel line segments of a specified length, separated by a distance equal to one-third their length with the ends aligned, with one set rotated 90 degrees with respect to the other, overlapping the other set and being displaced both to the right and down by a distance of one-third its length.

Not only is this description of a simple sketch extremely verbose, it is also possibly ambiguous and would pose grave difficulties for automatic interpretation by a computer.

Consider this next example which makes use of a language designed for representing sets of line segments. In this "language", each curly-bracket enclosed, comma-separated quadruplet specifies the starting point and ending point of a line segment (on a Cartesian plane):

{{0,1,3,1},{0,2,3,2},{1,0,1,3},{2,0,2,3}}

Not only is this representation shorter, it is unambiguous and can be easily manipulated by an appropriate computer program. This simple example clearly demonstrates the need to specify artifacts in a language that is specifically designed for their expression.

An alphabet is any finite set of symbols. This set of symbols may contain letters, digits, other characters or even abstract concepts like shapes. A sentence is any string of finite length composed from symbols in the alphabet. A language is any set sentences. Most meaningful languages contain an infinite set of sentences. By generalizing the notion of an

alphabet to be comprised of tokens, where a token is a symbol or a non-divisible grouping of symbols, the alphabet can be referred to as the terminals of the language.

Given an alphabet, how is a language represented? For a finite language (an uninteresting one), all possible sentences in the language can be listed. However, this is clearly not possible for an infinite language. For an infinite language, an alternate, finite representation is required. One such means of representing a language is the use of a grammar. It should be clear that a grammar describes the appearance of a sentence (its syntax) but not its meaning (its semantics).

A grammar is comprised of four components [2] [40]:

1. A set of tokens called terminal symbols. These are tokens (mentioned above) or variables. (See Section 3.2, Figure 10 - terminals have `Arity` = 0.)
2. A set of non-terminals. Non-terminals are syntactic categories expressed using the set of terminal symbols.
3. A set of productions. A production consists of a set of non-terminals and terminals, an arrow and a sequence of tokens and/or non-terminals. A production is actually a mapping - the left-hand side is replaced by the right-hand side.
4. A start symbol, which must be a non-terminal.

For notational clarity, non-terminals are italicized, keywords are bold, and terminals will be in the regular font. In addition to the four components listed above, the sequence mentioned in step 3 may also contain the symbol "|" which represents the logical "or" operation. Thus, a sentence can be said to be generated by the grammar if it consists solely of terminals or if it can be derived from the set of terminals and productions.

The grammar described above is called a type 0 grammar. Adding the restriction that the length of the set of symbols on the right-hand side of the production in step 3 above is longer than the set of symbols on the left-hand side, yields a context sensitive grammar. Restricting the left-hand side of the production to contain only a single non-terminal yields a context-free grammar. Restricting the left-hand side to consist of a single non-terminal and restricting the right-hand side to consist of a single terminal or a single terminal and a single variable yields a regular grammar. Clearly, all regular grammars are context-free, all context-free grammars are context sensitive, and all context-sensitive grammars are type 0.

For this work, context-free grammars, also referred to as Backus-Naur Form (BNF), will be used. Although it is conceivable that a less restrictive grammatical form could represent a wider variety of artifacts, there is a considerable body of previous work in the area of context-free grammars from which one can draw. Context-free grammars are parsable in linear times — the less restrictive grammars typically require much greater parsing time because certain sentences can be interpreted in several ways. Although only one of the possible parsings will be correct, many will, in general, need to be checked to determine that

one which is correct. Additionally, a sentence in a context-free language can be represented in a tree form, a representation that aids human understanding.

The language created by a grammar can only be understood within the context of that grammar. For example, although a person from China hearing English spoken would believe that the sounds have meaning, the sounds would be little more than noise. Similarly, a context-free grammar designed to be used within a specific context cannot be expected to be understood outside of that context. This leads to the important understanding that if the context as well as the grammar are to be defined, context-based information can be encoded into the grammar. Referring back to the tic-tac-toe board example, the English language is "designed" to represent a very wide range of concepts, but is not specifically "designed" to unambiguously describe the tic-tac-toe board. The language designed to represent line segments is not only understandable within the context of describing line segments, but it also incorporates context-specific information, such as the use of a Cartesian space for defining the endpoints and the ordering of the tokens to describe a line segment.

To help clarify these concepts, a grammar to describe a collection of line segments will be formally defined. Before defining this grammar, several assumptions are stated.

- All line segments lie in first quadrant of a Cartesian plane.
- The end points are defined by integers.
- A line segment is defined by a quadruplet of integers: the first pair is the x and y coordinates of one end point, the second pair is the x and y coordinates of the other point.

The first and third assumptions do not impose any restrictions on the set of line segments that can be drawn, however, the second does. For purposes of illustration, this restriction is acceptable. Also note that this grammar is redundant - there are two ways to express the same line segment. Rules can be built into the grammar to eliminate this redundancy. However, the added complexity will obfuscate the salient features of the grammar, so these additional rules will not be incorporated. Assumptions stated, the grammar is:

picture →	**{***line_segments***}**
line_segments →	*line_segments***,** *line_segment* | *line_segment*
line_segment →	**{***number***,** *number***,** *number***,** *number***}**
number →	*number digit* | *digit*
digit →	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

Table 1: Grammar to describe a set of line segments

This grammar has 13 terminals - the ten digits and the two tokens "**{**", "**}**" and "**,**". The simplest collection of line segments, hereafter called a picture, that can be generated is a single line segment. Figure 1 shows a tree representation of the picture **{{**12**,**3**,**5**,**67**}}**.

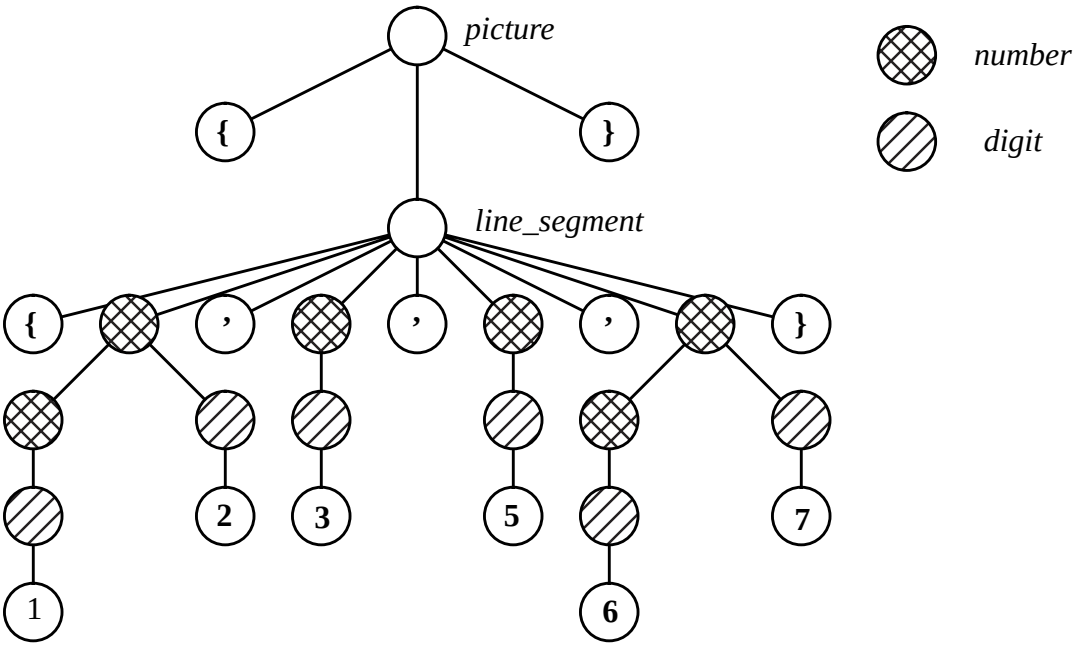


Figure 1. Tree diagram of simple line segment

In the diagram, tokens are shown in bold, terminals in normal font and non-terminals in italics. To simplify the diagram, the non-terminals *number* and *digit* are shown as shaded circles instead of writing the name of the non-terminal next to the circle.

This simple example highlights two important ideas. First, a grammar itself is not sufficient for describing anything — it must be coupled with a means of interpretation. In the preceding example, there is nothing that indicates how to interpret the sentence **{{**12**,**3**,**5**,**67**}}**. The designer of the grammar and language interpreters must share a common understanding. Second, grammars can be recursive, but they must also terminate.

The extension of the context-free grammar concept to describe physical artifacts is discussed in Section 4.2.

3 Introduction to genetic algorithms and genetic programming

➤ Genetic algorithms (GA) are search algorithms that use the notions of natural selection and genetics. The basic processes used are survival of the fittest, with information exchange among the survivors. Like biological systems, there is some randomness to this process, but instead of causing detrimental affects, this randomness gives GA robustness and the ability to generate better solutions.

Genetic programming (GP), like GA, uses the notion of natural selection to generate "computer" programs that solve problems. The word computer is quoted because although the theory underlying GP was developed for computer programs, in this thesis, the representational scheme developed for GPs are used to generate artifacts.

It is important to address the question: 'Why not use traditional optimization techniques?" The answer to this question is twofold. First, most traditional techniques require continuous functions and their derivatives[1]. Mathematical function cannot be defined for many design problem, thus traditional methods cannot be easily applied. Second, GP, used with an appropriate problem representation, can be used to generate novel results. A traditional optimization technique can only find an optimal value for the given function — GP modifies the function to find the optimal value.

One important fact to note is that GA and GP use random numbers, as do simulated annealing techniques. However, unlike simulated annealing techniques, GA and GP are not directionless. GA and GP techniques make use of past events to guide future events. These techniques yield continually improving performance of the functions being sought.

This chapter is not intended to present the theory of GA and GP, but rather to give the reader a familiarity with these techniques. The rest of this chapter will discuss the mechanics of GA and GP with some simple examples. Numerous books exist that go into detail of the completeness and correctness of GA and GP and the interested reader is referred to these for further details [34] [52]. Section 3.1 describes the mechanics of GA and provides a simple example. Section 3.2 describes the mechanics of genetic programming and provides a simple example. Section 3.3 introduces the concept of strongly-typed GP and also presents a method of hybrid GA/GP. Section 3.4 suggests a means for selecting appropriate control parameters for GA/GP problems.

1. This is not to imply that genetic techniques are the only optimization routines that do not require continuous functions and their derivatives. Other techniques, such as MINLP, also do not require this additional information.

3.1 Genetic algorithms

Genetic algorithms differ from traditional optimization techniques in four fundamental ways [34]:

- GA work with a coding of the parameters, not the parameters themselves
- GA use a population of samples, not a single sample
- GA use payoff information, not auxiliary information or derivatives
- GA use probabilistic, not deterministic, transition rules

The key to the successful implementation of a GA is the parameter coding. (The reason for this has to do with schemata theory, the underlying theory of GA that explains why they work. How they work is through fitness proportionate reproduction, which has been shown mathematically to be near optimal in some senses.)

Once the coding is done, the solution of an optimization problem using GA proceeds in the following manner, see Figure 2:

```
procedure geneticAlgorithm
begin
        initialize Population (t=0)
        evaluate Individuals in Population (t)
        while terminationConditions not satisfied, do
        begin
                t = t + 1
                select Population (t) from Population (t-1)
                recombine Individuals in Population (t)
                evaluate Individuals in Population (t)
        end
end
```

Figure 2. Pseudo-code showing genetic algorithm procedure

To initialize the procedure, the population is created by randomly selecting members from the allowable space based on the method of parameter coding used. Then, each of the members in the population are evaluated. Next, an iterative loop is entered in which some members of the population are selected for the next generation based on their fitness. These members are recombined to form the population of the next generation, and finally the members of the new generation are evaluated. Each step of the algorithm will be discussed and an illustrative example will be given.

The most common type of GA used are bit-string genetic algorithms. These GA represent the parameters by a binary string. Non-binary representations are possible but tend to be more complex without yielding any benefit. Binary numbers can be used to represent arbitrarily large integral values. Non-integral values can be represented by operating on

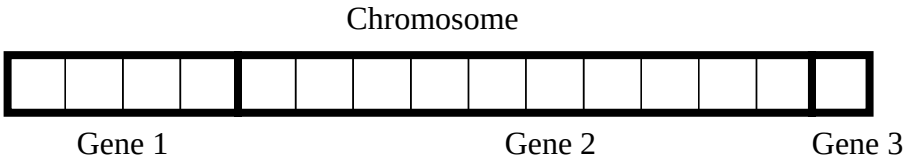
integral values. For example, real numbers, *r,* from 0.0 to 10.0 can be represented by a 10 bit binary string and

$$r = \text{string} / 1024 \tag{1}$$

where string is the decimal equivalent of the 10 bit binary string. It is important to note that this method of representation is non-continuous, and it is possible that the optimal answer cannot be exactly expressed using this method — irrational numbers, for example. However, in practice, the required accuracy of the answer is known and a properly conceived parameter representation will yield answers of sufficient accuracy.

In [44], the authors present five classes of problems and show the ability of GA to solve these classes of problems. Of the five classes of problems, only one problem, designated as *F5* in the paper, was shown to be particularly sensitive to parameter selection (see Section 3.4). Since *F5* can be visualized as a bed of nails it is expected that most design problems will not fall into this class, thus genetic methods should work well in this problem domain.

Bit strings can be used to encode more than one number. They can incorporate any number of parameters that are necessary for the problem. In this, bit strings are analogous to chromosomes. A group of bits that defines a feature is a gene and the values taken on by the bits are alleles. The following example should help clarify the terminology:



Gene	What it represents	Range of values	Conversion function
1	Number of sides	[0 .. 15]	Binary representation
2	Length of side	[0.0 .. 2.0) inches	length = Gene 2 / 512
3	Solid or wire frame	0 or 1	0-solid, 1-wire frame

Examples:

Chromosome = 001110000000001

Chromosome = 010001000000000

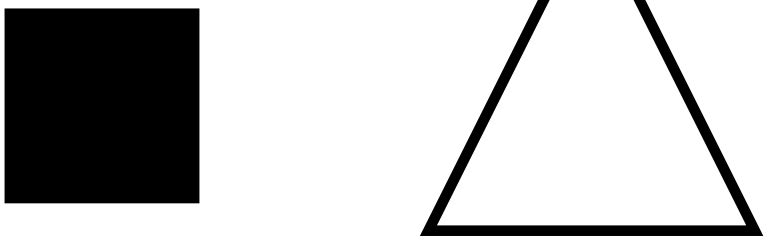


Figure 3. Chromosome to describe a polygon

The first step in the genetic algorithm is to determine an appropriate population size and then to create an initial population. Population size can be determined by applying theoretical results [35] or can be determined empirically. Typically, a larger population size will yield better results, however, more computation time is required. The composition of the initial population is determined randomly. For the example shown in Figure 3, random numbers drawn uniformly in the range [0 .. 32,768] will provide an appropriate initial population.

Along with the parameter coding, the evaluation of the individual members of the population is where the implementor of the GA can really impact the results obtained. Typically, the evaluation function is straight-forward. For the example in Figure 3, if the goal is to design a large reflector, the evaluation function might be the shaded area of the polygon. The value returned by the evaluation function for an individual is referred to as its fitness. The evaluation function must meet three general rules:

1. Fitness values should be non-negative.
2. Better individuals should have larger fitness scores.
3. Evaluation functions should execute quickly

Rule 1 is needed because of the selection process. Fitness values can be scaled to make negative values non-negative or all negative values can simply be set to zero. Rule 2 is needed to promote selection of fitter individuals for reproduction. The selection process will be shown to be a function of the individuals' fitnesses, thus the larger the fitness, i.e., the better the individual, the greater is its probability of reproducing. Finally, Rule 3 is needed for pragmatic reasons, since the evaluation function will be called repeatedly.

To improve the results of GA, fitness scaling is frequently used. Fitness scaling expands or contracts the range of fitness values to fit some predetermined range. There are numerous schemes for fitness scaling, one of the most common is linear scaling. Using this technique, the maximum fitness is defined to be n times the average fitness, and all other fitnesses are scaled accordingly. A typical value for n is 2.0. Fitness scaling has the dual advantage of preserving genetic diversity in both the early generations by preventing a small number of chromosomes from dominating the pool of surviving individuals, and in the later generations by preferentially selecting marginally better individuals.

The requirements for the evaluation function are not very different from the requirements for evaluation functions for other types of optimization algorithms. Where GA radically depart from other algorithms is how they generate the next points to be evaluated. Calculus based optimization techniques use knowledge of the functions derivatives to "hill climb" towards an optimal result. GA use the "random" contributions of successful individuals in one generation to produce individuals for the next generation. The first step of this process is selecting those members to be used to produce the next generation.

There are a number of selection algorithms commonly used. The most basic selection algorithm is stochastic sampling with replacement. To visualize this scheme, imagine a weighted roulette wheel that is partitioned according to fitness of the individuals, Figure 4.

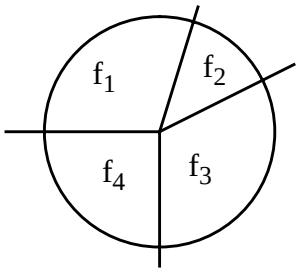


Figure 4. Weighted roulette wheel

"Spinning" this wheel will yield, probabilistically, a higher percentage of those individuals with higher fitnesses and a lower percentage of those with lower fitnesses. The problem with this method is that it is too random, and non-representative populations are frequently observed. A scheme that yields better results is called remainder stochastic sampling without replacement [34]. The individuals comprising the next generation are found by first calculating the expected number of copies that the individuals in the current generation are expected to contribute. Then, the integral parts of the expected contributions are assigned and the rest of the succeeding generation is found probabilistically using the remaining fractional parts.

Once the members of the next generation have been selected, a three step process is used to modify those individuals. The first step determines if a pair of individuals selected for reproduction will be crossed or not. This determination is done probabilistically. Experiments indicate that a cross-over probability of 60% yields good results. The second step, for those members that are to be crossed, is to select a cross-over point and to perform the cross-over operation. There are three types of crossover operations typically used, one-point, two-point and uniform [44]. (Throughout this work, only one-point cross-over is used.) For all three cross-over algorithms, the cross-over point(s) are chosen randomly. Figure 5 uses the chromosomes from the example in Figure 3 to illustrate one-point crossover. The results of this example are a 2.0", wire-frame pentagon and a 1.0", solid line. The final step is a bit-wise mutation. The user defines the mutation probability, usually on the order of 0.1%. Mutation is not the dominant force in a GA, but it is useful in restoring bits that may have been removed from a population at an earlier time.

Cross-over site: 3

Parents	Children
010|001000000000	010|110000000001
001|110000000001	001|001000000000

Figure 5. Cross-over example

This process, while seemingly complex from the description, is actually quite simple. To clarify the details, the following example is used. Consider the problem of forming the largest volume box from a flat piece of material of given dimensions, Figure 6.

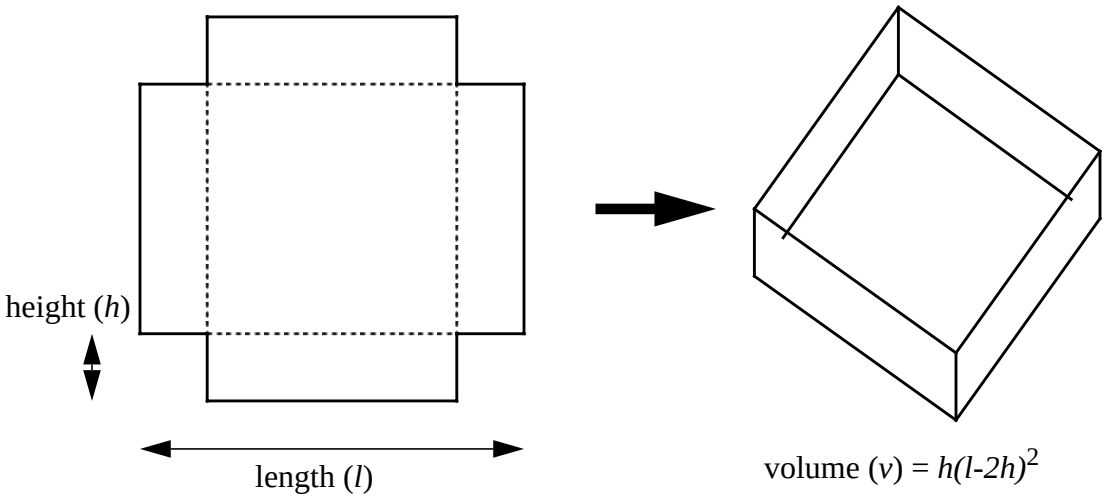


Figure 6. GA example problem

Since l is given, this problem can be solved in closed form using calculus:

$$\frac{dV}{dh} = 0 = 12h^2 - 8lh + l^2 \qquad h = \frac{l}{6} \tag{2}$$

Notice that this solution required the use of the derivative of the objective function.

The first step in solving the problem using GA is to encode the parameters. For this problem, the ratio of h/l will be chosen as the parameter to be coded. This parameter's range is restricted to lie on the range [0.0 .. 0.5] (from Figure 6 it is clear that a value of $h > l/2$ is meaningless) and is encoded with a five bit integer: 0.0 = 00000 and 1.0 = 11111. The formula to convert from the encoded form to the actual variable is

$$\frac{h}{l} = \frac{I/2}{31} \tag{3}$$

where I is the encoded form of the variable. The objective function is rewritten as

$$vl^3 = \frac{h}{l}\left(1 - 2\frac{h}{l}\right)^2 \tag{4}$$

to accommodate the change on the parameter. Without loss of generality, by setting $l = 1$, the objective function becomes

$$v = h(1-2h)^2. \tag{5}$$

(Substituting Equation (2) into Equation (5) yields a maximum volume of 0.074.) With the preliminaries taken care of, the optimization can proceed.

To simplify the example, a population size of four individuals will be used. A cross-over probability of 1.0 and a mutation probability of 0.0 will be used. Remainder stochastic sampling without replacement will be used for selection. Table 2 shows the results after the first generation. The population shown for this generation was created randomly:

Num	Chromo-some	h	Fitness $v = h(1-2h)^2$	Prob select $\frac{f_i}{\sum f_i}$	expected $\frac{f_i}{\bar{f}}$	Actual count
1	`01101`	0.210	0.0707	0.395	1.580	2
2	`11000`	0.387	0.0197	0.109	0.437	0
3	`00100`	0.065	0.0489	0.275	1.098	1
4	`00011`	0.048	0.0395	0.221	0.885	1
	sum		0.1788	1.00	4.00	4
	average		0.0447	0.25	1.00	

Table 2: First generation statistics

The next step is to form the next generation. Table 3 shows the results of the mating and the calculated fitnesses of the individuals in the next generation:

Chromosome (showing cross-over site)	new chromosomes	h	Fitness $v = h(1-h)^2$
`01|101`	`01011`	0.178	0.0704
`00|011`	`00101`	0.081	0.0565
`001|00`	`00101`	0.081	0.0565
`011|01`	`01100`	0.194	0.0725
sum			0.2595
average			0.0649

Table 3: Results from applying genetic operators to first generation

After one generation, the average fitness has increased by more than 50% and the maximum fitness achieved is within 0.3% of the optimal value.

3.2 Genetic programming

➤ Genetic programming (GP) is a recent extension of genetic algorithm theory [52] [53]. The reason for the development of GP was to help answer the question: Can computers learn to solve problems without being explicitly programmed to do so? GP is a domain-independent

means for automatically generating computer programs that can be used to solve a variety of problems. The key differences between GA and GP are the means of representation and the meaning of the representation. GA are typically represented with bit strings, GP uses rooted trees. In GA, the genome is different from the phenotype, in GP they are the same.

The search space for a bit-string GA is the space spanned by the number of bits in the representation of an individual. For the box example above, the space has 32 valid individuals. In a GP, the search space consists of all possible programs formulated from a set of terminals and functions. Since there are typically a large number of potential terminals and a moderate number of functions, the search space of GP is tremendously large. In practice, the number size of the individuals in a population are constrained not to exceed some maximum size.

The three major steps for running a GP are determining:

1. a set of terminals
2. a set of primitive functions
3. the fitness measure.

Each of these three steps is highly dependent on the problem for which a solution is being sought. For example, if the problem is to determine a function that best fits a set of points, the terminal set might include the value of the independent variable and the set of real numbers. The primitive functions chosen might include the set of arithmetic operators and some transcendental functions. It is important that the function and terminal sets have the closure property. That is, each of the primitive function should be able to accept any terminal or the output from any function as inputs. The closure requirements can be relaxed by introducing the concept of strongly-typed genetic programming (STGP), see Section 3.3.

An intuitive way to represent an individual in a population is by showing it as a rooted, point-labeled tree. Figure 7 shows two such trees. The tree on the left represents the mathematical function $X + (0.75X + Y)Z$ and the one on the right represents $(1.2 + X)\sin Z$. Internal nodes represent functions, external nodes represent terminals.

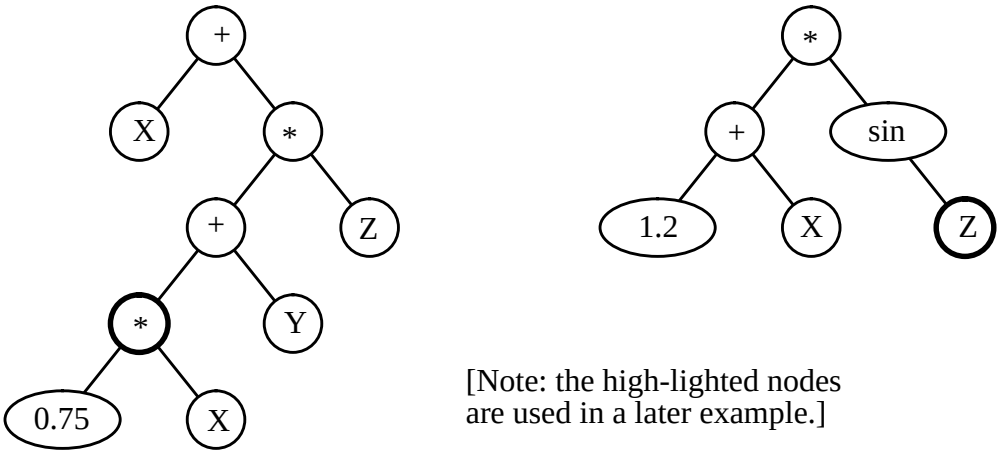


Figure 7. Example programs represented as rooted trees

An alternate means of representation is to use a prefix notation, like that used by the `LISP` programming language. The above examples are represented in using this notation as:

```
(+ x (* (+ (* 0.75 X) Y) Z))          (* (+ 1.2 X) sin (Z))
```

Figure 8. Example programs represented with prefix notation

Once the representation is decided upon, the GP procedure proceeds in a manner similar to the GA procedure, as shown in Figure 2. The only remaining difference is the means with which the individuals are evaluated. For a GA, the chromosome of the individual is usually a representation of some item, and the evaluation procedure first “translates” the representation into a usable form, then calculates the fitness of the individual. In a GP, however, the chromosome is the program and it is executed directly. Since all programs are assumed to be syntactically correct (this can be enforced, see the discussion of STGP, Section 3.3), successful evaluation of the program is not sufficient. What is typically done is to compare the results obtained from the GP generated functions to the actual results and assessing the fitness as a measure of how close the function comes to the actual results.

For previously mentioned example, determining a function to fit a set of data points, a typical means of evaluation would be to substitute each of the values of the independent variables into the GP generated function, and to measure the fitness as the sum of the absolute values of the differences between the dependent variables and the values produced by the functions.[2]

---

2. This method of evaluation requires knowledge of the answer *before* the GP is applied, thus the implementor already has a preconceived notion of the desired results. This raises the question of the ability of systems using this evaluation method to provide truly novel concepts, as any concepts developed by these systems must be favorably evaluated. Since GP are generally used to create programs to solve a specific problem, the idea of a “more-is-better” evaluation function does not yet appear in the literature. To generate new artifacts, however, different means of evaluation will need to be developed.

GP crossover is performed by selecting nodes in each tree and swapping them. With GP, there are no restrictions on the nodes that can be crossed - any node in one tree can cross with any node in another tree. Using the examples above, and preforming the crossover operation in the two nodes that have darker borders, yields:

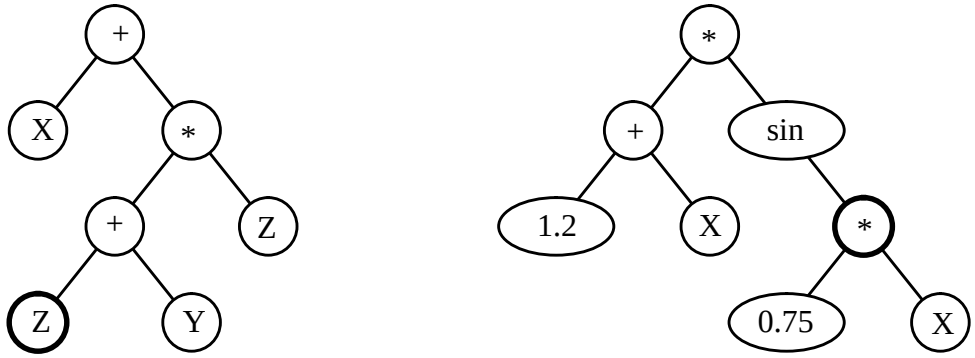


Figure 9. Crossover example

There are two important features to notice: first, both resulting trees are syntactically correct; and second, the “size” and “shape” of both trees have been altered. Unlike GA, which produce children that are the same as the parent when both parents are identical, in GP the crossover of two identical parents will typically not yield identical children.

A more interesting problem is how to handle mutation in GPs. The “traditional” method of GP mutation, creating a random tree and crossing it with the tree to be mutated at the selected node, does not typically yield meaningful results. (As was asked of the author by a GP researcher, “What is the probability that a subroutine from one of your programs would function in some other program?” This leads to the idea that GP tends to produce programs that are robust to genetic manipulation, similar in a sense to biological genomes that produce viable individuals even though parts of the DNA sequence might be corrupt.) A (possibly) better form of mutation is to add or delete a sub-tree at the node selected for mutation. This tends to preserve the overall structure of the tree, yet permits morphological changes to occur. This is the method of mutation used for the GD methodology.

One concern for GP researchers is the efficiency of evaluating a tree. Consider the following C-code fragments to see how this is done, Figure 10, based on Keith and Martin, [48]. The first code fragment shows a node structure and the second shows how the tree is evaluated.

```
struct node {
    enum {FUNCTION, VARIABLE_OR_CONSTANT} Type;
    unsigned char Arity;
    RETURN_TYPE Value;
    struct node *Children[MAX_CHILDREN];
    RETURN_TYPE (*EvaluationFunction)();
};

RETURN_TYPE EvaluateTree (struct node *N)
{
    register int i;
    RETURN_TYPE ChildrensValues[MAX_CHILDREN];

    if (n->type == VARIABLE_OR_CONSTANT) {
        return (N->Value);
    } else {
        for (i=0; i<Arity; i++) {
            ChildrensValues[i] = Eval (N->Children);
        }
        return  (*N->EvaluationFunction)(N->ChildrensValues);
    }
}
```

Figure 10. GP nodal representation and tree evaluation

In this example, `Arity` is the number of inputs to the particular function, `RETURN_TYPE` is the type of argument of the node (and for that matter, all nodes in the tree), `MAX_CHILDREN` is the maximum number of inputs any function can have, `Children` are pointers to the nodes rooted at the current node and `EvaluationFunction` is the function implemented by the node. The array `ChildrensValues` holds the values returned by evaluating each of the node's children. Using the highlighted node from the left-hand tree in Figure 7 as an example:

```
struct node {
    Type = FUNCTION;
    Arity = 2;
    Value; /* not used for a function node */
    *Children[MAX_CHILDREN] = {pointer to Node 0.75, pointer to Node X};
    (*EvaluationFunction)() = Multiplication ();
};

RETURN_TYPE Multiplication (RETURN_TYPE *Operands)
{
    return (Operands[1] * Operands[2]);
}
```

Figure 11. Tree evaluation - example

Since traversing the tree takes on the same order of time as evaluating the primitive functions, efficient representations are a key area of research in the GP community. As will be shown later, Chapter 4, the amount of time required to evaluate the artifacts designed is

quite large. Thus, optimizing the tree parsing routines will have little affect on the overall run-time of the program. Since trees are easy to understand and easy to implement they will be used because their time and memory inefficiencies are insignificant in the scope of the current work.

➤ The final issue to be examined are the apparent differences between GA and GP. Since most of the theoretical work to date has been done for GA, if it can be shown that GP is a special case of GA, then the important theoretical results, such as proofs of convergence, can be said to hold for GP. Since GP is implemented on a computer, the representation of the program is in fact a bit string. In fact, since GPs represent programs, they can be coded as Turing machines, which are bit strings. To make GP more efficient, we impose our beliefs about the problem, i.e., the function and terminal sets, instead of starting with the generality of a Turing machine. This is made apparent by the prefix notation representation, Figure 8. The differences between GA and GP are therefore not in the underlying representation, but rather in the genetic manipulation.

The basic premise of GP is that the developer of a GP has some domain specific knowledge that can be applied to guarantee that genetic operations take place at predetermined points on the bit string, points specifically chosen to maintain syntactic correctness of the sub-strings that are manipulated. Figure 7 and Figure 9 are an example of how this syntactic correctness is maintained during the cross-over operation. By restricting the loci at which genetic operations can occur, a large set of implausible solutions are eliminated from consideration. For GP and Genetic Design (GD), this restriction is important because both the programs and artifacts function in a syntactic setting; thus maintaining syntactic correctness is crucial.

3.3 Hybrid GA/GP

➤ This section introduces two concepts: strongly-typed GP (STGP) and a GA-type means of modifying numerical constants in a GP. The first concept, the idea of STGP is to allow different types of structures within the same tree [70]. One example for the need for this is to be able to generate a program that uses both scalars and vectors. The purpose of the second concept is to be able to modify the constants in a GP in a more direct manner. If the terminal set of a program tree (which is generated randomly) does not contain a specific numeric value, a program fragment that operates on numeric values is needed to produce that value. The following simple program fragment demonstrates how to generate the value 2.0 given the value 2.4:

`(/ (+ 2.4 2.4) 2.4)`.

Thus, the complexity of a program can increase tremendously just to get good numeric values.

Using the GP methodology set out in the previous section, it is not possible to have a program that handles both scalars and vectors because of the underlying assumption of

closure. To get around this, STGP requires that the output(s) from each node be given a type and that the inputs of each node take certain types. Remaining with the vector math example, and limiting the discussion, without loss of generality, to scalars, vectors of length two and operations with two inputs and one output only, several different types of nodes can be defined, for example:

symbol	type	input 1 type	input 2 type	output type
SS	scalar operations	scalar	scalar	scalar
VS	vector scaling	scalar	vector	vector
VD	vector dot product	vector	vector	scalar
VM	vector cross product	vector	vector	vector

Table 4: STGP typing example

To perform a cross-over operation between two trees using STGP, only the outputs of the nodes need to be the same. This example is somewhat uninteresting because several of these types have only one possible member in the set of that type. For example, there is only one recognized means of taking the cross product of two vector whereas there are numerous scalar operations, such as addition, subtraction, etc.

The implementational differences between GP and STGP is that STGP crossovers must occur between nodes of the same type and that the generation of the initial population must adhere to the type restrictions. (In a standard GP, all nodes are of the same type by default.) This is implemented by randomly selecting a node from one tree and then randomly selecting nodes from the second tree until it is of the same type as the first node selected. One possible problem with this method is that it is asymmetric since the distribution of types will typically be different between different trees. For example, if member A has many nodes that output scalars and member B has many nodes that output vectors, the probability of selecting a scalar or vector node for cross-over depends on the tree initially selected. This biasing is an area of on-going research. Syntactic validity is guaranteed by generating an initial population of syntactically correct trees. Chapter 4 will extend the concept of STGP to encompass functional components of physical artifacts.

As shown in the small example above, GPs can produce very convoluted trees to express simple numeric constants. It is proposed that an alternate method of generating constants is to select two constants from the parent trees and use GA-type crossover on them[3]. (Another method is to form a bit string comprised of all of the constants in both trees and cross-over these long strings. However, this method causes too much rearrangement of the constants within the tree and tends to slow down, and even halt, improvement of the individuals.)

3. The author and Howard [41] appear to have been working on this idea at the same time and plan to publish a co-authored paper detailing the method.

If this method is to be used, it must be used in conjunction with STGP. Each node in the tree must be one of two types - constant or terminal/primitive function. Unlike the work done previously, primitive functions and constants are of the same type because they can be freely interchanged within a GP. Of course, only constants of the same data type can be crossed.

The advantages of this method are that constants are expressed directly without the need to evaluate a tree structure to determine their values and the program trees can be effectively deeper since space is not being used for operations on constants. This speeds up the execution of the program and reduces the memory requirements. The disadvantages are the need to predetermine the range of the constants and the reduced resolution of the constants[4]. Although setting a range for the constants restricts the solution space, the range can be determined by observing the results. Should some constants be at the boundary, the range can be reset and the problem rerun. The achievable resolution of the constants may be less of a problem. For example, if 8 bits are used to represent a constant, the granularity of the constants is limited the range of values divided by 255. Using more bits reduces the granularity and a resolution of 32 or 64 bits may provide sufficient granularity for most applications.

To implement this GA-type constant crossover method, two crossover probabilities are chosen by the user. The first is the probability that a GP-type crossover will occur and the second is the probability that a GA-type crossover will occur. These probabilities must sum to a value less than or equal to one. During the crossover phase, a pair of adults can undergo GP-type crossover or GA-type crossover or not be crossed at all.

Similarly, two mutation probabilities are selected by the user. These are the probability that a node will mutate and the probability that a constant will mutate. For each program tree, first all nodes are checked to see if they mutate. Once all nodes have been mutated, all of the constants are checked for bit mutations.

3.4 Parameter selection

Once a GP is designed and ready to be run, the implementor is faced with the task of selecting appropriate control parameters. Koza lists thirteen numerical and six qualitative parameters that are selected in order to run a basic GP. Needless to say, these parameters are not completely independent in their net affect on the resulting programs. Thus, a proper choice of these parameters is necessary to maximize the probability that the GP will produce good results.

Koza does not directly address the question of selecting an “optimal” parameter set, see [52], page 114. Rather, in the examples presented, large populations of programs are tested for a relatively few number of generations and the implementor monitors the results for a program that performs well. This method of using very large initial populations seems

4. The reason that a range needs to be pre-selected is that genetic operations on floating point numbers do not work as expected because of the way that computers represent floating point numbers. Thus, floating point numbers are typically represented by applying some simple formula to an integer.

to run contrary to the principle of genetic approaches because good results occur if a good, random initial individual is created, thus minimizing the impact of the genetic manipulations. In theory, optimal result should be obtainable by starting from a population in which all members are identical.

One way to chose good parameters is to construct a GA program with individuals whose chromosomes are the parameters to be passed to the GP. The evaluation function of the GA becomes the results from the GP. In essence, this GA is a meta-GA. The natural question is how to chose the parameters for the GA? GA problems are typically easier to understand than GP problems and there exists a deeper experience pool, so, for a GA, experience can help to select appropriate values. Of course, one could construct a meta-meta-GA, a meta-meta-meta-GA, etc., but the return on investment will probably be quite low because of the tremendous computing resources required. (The idea of using a meta-GA is not a unique concept - the author learned of another researcher using meta-GA in his work, however, this other work has not yet been published.)

4 Genetic Design

➤ The underlying purpose of this thesis work is to develop a methodology that aids an engineer in the design of an artifact. To be useful, this methodology should present the engineer with a set of good alternative design configurations. The generation options and evaluation of these design alternatives must be carried out automatically. To achieve these goals, this methodology combines formal grammars for artifact description and representation, automatic artifact evaluation and genetic programming-like representation manipulations.

➤ The previous two chapters have introduced two enabling technologies for the genetic design methodology: formal grammars and genetic computation. Formal grammars are used to represent artifacts because they have the properties of being able to generate a wide variety of artifacts which can be easily manipulated by computer programs. GA and GP methods are used to direct the design of the artifacts as well as generating new artifacts.

This chapter will show how these components, and artifact evaluation, fit together and will discuss some of the underlying assumptions and constraints of GD. Chapter 5 presents an extended example showing the application of this methodology.

4.1 Genetic design implementation and application

The concept upon which GD is built is that systems to be designed are syntactically equivalent to computer programs. In a computer program, data are operated on by functions to produce results. The structure of a program is rigorously constrained by the definition of the grammar that describes the language. In a mechanical device[1], physical objects are acted on by forces to produce results. With little loss of generality, the structure of certain classes of physical objects can be constrained by a grammar that describes its composition.

This analogy is not as far-fetched as it might seems at first. A black-box description of a computer function and a mechanical device are essentially equivalent, see Figure 12. The left hand side of the figure shows a computer function operating on some data and producing a result. The right hand side shows mechanical device operating on some inputs and producing an output. As is clear from the diagram, a suitable mapping should allow representation of either of these systems in either domain, or perhaps a third, different domain. That this transformation is possible is evident by the fact that computer programs can be written that simulate mechanical systems and that mechanical computers can be built to run programs.

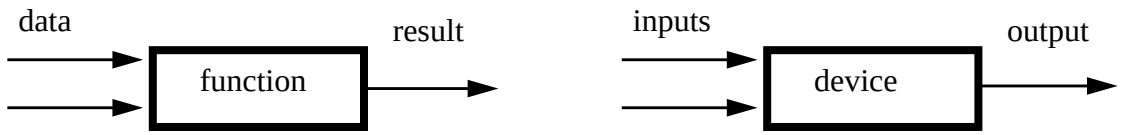


Figure 12. Black-box representation of a computer function and a mechanical device

1. Other types of real-world systems also exhibit the same syntactic equivalence, e.g., electrical circuits.

Why is the ability to map between these domains important? As mentioned in Chapter 2, all computer programs are based on formal grammars. Since sentences constructed from formal grammars can be represented as rooted trees, Figure 1 and Figure 7, this suggests that an artifact representation that is constructed from a formal grammar can be manipulated in much the same manner as a computer program is manipulated using GP.

The process of designing a system using GD is similar to the process of designing a program using GP. The key difference is that with GD the artifact is *represented* by a rooted tree whereas with GP, the artifact, a computer program, *is* the rooted tree. Figure 13, a copy of Figure 2, shows the process of design using the GD methodology.

```
process geneticDesign
begin
        initialize Population (t=0)
        evaluate Individuals in Population (t)
        while terminationConditions not satisfied, do
        begin
                t = t + 1
                select Population (t) from Population (t-1)
                recombine Individuals in Population (t)
                evaluate Individuals in Population (t)
        end
end
```

Figure 13. Psuedo-code showing genetic design process

Thus, GD is essentially a generalization of GA that uses the representation and genetic manipulations of GP. Like a GA, the chromosome is not the artifact itself, but rather an encoding of it. Like a GP, the artifact is described using a tree representation and the genetic operators operate both on the structure of the tree and the numeric values found at the leaf nodes.

In their work, Pahl and Beitz [73] introduce a systematic approach to engineering design. This approach divides the process of engineering design into four phases: clarification of the task, conceptual design, embodiment design and detail design. It is believed that GD can be applied to each phases of the four-phase model of design, except the clarification of task phase. The stepping stone walker, Chapter 5, is an example of conceptual design. Mechanism synthesis, Chapter 6, is an example of embodiment design. No examples of detailed design are presented in this thesis.

For the more abstract phases, the grammar needed to describe the model can generally be constructed more easily, however, evaluating the performance is more difficult. For the more concrete phases, grammar construction is more difficult, but evaluation is simpler. The reasons for this is that grammars for the more abstract concepts do not require the complexity of a grammar for a detailed part description, but the performance criteria are more

numerous and harder to evaluate. Consider for example the differences between a automobile and an automobile transmission. The former is relatively easy to describe, but must satisfy numerous, conflicting requirements. The latter is relatively difficult to describe with a grammar, but need only satisfy a fewer, more easily quantifiable requirements.

➤ Before addressing some of the implementational details of GD, the reasons for using this methodology are discussed. As shown in Section 1.2.3, there already exists a number of design methodologies - why should this new technique be considered? Genetic design

- can explore a wide (grammar-defined) space of design alternatives. This is a marked advantage over those methodologies that do not incorporate grammars because with those methodologies, the designs considered are limited to only those conceived by the designer.
- is essentially domain independent. Although each design domain requires a grammar and a means of evaluation, the underlying genetic operations remain unchanged. One can envision a design environment (see Chapter 7) in which the designer defines the grammar and evaluation functions and the rest of the design process is carried out automatically. Since GD does not require an integral evaluation routine, existing evaluators can be used. To use a non-integral evaluator, the designer is only required to write a program to translate between the GD representation and the representation required by the evaluator.
- does not require continuous functions, derivatives, etc. Traditional optimization functions are unsuitable to the task of artifact design because the evaluation functions are typically non-linear, and possibly non-continuous, making the application of traditional techniques very difficult.
- carries out synthesis and analysis automatically. Once the grammar and evaluation routines are developed GD will produce a number of viable alternatives for the designer to consider. Since the production takes place automatically, the designer is freed to carry out other tasks.
- performs in a “reasonable” time. Since GD is based on GA, the members of the population will successively improve with each generation. Thus, the “quality” of the answer generated by GD will be limited to the amount of computing resources available. Furthermore, since the calculation time is strongly dominated by the evaluation function (some tests indicate that the evaluation consumes in excess of 90% of the computing time), the time required for each generation can be estimated.
- uses a general grammar, a super set of other grammars, such as shape grammars, that is capable of defining higher level structures. This represents a potential improvement over other grammar based systems because they cannot handle these higher order structures, such as a controller (see Section 4.3). It is this capability that is the distinguishing feature of GD that sets it apart from most other design methodologies.

Although GD does possess some strong attributes, there are some limitations. The basic limitation of GD is described in Section 3.2, where the limitations of the tree representation are discussed. The other limitation of GD is that the technologies applied to the design of real artifacts also undergo an evolutionary process. That is to say that the highly evolved designs that are a part of our daily lives could not have been initially designed in their current form because the technologies, both hardware and knowledge, used in their highly evolved form simply did not exist at the time of their initial design. Consider video tape recorders for example. The highly evolved VCR's of today are functionally equivalent to the units of two decades ago, however they are but a small fraction of the size and weight, consume less power and cost less. Although there have been improvements in the fundamental technologies of video recording, the increased understanding of these systems, that can only come from experience with the actual systems, may have actually been the greater contributor to the change in VCR design.

4.2 Artifact representation

The GD methodology, as currently implemented, requires artifacts to be represented by a formal grammar[2]. Although the current work is restricted to context-free grammars, other more descriptive grammars can be used, however, they are more difficult to work with. In a computer language, the terminals are predefined mathematical functions or numbers. Keywords are used to define the flow of the program. The distinguishing feature of computer languages is not the data they operate on, but rather the grammar that is used to define the language. Similarly, to describe an artifact, the terminals are used to describe those items that the designer has control over. The keywords are used to describe the ways in which the terminals can be connected. The distinguishing feature of artifacts (within the same domain) is not the inputs they receive and the outputs they produce, but the manner in which the elements of the artifacts are connected.

What becomes apparent is that an artifact designer must carefully construct a grammar to encompass knowledge of the artifact being designed. The grammar constructed must be general enough to encompass all possible artifacts within the desired set and should utilize simple elements to permit building of new structures. Consider the example given in Chapter 2. By defining single digits as the terminals and providing a means to build up larger numerical values, the grammar is far more general, and simpler, than one in which all possible numerical values are defined.

The same holds true for defining a grammar that describes artifacts. Consider a grammar to define vehicles. One approach to defining this grammar might start with some functional notion of the type of vehicle, for example walking versus rolling locomotion. It might then consider the number of locomotors, etc. A better approach is to start with more basic

2. Actually, other representations may also work, but grammars were chosen because they provide a great deal of flexibility of representation.

units: joints and links. The descriptive ability of this grammar exceeds that of the previous. In addition, since the units are more basic, there are more loci for cross-over operations between two randomly selected parents.

Thus, all artifacts that can be designed using GD are domain independent in the limited sense that they all share the ability to be described by a formal grammar. Since this work is restricted to context-free grammars, the sentences formed can be represented by rooted trees. Although this particular representation can be used for a wide variety of problems, it also potentially eliminates from consideration other types of problems. The problems that might not be solvable are those problems that are represented by a general graph. Formally, a tree is a structure in which each component has a unique predecessor and a finite number of successors, whereas a graph is a structure in which any component may be related to any other component. An example of a problem that might require a graph representation is the generalized "packaging" problem. This is a problem in which each component affects a number of disparate system parameters. For example, in packaging a camcorder, the components must fit within a specified volume, certain components must be proximate to other specified components, the center of mass must fall within some region, etc. The reason that this class of problems may not be currently solvable using GD is that the technology for manipulating graphs genetically does not exist. If this technology is developed, extending GD to incorporate this new means of representation should be straight-forward. (Also see Section 6.2, in which a method of representing certain classes of network is proposed. This method represents the network indirectly by defining a program that defines the mechanism. If this method can be extended, this inherent limitation might be surmountable.)

To effectively describe artifacts, the GD tree-structure will, in general, contain many different types of nodes, see Section 3.3. To incorporate these types into the GP model, the tree structure previously described, Figure 10, is generalized by expanding the definition of `Type` to encompass all required types. One way to do this is to replace the `RETURN_TYPE Value` with a union of all possible types. This approach may create implementation problems and may be memory inefficient. A more general approach is shown in Figure 14. This approach replaces the value of the node with a pointer to the node's value. The only drawback with this technique is that additional memory management is required to store the values separately. It is important to remember that in GD, the tree is a representation of the object, not the object itself as in GP. Thus, there is no need to evaluate the tree directly as in GP. Instead, the designer writes a routine that interprets the tree structure and translates this representation of the object into a form usable by the evaluation routine, see Section 4.3. Using this approach, the type of argument pointed to by `void *Value` is determined by the `Type` of the node.

```
struct node {
      enum {TYPE1, TYPE2, ..., TYPEn} Type;
      unsigned char Arity;
      void *Value;
      struct node *Children[MAX_CHILDREN];
};
```

Figure 14. GD nodal representation

For certain classes of artifacts, a grammar that describes only the appearance of the artifact provides sufficient information for evaluating that artifact. One example of an artifact of this type is a truss. Other classes of artifacts require information in addition to the appearance of the artifact for evaluation. For example, a grammar that describes a walking vehicle should also provide the information necessary to describe how the vehicle moves. (The reason for this need is described in Section 4.3. Also see [9], in which the authors present an analytic model of the benefits accrued from modifying the plant design at the same time as the controller is designed.) The grammar might also incorporate the concept of sensors into the artifact configuration.

Incorporating control features into the model has tremendous impact on the evaluation of the model. For systems without control features, that is, systems with no degrees of freedom, such as a truss, only a single configuration of the system needs to be analyzed. For systems that incorporate control features, however, there is a range of configurations that must be analyzed. A vehicle, for example, might require evaluations at a number of different configurations. In general, we propose that the number of evaluations, *n*, for a system with $c > 0$ control features is given by

$$n = \aleph_1 . \tag{6}$$

In practice, this infinite number of configurations would be reduced to some finite number of representative states.

4.3 Objective functions

Like a bit-string representation in a GA, the rooted-tree representation used by GD is not evaluated directly. To clarify, consider the example shown in Figure 3, page 27. To operate on the bit-string directly is meaningless since the bit-string encodes three different concepts. This is the exact opposite of GP in which the genetic representation and the object being represented are one and the same. In GD, since the rooted tree is a representation of an artifact and must be “translated” into a more useful form before being evaluated, there are no restrictions on the form of the evaluation process. That is, the evaluation process can be integral to the GD program or it can be a stand alone program that is called by the GD process.

When working with iterative optimization methods, such as genetic computational techniques, emphasis is usually placed on the idea of making the evaluation process execute as quickly as possible because of the large number of times that it is called. While this

is a valid objective, the meaningful evaluation of certain classes of artifacts (those classes that represent practical design problems) just cannot be accomplished quickly using the computing resources currently available. For example, if the artifact being designed is a vehicle and the performance metric is power consumed normalized for vehicle mass and velocity, a dynamic simulation of the vehicle may be required. (Such a simulation does not necessarily have to be a part of the GD program. Rather, the GD program could translate the internal tree representation into the appropriate file format for use by an external program.) Even with current super-computers, such an evaluation will certainly require seconds, if not minutes, of computer time. Since computer capabilities have been doubling every 18 months, this current constraint does not seem to invalidate this work, but to highlight the need for efficient evaluators.

When optimizing a function using GA, the object function is typically the function being optimized. With GP, the objective function is typically how well the generated program fairs compared to some norm. But what is the objective function for an artifact? The answer depends entirely on the problem for which an artifact is being sought. If the problem is to design a mechanism that traces out a specified curve, the objective function might be how closely the generated mechanism follows the specified trajectory. For an extra-terrestrial vehicle, the objective function might be to minimize the power consumed normalized for vehicle mass and velocity. For many real systems, there is no single metric that is useful to defining the fitness of an individual. For the case of the extra-terrestrial vehicle, although minimizing normalized power consumption is important, there are numerous other factors that must be considered and a design based on this single criteria will almost certainly fail to satisfy other mission requirements, see Section 4.4.

➤ Once the objective function, or functions, are identified, the means for evaluating the artifact must be determined. Artifacts can be grossly categorized into one of two classes: deterministic and stochastic. The difference between the two classes is not the artifact itself, but rather how the artifacts are evaluated. Those artifacts classified as deterministic do not change their behavior due to external disturbances while those artifacts classified as stochastic can modify their behavior. Examples of deterministic artifacts include trusses and certain types of kinematic mechanisms (for example, single degree-of-freedom mechanisms). Although in both cases the artifacts might be subjected to time-varying inputs, the artifact cannot change in response to these inputs. Examples of stochastic artifacts include biological and robotic systems. These artifacts typically utilize information about the environment in order to modify their behavior in response to that environment.

There are two ways to evaluate a stochastic artifact: create an *a priori* list of all conceivable responses to a varying environment or incorporate the necessary means to permit the generation of adaptive responses. The first option does not allow the GD methodology to be used to its fullest extent. Since each artifact requires the designer to have developed a list of responses, the method may not produce novel concepts because only certain behav-

iors can be utilized. The incorporation of adaptive responses, hereafter called a controller, into the design of an artifact is the distinguishing feature of a stochastic (or using the terminology from Chapter 1, a complex) artifact.

The purpose of a controller is to permit the automatic evaluation of an artifact in which the input/output relationship is not fixed. An example of this is the sequence of foot placement, the gait, for a walking vehicle. The gait for such a machine is not fixed — it must respond to changes in the terrain. To evaluate such a walking machine, therefore, a generic gait planner is required. However, since the vehicle configuration is not known *a priori*, its gait planner cannot itself be designed *a priori*. Thus, the artifact's controller must be designed simultaneously with the artifact itself. Since the controller can typically be represented as a computer program, it can be generated in the same means as the artifact itself, genetically. For an artifact to perform well, it must therefore have an appropriate configuration as well as an appropriate controller.

The incorporation of an environment-adaptive controller suggests that the artifact needs to model the environment in some manner. This modelling requires the use of sensors. To achieve optimal performance from the artifact, it must have access to a range of potential sensors as well as the ability to integrate them into the model in an optimal manner. Reynolds [82] shows how performance is improved by not only letting the model determine a control algorithm, but by letting it place its sensors in appropriate locations.

When using optimization methods to maximize mathematical functions, it is imperative that the method used converges to the correct result. Holland [39], De Jong [44] and Goldberg [34] all show that GA do indeed converge to the optimal, or near-optimal, result for a wide variety of classes of functions. Unfortunately, there does not appear to exist a similar body of theory that shows that programs generated with GP converge to an optimal solution (however, see the discussion on page 36). However, because of the manner in which the genetic manipulations are carried out, after some number of generations, the programs generated with GP will be better than the initial, random population. Similarly with GD, after some number of generations, the artifacts generated will be better than those initially created. It can be argued that since GP and GD representations can be represented as Turing machines, which are bit-strings, and since convergence proofs exist for GA, then the convergence of GP/GD can also be proven.

The inability to prove convergence to an optimal solution with GD, unlike with function optimization, does not invalidate the method for three reasons. First, in practical design, there rarely, if ever, exists *an* optimal solution. Rather, there exists a set of good alternatives from which the designer chooses. Second, the results obtained from GD strongly reflect the optimization function used by the designer. In practical design problems, the evaluation functions are likely to include terms that are subjective, thus the results reflect this subjectivity. Finally, the output generated by GD is a set of good alternatives, thus the need to prove the optimality of a single design is mitigated.

One trap the designer must be aware of is that most artifacts are used over some range of operating conditions (see [52], page 74). When evaluating the artifact, if only a single operating condition is used, the artifact developed will certainly work well for that particular condition, but may fail miserably in other conditions. To generate artifacts that are robust across a range of operating conditions, the designer should evaluate the artifact across the range of operating conditions. This can be done by either testing each artifact for several different operating conditions or by changing the operating conditions for each generation of artifacts. For an extra-terrestrial, exploration vehicle, the terrain to be traversed can typically be modelled stochastically. Another possibility is to present noisy data to the artifact's sensors [82]. A vehicle design that results from using only a single instantiation of the terrain for evaluation is likely to perform poorly in the actual settings, since it will not be the same as the terrain that was used for testing. However, by generating a number of terrains for testing, there is a greater probability that the vehicle design will succeed. Obviously, if the operating conditions are known precisely, then these known operating conditions should be used for the evaluation.

4.4 Handling multiple objectives

All systems with multiple objectives can be divided into two sets: those in which the various objectives can be combined in some manner to yield a single, representative evaluation and those in which the objectives cannot be combined.

Combining multiple objectives into a single value is typically done by assigning a weighting factor to each of the objectives and summing the result:

$$O = \sum w_i o_i \tag{7}$$

where O is the overall fitness, o_i is the fitness of objective i and w_i is the weight assigned to this objective. This fitness obtained in Equation (7) is sometimes referred to as a single figure of merit (SFM).

There are several problems with using a SFM, including:

- Widely different characteristics, from easily quantifiable parameters like power consumption to subjective assessments of risk, are combined into a single weighted number.
- Assumes the same degree of confidence for all characteristics, which is typically not true. For example, the power utilization for hexapod walking robot can be readily estimated, but the complexity of the planning problem may be hard to estimate *a priori*.
- Choice of scores and weighting factors can be subjective.

What these points really say is that this method can range from subjective to subjective squared. Despite these difficulties, this method is frequently used. One readily accessible source is the journal *Info World*. In each issue, the editors rank certain products using a SFM, see Figure 15. Within the context of a journal, this is a valid approach since the edi-

tors provide the scores and weightings for each product and the products are closely related. A reader interested in the product can re-weight the scores according to his own needs.

(Weightings)	Product A	Product B	Product C	Product D	Product E
Performance					
Installation and configuration (150)	Good (93.75)	Good (93.75)	Satisfactory (75.00)	Good (93.75)	Poor (37.50)
Search and replace (150)	Good (93.75)	Good (93.75)	Good (93.75)	Good (93.75)	Poor (37.50)
Text manipulation (150)	Very Good (112.00)	Good (93.75)	Good (93.75)	Good (93.75)	Satisfactory (75.00)
File manipulation (100)	Very Good (75.00)	Satisfactory (50.00)	Poor (25.00)	Good (62.50)	Satisfactory (50.00)
Programming aids (150)	Very Good (112.50)	Good (93.75)	Good (93.75)	Good (93.75)	Satisfactory (75.00)
Workgroup support (150)	Good (62.50)	Satisfactory (50.00)	Good (62.50)	Good (62.50)	Poor (25.00)
Documentation (50)	Satisfactory (25.00)	Satisfactory (25.00)	Poor (12.50)	Good (31.25)	Poor (12.50)
Support					
Support policies (25)	Very Good (18.75)	Very Good (18.75)	Very Good (18.75)	Very Good (18.75)	Very Good (18.75)
Technical support (75)	Satisfactory (37.50)	Satisfactory (37.50)	Unacceptable (0.00)	Good (46.87)	Good (46.87)
Value (50)	Very Good (37.50)	Satisfactory (25.00)	Satisfactory (25.00)	Very Good (37.50)	Satisfactory (25.00)
Final score	6.6	5.8	5.0	6.3	4.0

GUIDE TO REPORT CARD SCORES

InfoWorld reviews only finished, production versions of products, never beta-test versions.

Products receive ratings ranging from unacceptable to excellent in various categories. Scores are derived by multiplying the weighting (in parentheses) of each criterion by its rating, where:

Excellent = 1.0 – Outstanding in all areas.
Very Good = 0.75 – Meets all essential criteria and offers significant advantages.
Good = 0.625 – Meets essential criteria and includes some special features.
Satisfactory = 0.5 – Meets essential criteria.
Poor = 0.25 – Falls short in essential areas.
Unacceptable or N/A = 0.0 – Fails to meet minimum standards or lacks this feature.

Scores are summed, divided by 100, and rounded down to one decimal place to yield the final score out of a maximum possible score of 10 (plus bonus). Products rated within 0.2 points of one another differ little. Weightings represent average relative importance to *InfoWorld* readers involved in purchasing and using that product category. You can customize the report card to your company's needs by using your own weightings to calculate the final score.

Figure 15. Product evaluation using SFM from *Info World* (reproduced with permission)

This approach does not work well for an automated design tool for two reasons [13] [103]. First, the designer must decide *a priori* how to assign the weights. Second, for certain design problems, the artifacts being evaluated might be quite different from each other, thus raising the issue of the validity of the weighting. For the extra-terrestrial vehicle, both wheeled and legged machines would typically be considered. Since these machines may be vastly different from each other, a simple weighting scheme may do nothing more than reflect the prejudices of the designer. Of course, the first objection can be overcome by hav-

ing the designer interact with the system regularly to update the values, but there is no reason to believe that this would yield objectively superior results.

A second method is to use a rank-based fitness assessment for multiple objectives. This is done by creating, for each individual, a vector of the fitness measures to be considered. With this method, there is no longer a best individual, but rather a set of best individuals. This set is known as the Pareto-optimal set. First defining [27] [22]:

Definition 1 (inferiority)
A vector $\mathbf{u} = (u_1, \ldots, u_n)$ *is said to be inferior to* $\mathbf{v} = (v_1, \ldots, v_n)$ *iff* **v** *is partially less than* **u**, *i.e.,* $\forall i = 1, \ldots, n,\ v_i \leq u_i \quad \wedge \quad \exists i = 1, \ldots, n: v_i \leq u$.

Definition 2 (superiority)
A vector $\mathbf{u} = (u_1, \ldots, u_n)$ *is said to be superior to* $\mathbf{v} = (v_1, \ldots, v_n)$ *iff* **v** *is inferior to* **u**.

Definition 3 (non-inferiority)
Vectors $\mathbf{u} = (u_1, \ldots, u_n)$ *and* $\mathbf{v} = (v_1, \ldots, v_n)$ *are said to be non-inferior to one another iff* **v** *is neither inferior nor superior to* **u**.

Each point in the Pareto-optimal set is a non-inferior solution to the multi-objective problem. For a two objective problem, an example of a Pareto-optimal set is shown in Figure 16. In essence, by creating such a set, the designer is claiming that there is no way to combine the multiple objective functions into a single function and that all points in the Pareto-optimal set are equally acceptable. Some research has been done using of Pareto-optimal sets in conjunction with GA [27]. This method rank orders the solutions, based on their relative superiority to other solutions. For limited-dimension objective space, this method works well. However, if there are a large number of objectives to be met, and the space of artifacts is sufficiently rich, it is not hard to envision the case in which a large percentage of the population falls on the portion of the convex-hull formed by the Pareto-optimal set. Such results would prove of limited value to a designer.

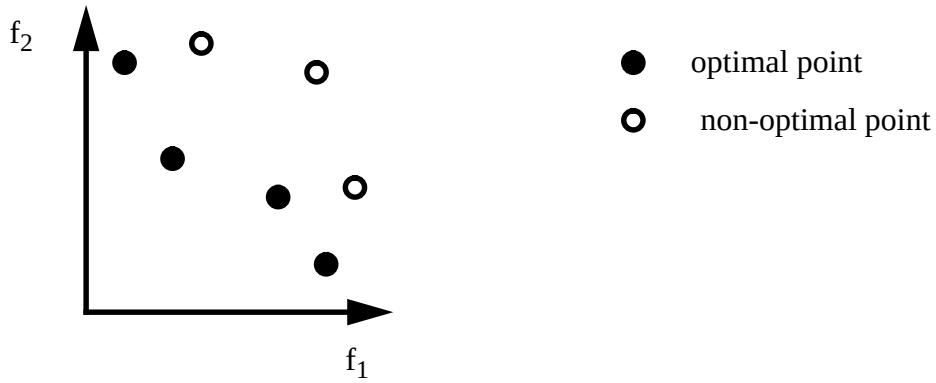


Figure 16. Two-objective function showing Pareto-optimal set

Another possibility mentioned in the literature is to use a “distance” in Pareto-space as a single valued functional valuation [22]. Subjectivity is introduced with this method

because the point from which the “distances” are measured must be specified by the designer. In many cases, however, the origin can be specified easily with little prejudice. To design a small part like a sewing machine bobbin for purposes of manufacturing, two “natural” axes are manufacturing cost and mean time between failures (MTBF). For the cost axis, the “optimal” point would be zero cost and for the MTBF axis, an infinite life is optimal. While this seems to be a good solution, there are two drawbacks. The first is that the different axes of the Pareto-space will typically need to be scaled because without scaling the values one axis may tend to dominate the other. Consider again the sewing machine bobbin. If the units of the cost axis measures are dollars and the unit of the MTBF axis are hours, the affect of cost would not be properly reflected in the design since these parts are inexpensive and last a long time. Second, even if the axes are scaled “properly”, such that the range of all axes is equivalent, the use of a Euclidean distance metric can yield results that are far from optimal. Consider the case of a Pareto-space with two axes using a Euclidean distance metric. If the value for one of the axes is at its maximum value and the other is at zero, the fitness will be 71% of the maximum possible fitness. As more axes are added, this effect is magnified - with 3 axes, with two maximized and one at zero, the resulting fitness is 82% of the maximum. In certain instances, this might be a valid evaluation, but for other designs, minimum values along certain axes may be required, thus the use of an unmodified distance metric may not be acceptable.

A third solution is to extend the concept of the SFM to make it more general. Instead of assigning a simple proportional constant to each of the multiple objectives, we propose Equation (7), a more general form:

$$O = f_1(o_1, \ldots, o_n)\, o_1 + \ldots + f_n(o_1, \ldots, o_n)\, o_n. \tag{8}$$

Since GAs do not require that the evaluation function be continuous, the weighting functions in Equation (8) can be used to express a wide range of possibilities.

Consider the example given in Section 4.3 of designing a planar mechanism. The objective functions might be to get as close to the path as possible with as few links as possible. To skew the results in favor of a four-bar mechanism, the designer could reduce the path error objective if there are four links. This can be done by including a conditional expression in the weighting function for path error, a conditional that reduces the path error by some percentage if and only if the number of links is equal to four.

Equation (8) provides the designer with tremendous freedom for specifying an objective function. To aid the designer, we propose the following objective function that combines the generality of Equation (8) with the Pareto distance metric. Given an objective vector in Pareto space, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and an optimal point in the Pareto space $\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$. Instead of computing a Euclidean distance in Pareto space

$$O = \sqrt{(x_1 - \bar{x}_1)^2 + (x_2 - \bar{x}_2)^2 + \ldots + (x_n - \bar{x}_n)^2}, \tag{9}$$

which has the problems listed above, the product of the projections is calculated:

$$O = (x_1 - \bar{x}_1)\,(x_2 - \bar{x}_2) \ldots (x_n - \bar{x}_n)\ . \tag{10}$$

In formulating Equation (10), it is assumed that each of the difference terms is non-negative. The differences between the two formulations is shown in Figure 17. This figure shows lines of equal objective values for two parameters that are equally scaled. The plot on the left shows the Euclidean distance metric measure, the one on the right the product of the projections.

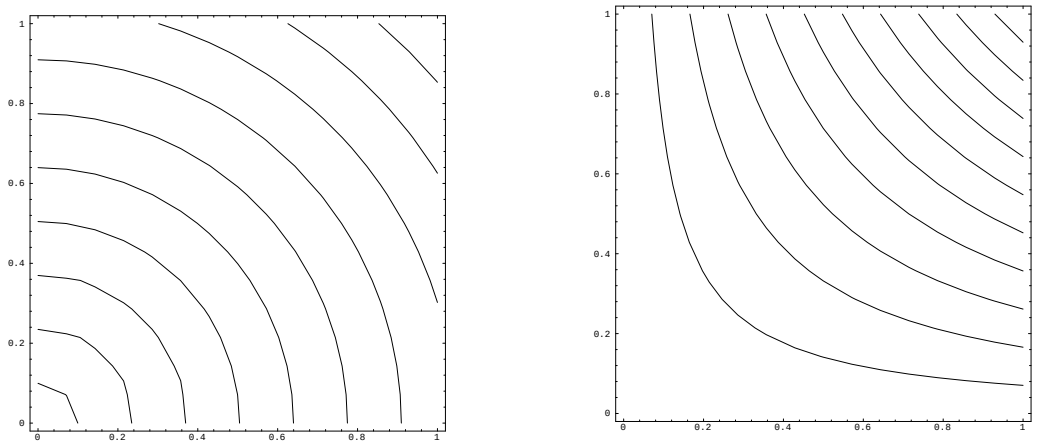


Figure 17. Comparison of distance and product of projection evaluation functions

Choosing between Equations (9) and (10) will produce very different results. Using the distance metric can yield a wide variety of good options, with a wide variety of values for each of the parameters. To achieve a similar objective evaluation using the product of the projections methods yields a narrower range of options, but neither parameter value will be low. The choice of the method to use depends on the problem being studied. If an acceptable problem solution requires that all parameter values be good, the product of projections method should be used. Otherwise, the distance metric, which will typically produce a wider variety of results, can be used.

In summary, multi-objective optimization is a far more difficult problem than single objective optimization. Using any technique to change a multi-objective problem into a single objective problem entails subjectivity. All practical design problems have multiple objectives and these subjective decisions are made on a regular basis by practicing engineers, although they are not necessarily aware of the subjective nature of their decision (see [102], pages 45-6). Thus, the use of Equation (8), although its subjective nature might be unsettling, is what is done in practice, and this work simply requires that these subjective weights be brought forward and be clearly stated.

4.5 Control parameters

In addition to selecting two cross-over probabilities and the two mutation probabilities, Section 3.3, a number of other control parameters must be selected. Koza, [52], page 105ff, identifies twelve other numerical parameters and six qualitative parameter for use with GP. Each is described below:

Numerical parameters:

1. Population size - The number of individuals in the population.
2. Generations - The number of test/select/reproduce cycles, see Figure 13.
3. Probability of crossover (p_c) - See Section 3.3.
4. Probability of reproduction - This is $1.0 - p_c$ and is not used explicitly in GD.
5. Probability of crossing over at an internal node - In a binary tree, if nodes are selected randomly, the probability is 0.5 that a terminal node will be selected. (If the `Arity` of a node is greater than two, then this probability decreases.) To achieve a greater number of non-terminal node cross-overs, the ratio of internal node cross-overs to terminal node cross-overs is specified.
6. Maximum size for S-expressions - This constraint is imposed for practical reasons, to limit the amount of time it takes to evaluate a tree. In this work, a maximum artifact size constraint is imposed. This constraint is based on practical design considerations. For instance, in creating a planar mechanism, the number of links might be limited by manufacturing costs. For GD, simply limiting the size of the overall tree may not yield meaningful results. Rather, the size of sub-trees can be limited. For example, consider the design of a riveted truss. By simply limiting the size of the tree, a truss with many members and few rivets may be generated. By separately limiting the number of members and the number of rivets, however, more meaningful results can be obtained.
7. Maximum initial size for S-expressions - The largest size for the randomly created population of generation 0.
8. Probability of mutation - See Section 3.3.
9. Probability of permutation - In the context of GP, permutation is essentially a type of mutation. To perform the permutation operation, a node in the tree is selected (with probability of permutation). The *k* nodes "beneath" this selected node are rearranged into one of *k*! possible new permutations. Permutation is not used in GD because there is no theoretical work to suggest that this is a useful operation and to implement this within the framework of STGP would be quite difficult.
10. Frequency of applying the editing operation - The editing operation is used to simplify the structure of an S-expression. For example, in a GP, for an S-expression such as `(+ 1 2)`, the editing operation would replace this with a `3`. Editing is not used in GD

because the effect of editing is unclear and “editing” a physical artifact does not make sense. However, for stochastic artifacts that generate controllers, editing of the controller is performed every generation.

11. Probability of encapsulation - Encapsulation is the automatic identification of identifying S-expressions and naming them for future reference. In GP, a sub-tree selected (with probability of encapsulation) is removed from the parent tree and replaced with an atomic function, where the function is the sub-tree that was just removed. This encapsulated function is added to the set of terminal functions and can be used by all members of the population. Encapsulation is used in GP for reasons of efficiency and because encapsulation reduces the probability that a genetic operation will destroy a useful routine. Encapsulation is not used in GD because there is no efficiency gain to be accrued and because of the complexity of implementation.
12. Probability of decimation/decimation percentage - Decimation is the operation of removing some portion of the population from further consideration. This can be useful in the initial population has a large number of members that have poor fitness. To use the decimation operation, an initial population larger than the actual population is generated. Immediately after the evaluating the initial population, fitness proportionate selection is performed to select the members of the actual population.

Qualitative parameters:

1. Means of generating initial population - Koza presents two methods for generating initial populations. The first is called “full” and is implemented by generating individuals whose size is the maximum allowed by the maximum initial size parameters (see above). The second is called “grow” and is implemented by adding branches to the initial tree with a certain probability with the restriction that the maximum size is not exceeded. Koza uses a method that combines both of these methods to generate the widest possible range of initial structures. For GD, the “grow” method alone is used, because the combined method would be very difficult to implement and might need to be tailored to each specific application.
2. Selection method - Fitness proportionate selection is used throughout, Section 3.1. Some experiment were conducted using tournament selection (an alternative means of selection reported in the literature), but this method of selection failed to produce good results.
3. Method for selecting second parent - Same as the method for selecting the first parent, random selection.
4. Fitness scaling - See page 28.
5. Over selection - Over selection is a means to select a higher proportion of fitter individuals from the population. Koza uses this method for problems that have large popula-

tions. Over selection is implemented by first rank-ordering the population, the dividing the population into two groups - the first group containing *c* percent of the fittest individuals, with the second group containing the rest of the population. During selection a higher percentage of the individuals selected are taken from the first group than the second. This was not implemented for GD since the selection method used, remainder stochastic sampling without replacement, (see page 29), does not have some of the problems associated with the “pure” fitness proportionate selection method of sampling used by Koza.

6. Elitists - In each generation, some small number of the best individuals in the population can be copied directly into the next generation (they are subject to mutation, which is typically a low probability). This will tend to make the maximum fitness monotonically increase with time as the best individuals will persist.

The values of the control parameters will be given for each experiment or group of experiments.

5 Stepping stone walker

➤ The stepping stone walker model is a planar abstraction of a frame-walking robot. This model is intended to test the GD methodology. This model explores both deterministic and stochastic models. The deterministic model uses a fixed gait to cross the terrain, the stochastic model incorporates a terrain-adaptive controller.

This model abstracts walking across a terrain that has areas onto which the vehicle can stand and areas on which it cannot go. This terrain model is referred to as a stepping stone terrain. The stepping stone terrain model can be derived from models of actual terrains. Consider Figure 18. This figure shows three terrain models. The first is a fractal model as is frequently used for terrain modelling, [59] [75]. The second shows how this terrain might appear to a scanning sensor, such as a scanning laser range finder, which averages the nearby points in the terrain. These sensors are essentially low-pass filters. The third shows the effect of thresholding the second terrain. This thresholding operation indicates a step region if the change in height of the sensed terrain is less than a specified value. Since these operations are those typically performed by autonomous robotics systems and since the third terrain resembles those shown in Figure A2, Example 4, the stepping stone terrain model can be said to be analogous to natural terrain.

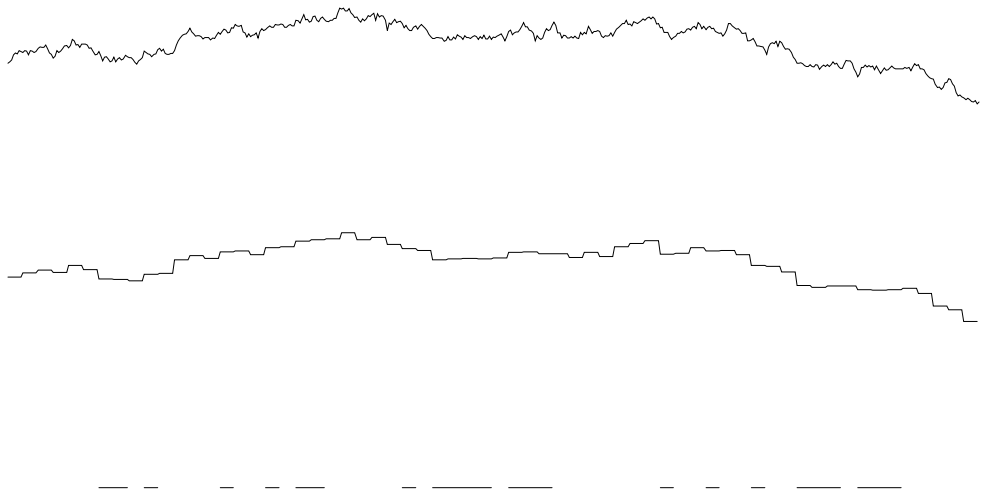


Figure 18. Comparison of natural and stepping stone terrains

One way to envision the stepping stone walker problem is shown in Figure 19, which shows a frame-walker attempting to cross a stream by stepping only on the few, available rocks (footfalls).

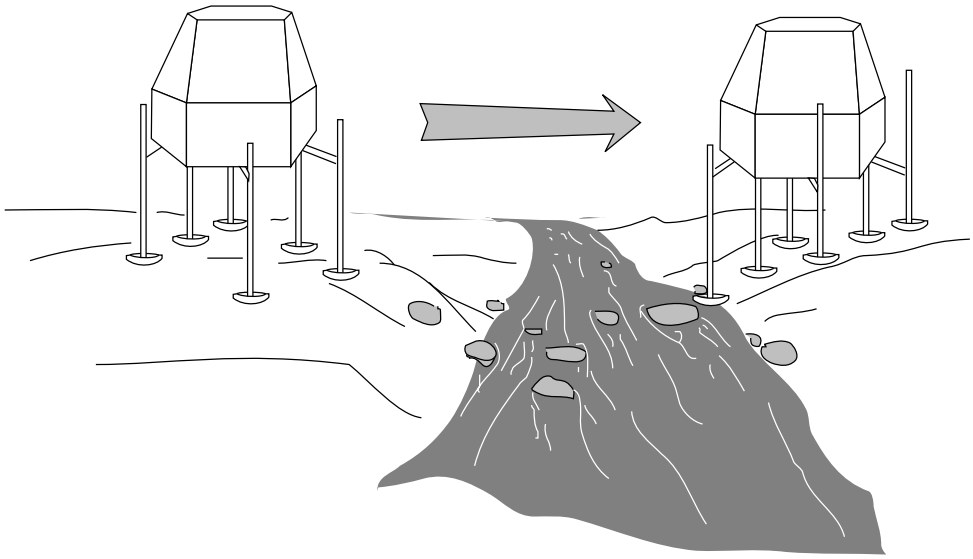


Figure 19. Frame walker crossing rugged terrain

A frame-walker is considered to be the simplest, statically-stable walking machine that can negotiate rugged terrain [5][85]. Frame-walkers, as currently implemented, consist of two frames, each of which has three or more legs. To move forward one “step”, a frame-walker lifts its first frame, moves it with respect to the second frame, sets down the first frame, then repeats the process with the second frame. The legs on a given frame are fixed (in the direction of motion) and do not move with respect to each other.

The stepping stone model is a planar abstraction of a frame-walker with as many as *n* frames, where each frame can have as many as *m* legs. The vehicle is said to make forward progress if the currently moving frame can find support points under at least two of its legs. Limits on the maximum number of frames and legs are imposed for three reasons:

- to prevent the GD algorithm from simply enlarging the vehicles to span the area that needs to be traversed,
- to maintain a certain degree of “reality”, since an actual vehicle must have a relatively small number of frames and legs for cost and control reasons, and
- to simplify computer implementation.

Although this is a simple model, several concepts can be explored. This model incorporates a controller into its design. The controller is used to define the forward movement of a frame. This model can also incorporate a sensor[1]. Such a sensor could be used to modify either the local or global foot placement patterns. Although the model below employs a single type of leg and a single type of foot, more types with different capabilities could be added. One potentially useful additional type is a “space”, where a space is similar to a leg but does not have a pad, see below for descriptions.

1. This model already incorporates an implicit sensor model in that a frame cannot move forward unless there exists terrain beneath the legs.

5.1 Stepping stone walker model

The following sections describe the stepping stone walker model and the terrain with which it interacts. Section 5.1.1 describes the grammar used to construct a stepping stone walker. Section 5.1.2 describes the terrain and the parameters for varying the terrain. Section 5.1.3 explains how the vehicles are evaluated. Finally, Section 5.1.4 describes the results from using a meta-GA (see Section 3.4) to determine a set of control parameters that were obtained for this model. The remainder of the chapter, starting with Section 5.2, describes the results obtained from various tests using this model.

5.1.1 Vehicle grammar

Figure 20 shows a sketch of one frame of the stepping stone walker. Stepping stone walkers are comprised of as many as *n* frames, where the frames can be envisioned as being stacked in a direction perpendicular to the plane. One important constraint is that all frames of a stepping stone walker must overlap in the direction of motion. This constraint is imposed to mimic a real frame-walker in which the frames must remain connected.

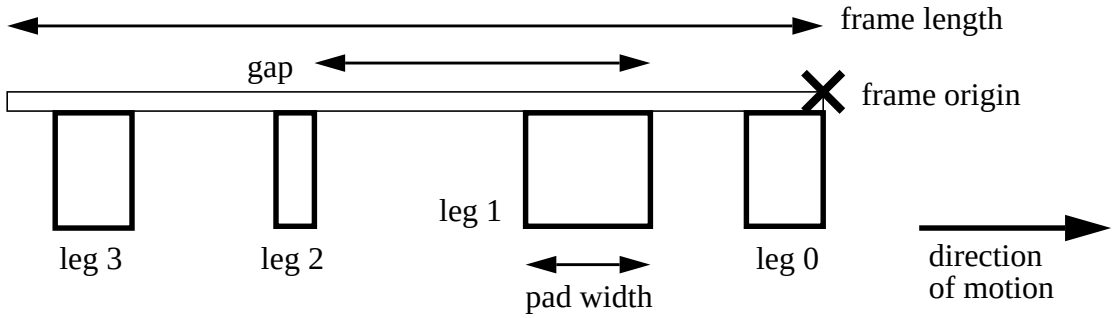


Figure 20. Stepping stone walker vehicle model

For the GD methodology, a formal grammar is required to describe these vehicles. Table 5 shows a context free grammar for generating stepping stone walkers. The **F**, **L**, and **C** are tokens to help identify the individual frames and legs and the controller.

vehicle →	*frames* **C** *controller*
frames →	*frame frames* | *frame*
frame →	**F** *number legs*
legs →	*leg legs* | *leg*
leg →	**L** *number number*
number →	8 bit number

Table 5: Stepping stone walker context-free grammar

The *number* associated with *frame* is the frame control parameter. The stepping stone walker moves by moving each frame in sequence. The distance that a frame moves forward

is given by the minimum of the frame length and the frame control parameter. Since *numbers* are specified by eight-bit numbers, the largest terrain period that can be negotiated with a two-frame, two-legged vehicle is 510. To accommodate larger periods, the vehicle model used in Section 5.2 and Section 5.3 can incorporate a user specified constant parameter to multiply the control parameter, but the distance moved forward is still constrained by frame length. The experiments in Section 5.2.3 and Section 5.2.4 demonstrate the difference in the result when this multiplier, the control parameter multiplier (CPM), is used. In Section 5.3.6 the ability to generate a vehicle level *controller* is added to the model. This controller automatically sets the control parameter multiplier based on terrain information received from “sensors”.

The first number associated with *leg* is the gap distance. The second is the pad width, which comes from the notion of a foot-pad at the end of the leg. A two-frame vehicle with three legs on the first frame and two on the second would be represented as:

F 142 **L** 127 12 **L** 255 233 **L** 108 20 **F** 220 **L** 45 18 **L** 120 29

where the values of the constants were chosen “randomly”. For the first frame, the control parameter (142) is less than the frame length (490), thus the distance stepped is equal to the control parameter. For the second frame, the control parameter (220) is greater than the frame length (165), thus the distance stepped is equal to the frame length. Figure 21 shows a graphic representation of this vehicle walking across a continuous terrain. The terrain is represented by the light green color, the frames by the light red and the legs by the dark red. This diagram, like those in Sections 5.2.4 and 5.4.11, is drawn to scale.

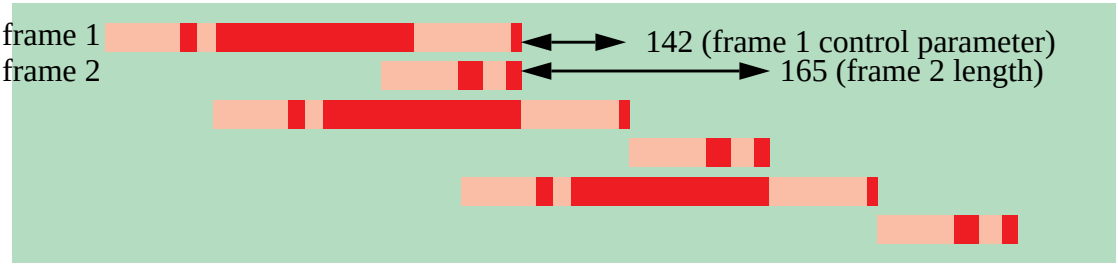


Figure 21. Stepping stone walker motion

Figure 22 shows a portion of this stepping stone walkerstepping stone walker represented schematically as a tree structure. (The dashed arrow represents missing, intermediate nodes.)

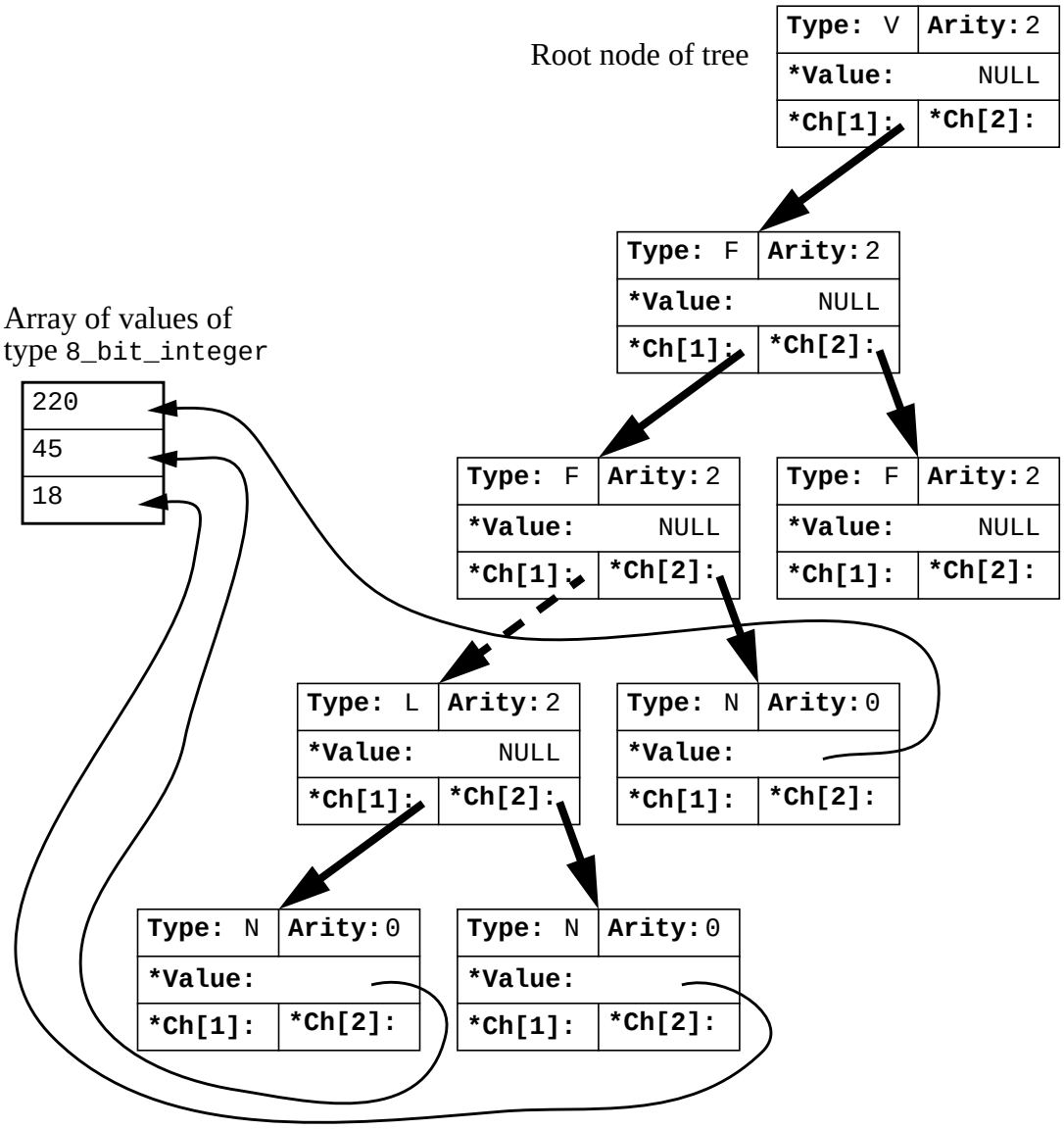


Figure 22. Tree representation of example stepping stone walker

5.1.2 Terrain model

The terrain for the stepping stone model is a binary terrain. The two binary values in this terrain represent places where a leg can and cannot be placed. The simplest terrain is a periodic terrain, with periodic spacing of step and no-step regions. A more difficult terrain model adds a Gaussian distributed random distance (mean zero, standard deviation defined by the user) to the edges of the step region. Another level of difficulty can be added by allowing the period to be modified by a Gaussian distributed random distance. Fractal terrains can be generated by combining both deviations. These binary, fractal terrains bear

some resemblance to cross-sections of stochastic boulder fields, such as those found on the moon [66], since they are both Gaussian distributions. The effects of the different terrain models on the vehicles is discussed in Section 5.2. Figure 23 presents a graphical interpretation of the parameters that affect the generation of a terrain model. For further details and some examples, see Appendix A.

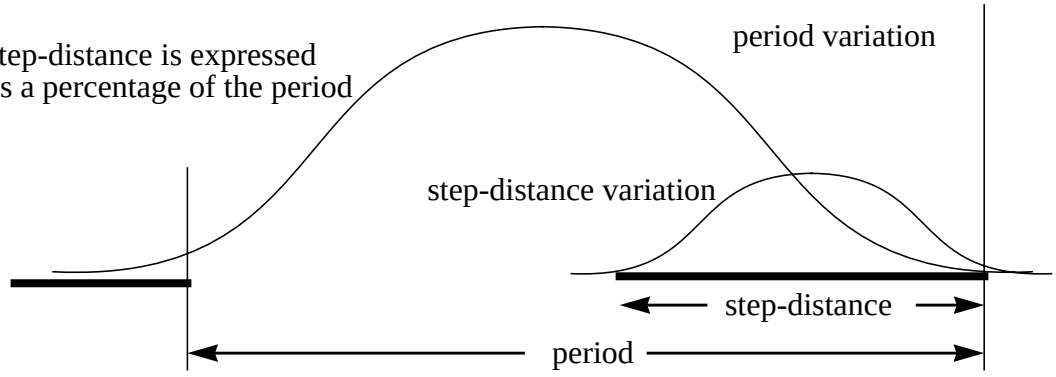


Figure 23. Terrain model parameters

To simplify the evaluation, the terrain starts at the world origin, and exists only on the positive direction. (In this work, the vehicles move to the right.) All points to the left of the origin and to the right of the goal position are considered to be step locations. The goal position is given by the terrain period multiplied by the number of terrain periods.

To ensure that the vehicles generated are robust with respect to the terrain model and not a particular instantiation of that model, a new terrain is generated each generation. For those terrains with no period or step-distance variation, the terrain remains the same from generation to generation.

5.1.3 Model evaluation

There are two steps in the model evaluation. The first step checks to see if the model is viable, the second evaluates its performance.

A model is considered viable, if for every leg, the gap distance is greater than or equal to the pad width. This restriction is applied to ensure that the legs are distinct. Also, each viable vehicle must have at least two frames and each frame must have at least two legs. These restrictions reflect the notion of an actual frame-walker which requires a minimum of two frames for locomotion and three legs for stability. Since this is a planar abstraction, the stability requirement is reduced from three legs to two.

Vehicle performance is based on the vehicle's ability to traverse the specified terrain. To traverse the terrain, the vehicle moves one frame at a time. The vehicle starts with the frame origins of all frames aligned with the world origin. The vehicle moves one frame at a time, starting with frame 0, then moving all successive frames in order. After the last frame is moved, frame 0 is moved again and the process repeats until the vehicle can no

longer move or the goal position has been reached. Frame motions are constrained in the sense that each frame must overlap at least one other frame in the direction of motion. The new location of a frame is determined by adding its control distance to the origin of the frame that previously moved, see Figure 21. This location is checked to see if there are step locations beneath at least two legs (stability). If two such step locations are found, this location is the new location for the moving frame. If not, the frame returns to its original position and the cycle continues. Forward progress ceases when all frames can no longer advance.

This method of evaluation assumes a predefined gait. With this gait, the legs cycle in order: for example $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$. This predefined gait can have the affect of causing a vehicle to perform poorly that might perform better with another gait pattern. However, because of the restriction on over-lapping, once a gait is committed, it cannot be changed; that is, a particular frame cannot move twice before every other frame has been moved. To change the gait pattern, $n-1$ of the frames would have to line up before the pattern could be changed. This would require some sophisticated planning and additional steps are required to traverse the terrain. Since there are no special frames, the fixed gait pattern restriction should not impact the results.[2] Consider, as an example, a perfectly ambidextrous hurdler. Such an individual would be equally capable of completing a hurdle course regardless of the gait pattern used, that is, whether the right leg or left leg is used to clear the hurdle. There are terrains, however, for which the fixed gait pattern will be inefficient. Consider again an ambidextrous hurdler, but this time, the hurdler can only hurdle high hurdles with the right leg and low hurdles with the left. On a track set with all hurdles of the same size, the hurdler will perform efficiently. However, if the hurdles alternate in size, the hurdlers gait pattern will need to be changed after each hurdle. While this can be accomplished, it results in a poor performance. Within the scope of this work, no terrains have been knowingly designed that would require the need to switch gait patterns for efficient performance.

Since the primary purpose of a walking machine is to traverse terrain, it is logical that the evaluation function should be a function of distance travelled. However, simply using this as a single fitness metric would produce a tremendous variety of possible designs since no secondary objectives were stated. Other objectives used in evaluating this model are:

- number of frames - smaller is better since additional frames increase the cost and complexity of the system.
- average number of legs per frame - only two legs are needed for stability, more than two increases cost.
- average pad width - smaller pad widths are less expensive to manufacture and do not have as great a potential to block sensor views as larger pads. For an actual vehi-

2. If the fixed-gait restriction were found to be detrimental to the results, another set of nodes, possibly attached to the top level node, could be employed to describe the gait pattern of the vehicles. This pattern would be subject to genetic manipulations just like all other parameters.

cle, the minimum size must be determined by the leg loading and the load carrying capability of the terrain. For this problem, both the leg and the terrain are considered to be sufficiently strong, thus smaller pads are best.

- number of steps to cross terrain - the fewer the number of steps required, the less energy is needed to move the vehicle. Also, taking fewer steps typically requires less computing power.

In keeping with Equation (10), each of these metrics is expressed as a distance in Pareto space. These distances are summarized in Table 6:

metric	value	measured from	optimal value
distance travelled	d	0	goal distance
number of frames	$frames$	MX_FR	2
average number of legs/frame	$\frac{\sum legs}{frames}$	MX_LG	2
average pad width	$\frac{\sum pads}{\sum legs}$	255	1
number of steps	$steps$	$dist/(255/2)$	1

Table 6: Metrics for evaluating the stepping stone walker model

The *number of steps* term in the fitness function is used to penalize vehicles that take many steps to cross the terrain. This term is added for practical reasons since on a real vehicle, extra steps would require additional computing resources and more mechanical power. This term is calculated as a distance relative to a vehicle that performs poorly. To traverse a periodic terrain with the fewest steps, a vehicle's period should match the terrain's period. For terrains with periods of up to 255, the optimal gait for the vehicle is the terrain period. However, if the terrain period is sufficiently small, that is, less than 255/2, the optimal gait is twice the terrain period. Thus the worst case number of steps that a vehicle would take to cross a terrain is given by

$$dist/(255/2)\ . \tag{11}$$

where *dist* is the distance covered by the vehicle. The goal distance should not be used instead of *dist* because this would falsely award certain vehicles that do not traverse the entire terrain.

When a fixed control parameter multiplier is used, the “number of steps” term has the effect of reducing the maximum achievable fitness for terrains whose period is other than an even integral multiple of 255. Consider the two following examples:

	Terrain 1	Terrain 2
maximum position	25500	25500
terrain period	500	510
controller parameter multiplier	1	1
number of moves to traverse terrain	51	50
steps (from Equation (11))	149	150

The important thing to note is that in comparing vehicle fitnesses to other vehicles or to maximum achievable fitnesses, those vehicles that were not tested on terrains with an even integral multiple of 255 may never achieve the greatest possible optimal fitness value.

In Table 6, the term $\sum pads$ is the summation of the vehicle’s pad widths, *frames* is the number of the vehicle’s frame, $\sum legs$ is the total number of legs the vehicle has, *MX_FR* is the greatest number of frames a vehicle can have and MX_LG is the greatest number of legs per frame a vehicle can have. The value for steps is calculated as a percentage of the fewest number of steps the vehicle can take to cross the given terrain. The product of the projections in Pareto space, Section 4.4, determines the fitness a vehicle can obtain and is given by

$$O = d\,(MX\_FR - frames)\left(MX\_LG - \frac{\sum legs}{frames}\right)\left(255 - \frac{\sum pads}{\sum legs}\right)(dist/\,(255/2) - steps)\,. \qquad (12)$$

Substituting the value 18 (chosen for practical reasons - to limit the largest possible vehicle size) for both *MX_FR* and *MX_LG* and the other optimal values shown in Table 6, and assuming that the vehicle traverses the entire terrain, yields the maximum possible fitness obtainable

$$O = d\,(18-2)\,(18-2)\,(255-1)\left(\frac{2d}{255} - \frac{d}{255c}\right) \cong 255d^2\,(2c-1) \qquad (13)$$

where *c* is the control parameter multiplier. (In practice, when presented, the actual fitness is divided by a large constant so the values are not so large, since typical values of *d* are on the order of 10^{10}. However, this scaling cannot be used within the program. Consider, for example, vehicle’s whose respective fitness is 20, 10 and 0. If these values are scaled by 10, their new fitnesses are 2, 1, 0. These new fitness values do not properly reflect the marked superiority of the first two vehicles to the third.)

An alternative formulation of Equation (12) is

$$O = \frac{d}{frames\,(\sum legs)\,(\sum pads)\,(steps)}. \tag{14}$$

The problem with this formulation is that small differences in the vehicle will have a tremendous impact in the vehicle's evaluation. For example, if the term $\sum pads$ equals four for one vehicle and eight for another, the former vehicle's performance measure will be twice the second's, all other factors held constant. While one of the goals of this evaluation function is to reduce certain parameters, the function shown in Equation (14) seems to over-weight them. The inability of this objective function to produce "proper" results was verified experimentally. (What typically happened is that a "large" vehicle would traverse 5% of the terrain in two steps. Since the number of steps taken was so small, this vehicle's fitness would be large, however, neither its performance nor its form are the desired results. This should serve as a warning that the results achieved from methods such as GD are highly dependent on the form of the evaluation function.) In this context, "proper" refers to the ability of the vehicle to traverse the entire terrain.

5.1.4 Control parameters

As mentioned in Section 3.4, properly choosing a large number of parameters is a daunting task. Thus, a GA program was written to help in the selection of the control parameters. The results from sample runs were combined and the resulting statistics are presented in Table 7. This table combines the results from using two different types of evaluation function. The first simply tracked the maximum score of the test vehicles after a given number of generations. The second evaluated the parameters by how fast the average and maximum scores converged to within some user-specified percentage of the absolute maximum score. A limited number of cases were run due to the very long execution time of this program. Each run of the stepping stone program requires about nine minutes on a Sparc 10. With a population of 50, each generation of the meta-GA requires about 450 minutes, and to run this for even 20 generations requires 9000 minutes, about 150 hours of computing time. Further, as indicated in Section 5.4.12, to perform the meta-GA experiment properly, each set of input parameters should be tested with a number of different random seeds and terrain models, thus increasing the required computing many-fold.

parameter	bits	range	average	stand dev	median	mode
genetic algorithm cross-over probability	6	0.0 - 1.0	0.5196	0.2962	0.5397	0.9524
genetic algorithm mutation probability	4	0.0 - 0.015	0.0064	0.0031	0.0060	0.0050
genetic programming crossover probability	6	0.0 - 1.0	0.4957	0.3009	0.4286	0.2698
genetic programming mutation probability	4	0.0 - 0.015	0.0098	0.0036	0.0110	0.0130
fitness multiplier	3	1.0 - 8.0	4.6848	2.5819	5.0000	1.0000
number of elitists	4	2 - 30	11.890	7.8007	10	4
population	3	50 - 400	328.48	63.515	350	300
ratio of non-terminal xovers to terminal xovers	2	1 - 4	2.8606	0.9794	3	3

Table 7: Result of meta-GA to determine GD parameters

The results from this table must be used with extreme caution. When using the second evaluation function, (the number of generations to achieve a fitness within a specified percentage of the maximum achievable fitness), certain combinations of input parameters did not allow the population to converge to the required percentage within a specified number of generations. Such parameter sets received a fitness of zero, however, these parameter sets appeared in the calculations summarized in the above table.

Due to the length of time needed to execute the meta-GA, the population size was not as large as typically desirable. It is probable that the small population size and limited number of generations led to less than optimal results. However, given all of the caveats, a set of parameters to be used for the following experiments can be selected, Table 8. While this set is certainly not optimal, it is almost certainly better than randomly guessing parameter values.

genetic algorithm cross-over probability	0.54	genetic programming crossover probability	0.45
genetic algorithm mutation probability	0.006	genetic programming mutation probability	0.010
fitness multiplier	5.0	number of elitists	12
population	350	ratio of non-terminal xovers to terminal xovers	3

Table 8: Control parameters for stepping stone walker experiments

Three sets of experiment were carried out with the stepping stone walker model. The first set used periodic terrains and fixed gait vehicles. The second set used aperiodic terrains and fixed gait vehicles. The third set repeated each of the experiments in the first two sets, but with non-fixed gait vehicles. The terrain parameters and the vehicle types used in the experiments are summarized in Table 9.

period	number of cycles	step per-centage	step variation	period variation	fixed gait section	non-fixed gait section
500	150	50	0	0	5.2.1	5.4.1
500	150	10	0	0	5.2.2	5.4.2
1000	150	50	0	0	5.2.3	5.4.3
100	150	10	0	0	5.2.4	5.4.4
500 375	75 100	50 50	0 0	0 0	5.3.1	5.4.6
375 500	100 75	50 50	0 0	0 0	5.3.2	
500 375	75 100	10 10	0 0	0 0	5.3.3	5.4.7
1000 800	50 62	25 25	0 0	0 0	5.3.4	5.4.8
500	150	50	10	0	5.3.5	5.4.9
500	150	50	0	100	5.3.6	5.4.10
500	150	50	10	100	5.3.7	5.4.11

Table 9: Summary of stepping stone walker experiments

5.2 First experiments - Periodic terrain

The purpose of the first set of experiments is to determine if the GD methodology produces optimal stepping stone walkers. Based on the performance metrics outlined in Table 6 and Equation (13), an optimal vehicle will have two frames, each frame will have two legs and the size of each leg will be one. Of course, this optimal vehicle can only exist for certain limited classes of terrains, specifically, those terrains with periods less than 510 (twice the maximum control parameter value). For terrains with larger periods, more legs will be needed. It is also expected that the sum of the control parameters will be equal to the terrain period (or be an integral multiple of it).

Each experiment was run with three control parameter multipliers (CPM): 1, 2 and 4. With CPMs other than 1, estimating the optimal value is difficult because to achieve the gait that minimizes the number of steps, the size of the vehicle is increased, and this increased size is difficult to predict. Thus, the fitness of the vehicles with CPMs other than 1 may exceed the maximum possible fitness value listed for that experiment. Experiments with CPMs other than 1 are presented primarily for comparison to the results obtained using the vehicle level controller, Section 5.4.

For each experiment, a table is presented showing the parametric values used for the terrain, the vehicle and the maximum possible fitness a vehicle can achieve on the specified terrain. (All other parametric values are shown in Table 8.) Then, a brief discussion of the expected results will be presented, followed by the actual results (the genome of the best individual and its fitness) and pertinent comments. To minimize the probability of getting anomalous results due to either a fortuitous or unlucky choice of an initial random seed, each experiment was run three times, each time with a randomly selected random seed. The results from three runs are presented for each experiment.

All experiments presented limited the initial number of frames and legs to six. The results of all experiments in which a larger, or smaller, initial number of legs were used, yielded the same results.

5.2.1 Experiment one

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: This is one of the simplest possible terrain configurations. With a CPM equal to 1, the resulting vehicles should approach the theoretically optimal vehicle, two frames, two legs per frame with small pads. The two control parameters should sum to 500, the terrain period, however, there is no reason for either control parameter to assume any specific value because the step percentage is quite large. Although specific values of the control parameters cannot be predicted, the control parameter associated with the second frame should be larger than 250 because the second frame moves first. Were the value of this parameter less than 250, the first frame would move first, thus necessitating one extra move, thereby reducing the fitness. The sum of gap dimensions on both frames should exceed the values of the control parameters (necessary for maximum forward progress).

With the CPM equal to 2, the vehicle will be able to traverse the terrain in its optimal configuration if all four gap values exceed 250, half the terrain period.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F246 L246 1 L252 1 F254 L255 1 L7 1	14008
2	F246 L247 1 L247 1 F254 L255 1 L253 1	14008
3	F248 L79 1 L252 1 F252 L253 1 L144 1	14008

run	genome: CPM = 2	fitness
1	F250 L249 2 L253 2 F250 L249 2 L252 2	21239
2	F251 L246 3 L254 4 F250 L246 3 L254 2	21155
3	F251 L247 1 L230 1 L245 1 F249 L245 1 L254 2	20640

run	genome: CPM = 4	fitness
1	F247 L247 12 L241 1 L255 12 L225 1 F247 L241 15 L246 4 L255 8 L254 6	21268
2	F251 L254 4 L254 1 L243 11 L222 5 F230 L254 5 L254 12 L254 4 L255 6	21260
3	F252 L237 2 L204 9 L250 1 L252 3 F253 L245 4 L232 6 L249 4 L168 14	21230

COMMENTS: The results of the three runs with the CPM equal to 1 are as expected. In all cases, the pad widths had been optimally minimized (the optimal value is 4) and the control parameters add to 500, the terrain period. The minimum number of frames is achieved because the sum of the gap dimensions on both frames exceeds the control parameter. As expected, the individual values for the control parameters were not predictable, except that the control parameter for the second frame does exceed 250 in all cases. The difference between the maximum observed fitness, 14008, and the maximum possible fitness, 14295, is explained by the fact that the terrain has a period that is not an integral multiple of 255.

The results with the CPM equal to 2 and 4 show two things: first, the effect of taking fewer steps has a greater impact on the vehicle's fitness than does a small increase in the number of legs. This is seen by the fact that the fitnesses of these larger vehicles exceed that of the smaller vehicles. Second, the solutions found to the problem are not intuitively obvious. In run 1 with CPM = 2, the solution found is aperiodic with respect to the terrain. However, the amount of the aperiodicity times the number of terrain cycles is less than the step region, so this does not have any detrimental affect on the vehicle's performance. However, if the terrain size were increased to be greater than 251 cycles, this vehicle may fail to traverse the entire terrain.

5.2.2 Experiment two

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: In this experiment, the percentage of the terrain on which the vehicle can find a footfall is quite limited. With the CPM equal to 1, the second frame (the one that moves first), will require three legs. The reason for this is that three legs will be needed to span the large gap. The second frame should have only two legs, and the pads on all legs should be small. As in the previous experiment, the results with the larger CPM values are hard to predict.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F252 L35 1 L243 1 F248 L252 0 L10 1 L183 1	13581
2	F250 L27 1 L247 1 F250 L249 1 L26 2 L56 1	13559
3	F245 L32 1 L213 1 F255 L253 0 L17 4 L197 1	13549

run	genome: CPM = 2	fitness
1	F251 L29 1 L254 2 L253 0 F249 L43 1 L231 1 L245 0	20004
2	F250 L250 1 L250 1 L139 1 F250 L251 1 L250 0 L4 4	19964
3	F254 L20 1 L254 2 L251 1 F246 L251 3 L247 1 L139 1	19951

run	genome: CPM = 4	fitness
1	F254 L254 1 L252 1 L239 0 L252 0 L248 10 F246 L250 31 L252 1 L228 3 L254 17	20570
2	F129 L240 2 L251 14 L255 14 F246 L255 1 L228 0 L201 11 L202 4 L245 1	20392
3	F131 L243 2 L242 2 L239 19 F244 L255 25 L237 1 L238 2 L241 2 L227 17	20157

COMMENTS: With the CPM equal to 1, the leading leg of the second frame (the frame that moves first) will fall in the no-step region. For the second leg on the second frame to make contact with the step region, the sum of the first gap parameter and the second pad width must be greater than the control parameter, allowing it to reach the previous step region. This necessitates a frame with three legs, where the leading leg is not expected to contact the terrain. Since this leading leg is not used, its pad width should be reduced to zero, as in Runs 1 and 3. Run 1 also resulted in the optimal vehicle that can traverse this terrain.

As with the previous experiments, the larger CPM led to an increase in fitness. It is interesting to observe that with the CPM equal to 4, the best vehicle developed is also the largest. These results suggest that letting the simulation run for a greater number of generations will eventually result in Run 2 or 3 yielding a higher fitness, because these vehicles are smaller, thus they can potentially have a higher fitness than the larger vehicle. This premise is shown to be correct in the following table. Where Run 1 showed little to no improvement, the fitness values for Runs 2 and 3 not only showed improvement, their fitnesses exceed the fitness from Run 1.

run	genome: CPM = 4, generations = 300	fitness
1	F254 L255 4 L240 0 L250 12 L253 2 L253 4 F246 L253 8 L254 4 L255 2 L254 5	20782
2	F129 L249 4 L247 3 L215 10 F246 L251 3 L255 3 L255 3 L247 0	21304
2	F131 L255 4 L253 1 L189 1 F244 L251 4 L252 3 L253 1 L252 1	21437

5.2.3 Experiment three

# of terrain cycles: 150	*terrain period:* 1000	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 57277

EXPECTED RESULTS: With CPM=1, the expected results are vehicles with two frames, with three legs on the second frame (the frame that moves first) and four on the first. At first, the need for four legs on the first frame is not obvious, since the no-step region can be spanned with a frame that has three legs. However, to efficiently traverse the terrain, the sum of the control parameters should sum to 500, half the terrain period, thus the leading leg on the second frame will fall short of the edge of the step region on its first move. Since the second leg will be in the middle of the no-step region, two more legs are required. With CPM=2, this experiment is analogous to the experiment in Section 5.2.1 with the terrain parameters doubled. The resulting vehicles should have three legs on frame two (the frame that moves first), so the no-step region can be spanned, and two legs on the first frame.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F251 L251 2 L247 1 L167 3 L3 1 F249 L247 1 L251 7 L64 1	50610
2	F251 L251 2 L253 1 L7 4 L205 3 F249 L247 3 L114 5 L100 1	50524
3	F245 L254 3 L246 1 L247 1 L254 2 F255 L254 3 L68 5 L125 4	50524

run	genome: CPM = 2	fitness
1	F248 L245 1 L254 1 F252 L255 1 L254 1 L97 1	82721
2	F247 L252 1 L252 1 F253 L255 1 L255 3 L253 3	82461
3	F249 L254 3 L247 1 F251 L251 3 L252 1 L66 2	82396

run	genome: CPM = 4	fitness
1	F255 L249 2 L251 12 L250 15 L243 3 F235 L249 8 L254 4 L241 8 L122 1	84662
2	F249 L255 6 L242 5 L251 5 L238 18 F252 L245 4 L246 28 L251 13 L250 1	84334
3	F255 L249 17 L252 12 L253 3 L239 12 F237 L244 10 L252 6 L230 6 L225 22	83744

COMMENTS: The results from these experiments are as expected. As in the previous experiments, the results with CPM = 4 are the best because of the smaller number of steps taken by the vehicles, even though the vehicles developed are the largest.

5.2.4 Experiment four

# of terrain cycles: 150	*terrain period:* 1000	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 57277

EXPECTED RESULTS: The very large no-step region in this experiment indicates that with CPM=1, both frames will require a large number of legs, six legs on one frame and five on the other. As before, the control parameters should sum to an integral multiple of the terrain period.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F255 L33 4 L233 26 L237 1 L46 5 L218 6 F245 L235 7 L80 14 L184 5 L172 8 L149 93 L180 3	41324
2	F254 L50 21 L237 6 L234 1 L35 5 L34 5 F246 L221 35 L53 32 L190 5 L241 24 L82 51 L180 6	41026
3	F253 L85 12 L236 4 L242 197 L224 2 F247 L174 2 L129 75 L219 2 L239 4 L31 9 L135 1	40254

run	genome: CPM = 2	fitness
1	F247 L33 4 L251 1 L246 2 F253 L255 0 L251 9 L86 1 L252 1	76906
2	F254 L56 3 L244 2 L240 2 F246 L255 7 L241 2 L57 2 L144 2	76819
3	F252 L64 2 L255 3 L213 7 F248 L248 1 L255 1 L15 5 L117 3	76732

run	genome: CPM = 4	fitness
1	F20 L85 1 L255 5 F230 L254 1 L254 1 L163 0 L252 0 L47 4	77167
2	F130 L85 3 L206 1 L242 0 F120 L252 3 L254 0 L33 4 L242 5	76993
3	F8 L58 1 L187 3 F242 L58 1 L251 1 L234 5 L240 4 L225 3	76906

COMMENTS: The results for this set of experiments are as expected. However, the pad widths do not appear to have been reduced to their smallest achievable values. A possible explanation for these results is that an insufficient number of generations were examined. This design problem can conceptually be divided into two sub-problems: finding a configuration (the number of frames and the number of legs per frame) and finding numerical values for this configuration. Since this particular test terrain is challenging, in Run 1, the first 37 generation are used just to find a configuration that can traverse the entire terrain. Once such a configuration is found, its genes start to propagate throughout the population and the remaining generations act to optimize this configuration. This assumption is born out by rerunning run 1 for 300 and 450 generations. The results from this run are shown below.

run	genome: CPM = 1, generations = 300 and 450	fitness
1a	F255 L18 14 L232 8 L254 4 L22 7 L54 9 F245 L248 2 L77 2 L166 2 L251 1 L30 9 L171 5	43035
1b	F255 L22 3 L241 1 L240 1 L22 1 L200 6 F245 L237 4 L93 13 L206 1 L248 5 L40 6 L162 2	43349

The results with CPM=4 are quite interesting in that two very different solutions were found. In Runs 1 and 3, the vehicle takes one large step to cross the no-step region followed by a small step to stay on the step region. This resulted in a configuration with two legs on one frame and five on the other. Run 2, however, produced a vehicle with four legs on one frame and three on the other that does not span the no-step region with its longer frame. Figure 24 shows these vehicles traversing the terrain. In this scale drawing, the green areas represent the step regions of the terrain, the light red rectangles are the vehicle frames and the dark red linesa re the feet. Since this is a scale drawing, the feet may not be clearly distniguishable. It is

this type of discovery, the existence of an artifact configuration other than the one expected, that is one of the key features of the GD methodology.

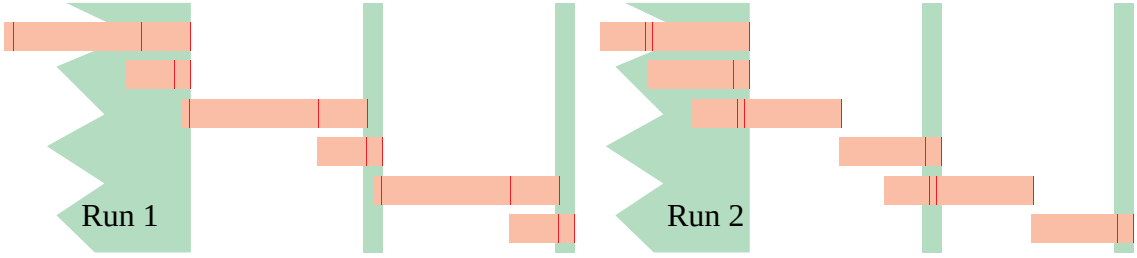


Figure 24. Stepping stone walkers traversing terrain

This discussion would seem to indicate that the GD methodology might produce better results if the parameters could be adjusted dynamically. For example, at the beginning of the run, the GP type operators would be set higher and after some number of generations, the GA type operators would be set higher. The primary difficulty with this concept is knowing when, and how, to modify these parameters. One method might be to look at the fitness value of the best of generation and once it reaches some certain threshold to modify the parameters. This method can lead to erroneous results. Consider Section 5.2.2, CPM=4: in this case, a configuration, Run 1, that looked quite promising after 150 generations yielded relatively poor results after 300 generations.

Another alternative might be to look at the change in the fitness of the best of generation from generation to generation, and to change the parameters when a particular threshold is exceeded. The difficulty with this method is being able to predict how the fitness of the best of generation will change so a threshold can be selected. In running these experiments, there were cases where the change in fitness over the generations is fairly smooth, and there have been other cases in which the fitness changed dramatically. Although finding the point for parameter changing is easily done after the run has been completed, finding an appropriate point during the run may prove to be quite difficult.

5.2.5 Summary of first experiments

The primary value of this first set of experiments was three fold: first, to test the GD methodology; second, to carry our experiments that were simple enough that the results could be predicted; and third, to provide a point of comparison for the later experiments with the vehicle level controller. The first two objectives were clearly achieved. These experiments were also helpful in developing a set of computer rotuines for performing genetic operations.

5.3 Second experiments - Aperiodic terrain

In this set of experiments, the simple, periodic terrains of the previous set of experiment have been become more “rugged”. In this set of experiments, the terrain may be the combination of two or more periodic terrains or aperiodic terrains, as shown in Figure 23 and Appendix A. It is expected that, in general, vehicles with fixed gaits will do poorly on these terrains because they will not be able to match periods with the terrain.

When examining the results of the experiments with aperiodic terrains (Sections 5.3.5 through 5.3.7 and Sections 5.4.8 through 5.4.11), it is important to remember that the terrain is changing after each generation. The fitness values shown are those achieved on the last terrain traversed by the vehicle. For those experiments with terrain adaptive controller (Section 5.4) if the terrain produced in the last generation is more fragmented than usual, the resulting vehicles may require more steps to cross the terrain, thus lowering their fitness. Similarly, if the terrain produced in the last generation is more benign than usual, the resulting vehicles might have somewhat inflated fitnesses. Thus, when comparing the results between runs, or between controller types, it is important to look at the range of results presented, not just the best.

5.3.1 Experiment one

# of terrain cycles: 75	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 100	*terrain period:* 375	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: This terrain starts exactly the same as the terrain in Section 5.2.1, however, at the mid-point, the terrain period is changed. The terrain period after the change, 375, is the value that would cause a simple vehicle with a fixed gate of 500 to fail most quickly. Thus, in all cases, it is expected that the solutions to this experiment will have more than two legs per frame. With CPM=1, the control parameters should sum to 500, as opposed to 375, since fewer steps will be required. The extra legs are used to traverse the terrain after the change in period.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F249 L183 2 L112 2 L179 1 F251 L177 4 L112 2 L184 1	13081
2	F249 L159 1 L110 2 L154 1 F251 L145 4 L130 2 L12 3	13072
3	F249 L128 1 L140 1 L6 1 F251 L142 4 L135 1 L219 5	13072

run	genome: CPM = 2	fitness
1	F250 L249 3 L254 5 L255 1 F250 L254 3 L250 3 L56 4	19820
2	F250 L158 1 L135 2 L217 3 F250 L151 5 L141 3 L222 9	19767
3	F250 L138 3 L136 2 L243 6 F250 L131 4 L137 2 L253 6	19767

run	genome: CPM = 4	fitness
1	F156 L230 22 L158 2 L227 2 L214 7 F220 L254 3 L212 3 L191 1 L238 12	20383
2	F152 L122 6 L190 5 L219 6 F242 L239 12 L214 11 L238 12 L207 11 L188 4	20188
3	F217 L181 5 L188 20 L191 4 F235 L247 7 L193 4 L199 23 L189 4 L252 7	20116

COMMENTS: The vehicles generated are as expected. It is interesting to observe that the vehicles generated with CPM=1 and CPM=2 are of the same, general configuration. One question that arises from this experiment is whether the control parameters sum to 500 because it is more efficient or because the period of the first half of the terrain is 500. This question is answered by the next experiment.

5.3.2 Experiment one - variant

# of terrain cycles: 100	*terrain period:* 375	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 75	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: This experiment is identical to Section 5.3.1 except that the "order" of the terrain has been changed. That is, in Section 5.3.1, the first terrain region has a period of 500 and the second region has a period of 375. In this experiment the first terrain region has a period of 375 and the second has a period of 500. It is expected that the results should be the same as in Section 5.3.1.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F248 L148 2 L153 1 L192 1 F252 L133 3 L153 1 L28 1	13107
2	F249 L153 2 L101 1 L182 1 F251 L135 2 L165 2 L18 1	13107
3	F248 L176 1 L102 1 L47 2 F252 L138 5 L116 1 L4 1	13089

run	genome: CPM = 2	fitness
1	F254 L249 1 L253 3 L251 1 F246 L251 3 L251 1 L178 3	19912
2	F255 L174 1 L87 2 L254 1 F245 L253 3 L244 1 L244 5	19899
3	F252 L249 3 L253 2 L246 3 F248 L125 1 L146 5 L246 5	19820

run	genome: CPM = 4	fitness
1	F210 L247 5 L193 2 L193 2 L241 1 F164 L193 3 L246 5 L217 2 L203 16	20505
2	F226 L207 38 L229 6 L226 33 L218 6 F227 L202 4 L204 13 L224 3 L204 3	20317
3	F255 L210 22 L238 6 L182 5 L248 5 L243 5 F245 L202 9 L63 1 L218 11	20260

COMMENTS: The results from this experiment are essentially identical to those in Section 5.3.1. Thus, that the control parameters sum to 500 is because this results in a more efficient terrain traversal and not due to the ordering of the terrain regions.

5.3.3 Experiment two

# of terrain cycles: 75	*terrain period:* 500	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 100	*terrain period:* 375	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: As in the previous two experiments, it is expected that the control parameters of the vehicles, with CPM=1, will sum to 500, the larger of the terrain periods. Because of the large no-step regions, additional legs will be needed on each of the two frames.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F251 L156 1 L111 29 L17 13 L235 105 L220 1 F249 L148 1 L104 6 L27 19 L251 96 L145 12	10158
2	F248 L161 10 L109 100 L233 109 L87 4 F252 L163 16 L108 100 L130 2 L130 2 L161 2	10082
3	F254 L138 31 L141 6 L119 15 L105 2 L132 16 L139 7 F246 L136 2 L136 14 L247 2 L105 7 L94 7 L104 31	10065

run	genome: CPM = 2	fitness
1	F251 L27 1 L234 4 L248 25 L21 5 L123 102 F249 L31 161 L233 6 L233 6 L122 3 L29 11	15183
2	F236 L251 7 L20 2 L126 17 L142 3 L209 7 F139 L28 62 L251 18 L28 17 L93 20	14930
3	F253 L142 32 L126 3 L142 11 L220 95 L236 2 F247 L249 129 L134 5 L250 131 L192 6	14804

run	genome: CPM = 4	fitness
1	F127 L20 13 L245 25 L245 24 F248 L239 16 L37 8 L255 25 L247 16 L223 17	19400
2	F249 L175 19 L223 30 L132 25 L234 11 L238 10 F126 L142 13 L247 5 L131 13 L204 3	18996
3	F177 L196 19 L202 11 L143 3 L223 21 F198 L221 2 L213 75 L149 32 L217 5 L119 9	18575

COMMENTS: The results of this experiment are somewhat surprising in the large number of additional legs per frame that were used. One interesting result is seen in Run 2 with CPM=2. This is the smallest vehicle for the experiments with CPM=2 and its small size is achieved by selecting a gait that is aperiodic with respect to both terrain periods. This vehicle required 201 steps to cross the terrain, as compared to Run 1 which required only 151 steps, but its smaller size boosted its fitness.

5.3.4 Experiment three

# of terrain cycles: 50	*terrain period:* 1000	*step percentage*: 25
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 62	*terrain period:* 800	*step percentage*: 25
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 25231

EXPECTED RESULTS: For all three values of CPM, the control parameters should sum to an “integral” multiple of 1000, the larger terrain period. Integral is quoted because for the case CPM=1, the multiple should be 0.5. All vehicles will require numerous legs: more than four per frame (see the results of Section 5.2.3) are expected, because of the large terrain periods.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F251 L195 87 L102 54 L184 4 L102 32 L148 19 L248 1 F249 L252 105 L242 16 L86 4 L149 3 L230 38 L248 22 L109 11	15726
2	F247 L238 19 L255 1 L251 32 L188 11 L249 1 L200 54 L252 63 F253 L192 84 L154 71 L166 45 L76 53 L186 54 L247 5	15204
3	F248 L196 11 L28 9 L186 86 L179 16 L61 45 L163 27 L163 3 F252 L240 7 L200 45 L91 19 L178 27 L124 41 L204 52 L232 24	15112

run	genome: CPM = 2	fitness
1	F254 L139 19 L239 64 L209 24 L189 28 L29 10 F246 L84 3 L205 29 L195 19 L108 12 L119 9	28090
2	F248 L193 3 L174 8 L222 71 L122 25 L196 9 F252 L100 60 L151 14 L245 60 L98 11 L90 25	27259
3	F253 L196 20 L173 22 L210 37 L162 33 L42 6 F247 L7 9 L210 13 L53 15 L210 38 L60 38 L165 1	27080

run	genome: CPM = 4	fitness
1	F252 L190 21 L169 9 L236 71 L165 48 L251 3 F248 L228 5 L148 10 L173 5 L237 46 L236 9	32782
2	F250 L229 8 L221 17 L108 4 L242 15 L232 45 F250 L161 3 L227 65 L249 7 L129 22 L177 8 L126 18	31986
3	F199 L188 1 L231 42 L164 1 L209 22 L169 20 F201 L193 14 L219 26 L165 4 L164 26 L86 14 L82 17	31109

COMMENTS: The results of this experiment are similar to those of Section 5.2.3 in the sense that the vehicles with CPM=2 and 4 are vastly superior to the vehicles with CPM=1. The reason for this is the large terrain period, which makes negotiating the terrain with small steps, as is the case with CPM=1, inefficient.

5.3.5 Experiment four

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 10	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: Although the nominal terrain period is traversable by a minimal vehicle, such as the resulting vehicles from Section 5.2.1, CPM=1, the fact that the terrain is varying will necessitate additional legs per frame.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F246 L141 0 L16 4 L251 1 F254 L246 0 L137 5 L18 5 L30 2	12623
2	F250 L71 0 L82 2 L227 2 F250 L218 6 L82 2 L58 5 L42 3	12602
3	F251 L130 3 L41 3 L190 1 F249 L239 9 L134 1 L25 3 L155 1	12595

run	genome: CPM = 2	fitness
1	F250 L142 8 L187 11 L254 5 L10 1 F250 L166 7 L163 8 L250 1 L236 1	18346
2	F253 L103 3 L12 4 L254 7 L208 3 F247 L139 3 L30 2 L175 5 L240 16	18336
3	F250 L148 0 L8 3 L250 8 L112 5 F250 L166 3 L225 12 L210 11 L244 1	18336

run	genome: CPM = 4	fitness
1	F176 L118 14 L232 4 L253 13 L246 1 F199 L203 13 L249 2 L238 9 L232 4 L172 13	19487
2	F141 L96 2 L62 1 L157 1 L255 37 F234 L247 6 L181 8 L171 7 L231 2 L113 20	19391
3	F141 L163 0 L240 1 L228 12 L171 7 F234 L208 7 L222 22 L175 6 L239 7 L247 62	19040

COMMENTS: The vehicles generated in this experiment are larger than might be expected, compared to the results of Section 5.2.2. The reason for this is that the step region is "anchored" at its mid-point (see Appendix A) and the ability of the vehicle to walk periodically is disrupted. Had the terrain been generated as discussed in the last paragraph on page 151, the vehicles generated would most likely have been smaller, probably the same size as those generated in Section 5.2.2.

5.3.6 Experiment five

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 100	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: In the previous experiment, the step percentage variation was 10%. In this experiment the period variation is 20%. It is expected that the vehicles developed in this experiment will not perform as well as those from the previous experiment.

ACTUAL RESULTS:

run	genome: CPM = 1	fitness
1	F255 L253 11 L228 15 L239 11 L227 25 L156 22 F255 L197 6 L216 13 L180 19 L235 15 L254 11 L148 23	10532
2	F255 L251 38 L254 19 L255 14 L196 7 L142 3 F253 L251 6 L250 3 L253 14 L255 7 L208 47 L11 6	10524
3	F255 L181 2 L220 9 L228 2 L228 6 L200 19 L190 51 F255 L217 2 L203 2 L217 4 L204 34 L237 3 L9 1	10293

run	genome: CPM = 2	fitness
1	F254 L56 17 L252 12 L236 8 L248 5 L233 23 L228 16 F254 L248 17 L252 8 L248 5 L252 32 L228 16	15884
2	F250 L238 19 L252 1 L248 6 L248 5 L189 39 F250 L235 3 L255 43 L248 7 L227 204	15489
3	F248 L216 13 L130 15 L67 7 L193 8 L239 3 L18 12 F252 L211 21 L195 5 L196 11 L250 18 L212 7 L248 6	15394

run	genome: CPM = 4	fitness
1	F250 L250 4 L230 28 L245 27 L234 5 L6 2 F219 L234 5 L222 12 L180 2 L230 14 L158 13 L110 4	18563
2	F242 L243 4 L237 72 L251 1 L244 4 L145 13 F254 L243 4 L205 24 L233 62 L243 53 L171 15	18363
3	F253 L210 14 L131 14 L105 6 L71 5 L249 8 L167 17 F237 L245 6 L243 22 L241 7 L229 9 L199 53	18324

COMMENTS: The results are as expected, and again, the inability of a vehicle with CPM=1 to efficiently traverse the terrain is observed. This is seen in the smaller vehicle sizes that were generated with CPM=2. Also note that, in general, the control parameters do not sum to the nominal terrain period, indicating that this terrain is aperiodic.

5.3.7 Experiment six

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 10	*s.d. of period variation:* 100	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: This experiment has the most difficult terrain of any of the experiments presented. As such, it is expected that the vehicles will not perform as well

as in any of the previous experiments. The results of Section 5.3.5 can be considered to be an upper bound for the fitnesses that can be achieved with this terrain.

run	genome: CPM = 1	fitness
1	F247 L201 1 L197 14 L197 4 L210 53 L189 7 L99 57 F252 L179 21 L197 122 L246 21 L235 109 L187 33 L234 44	8872
2	F252 L229 24 L242 8 L196 132 L164 42 L177 60 L237 23 L239 53 F251 L244 8 L114 9 L221 60 L205 17 L128 56 L247 37	8554
3	F254 L220 36 L164 67 L190 2 L87 3 L196 25 L198 147 F252 L197 13 L175 7 L186 3 L198 89 L112 15 L152 13 L137 28 L174 13	8537

run	genome: CPM = 2	fitness
1	F248 L175 15 L216 38 L214 47 L53 38 L114 22 L194 122 F237 L179 37 L212 44 L120 8 L175 48 L220 2 L214 45	13486
2	F246 L238 55 L220 77 L193 11 L231 38 L216 40 L102 2 F255 L180 22 L223 27 L57 14 L192 121 L198 25 L223 72	13442
3	F253 L253 71 L134 4 L75 3 L183 4 L121 13 L62 21 F250 L168 80 L196 25 L179 20 L195 15 L198 45 L105 6 L180 159	13254

run	genome: CPM = 4	fitness
1	F237 L252 120 L208 3 L204 16 L252 12 L222 15 F235 L206 10 L115 2 L65 11 L212 12 L150 5 L249 69 L17 11	16918
2	F255 L234 73 L188 52 L221 27 L137 7 L127 76 L210 6 F223 L189 13 L153 23 L238 16 L180 48 L249 7 L214 1	16566
3	F253 L209 1 L224 95 L199 14 L192 4 L223 5 L200 37 F233 L130 5 L106 18 L178 39 L107 41 L227 15 L208 9 L120 5	16395

COMMENTS: As expected, the vehicle fitness values are less than in the previous experiments. The great number of legs needed with each of the values of CPM indicates the extreme variability of the terrain. As in the previous experiment, the control parameters of the vehicles generated do not sum exactly to the nominal terrain period due to the terrain variability.

5.3.8 Summary of second experiments

The results of this set of experiments indicates that for more challenging terrains, simple controllers may not suffice for generating vehicles that perform well. For both types of experiments, the dual period terrains and the probabilistic terrains, the higher values of CPM led to vehicles that performed better. It is also seen that with the more complex terrains, more generations may be required to achieve “optimal” results.

One phenomenon that was observed with the variable terrains is, that occasionally, the fitness of the best individual in the population would show a marked decrease, as would the average fitness of the population. This phenomenon can be likened to a “mass extinction”, a change in the environment that causes all but a few individuals to become extinct. As in nature, the phenomenon causes the variability in the gene pool to be reduced leading to a more homogenous population after the incident. Since those individuals that survive may be lucky, instead of good, there is a chance that the best members of that population are not truly very good. This premise is born out by comparing the standard deviations of the fitness of the best individual in the population across the six experiments:

	CPM		
experiment	1	2	4
Section 5.3.1	205	817	344
Section 5.3.3	2085	2255	3079
Section 5.3.4	4060	6545	2604
Section 5.3.5	614	1087	781
Section 5.3.6	326	556	385
Section 5.3.7	254	380	503

Table 10: Standard deviations of vehicle fitnesses

That the standard deviations for the results of Section 5.3.1 are small is expected since this is a relatively easy terrain. However, for the other five experiments, the standard deviation of the best vehicle fitness for the experiments with varying terrain is much less than that for the fixed terrains. This supports the premise that the varying terrain forces all vehicles to achieve a certain level of capability where the fixed terrain allows certain vehicles to excel. However, the effects of “luck” can be mitigated by testing each individual on a number of terrains and averaging the results. This greatly increases the execution time, but would provide a more realistic evaluation.

5.4 Third experiments - Vehicle level controller

This set of experiments is a repeat of all of the previous experiments. However, instead of a frame-based, fixed controller, these vehicles incorporate a vehicle level controller (VLC). The vehicle level controller is a “program” that outputs a single number that is the distance each frame is to be moved. It is expected that vehicles with a VLC will perform better than those with fixed gait controllers because the VLC has the ability to access three terrain “sensors”. These sensors are:

- s1 - This is a binary sensor that indicates the state of the terrain under the first leg of the current fixed frame. If there is terrain under this leg, the value is 1, otherwise it is zero.
- s2 - This sensor provides a numeric value that indicates the distance from the first leg of the current fixed frame to the first point where the terrain changes state, from no-step to step or step to no-step.
- s3 - This sensor provides a numeric value that indicates the distance from the first leg of the current fixed frame to the first point where the terrain changes state back to the state under the first leg of the current fixed frame.

These models were utilized because they approximate the type of sensor information that could be collected from existing sensors, such as contact sensors (for s1) and stereo vision or active range-finding (for s2 and s3). The distance moved forward is the minimum of the distance set by the controller and the length of the moving frame. In these experiments, the value of the control parameter is not used and has no meaning. It is expected that the difference in fitnesses will be greater for the more challenging terrains because fixed-gait vehicles can only achieve optimal performance on periodic terrains.

The grammar used to generate the VLC is shown in Table 11:

controller →	*function*
function →	*cond_function* | *math_function* | *sensor* | *number*
cond_function →	**{** *function* **>** **0** *function function* **}** | **{** *function* **=** **0** *function function* **}** | **{** *function* **<** **0** *function function* **}**
math_function →	**(** **+** *function function* **)** | **(** **-** *function function* **)** | **(** **x** *function function* **)** | **(** **/** *function function* **)**
sensor →	s1 | s2 | s3
number →	8 bit number

Table 11: Vehicle level controller context-free grammar

For the *cond_functions*, if the result of the conditional operation is true, the first *function* is executed, otherwise the second *function* is executed. For the subtraction operation, the *function* on the right is subtracted from the *function* on the left. For the division operation, the *function* on the left is divided by the *function* on the right, unless the *function* on the right equals 0, in which case the division function returns a value of zero. For STGP purposes, there are two types: functions/sensors and numbers. This allows cross-over between the numeric values in the controller and numeric values used by the vehicle. A controller program is similar to a `LISP` program in the sense that it is a prefix defined mathematical

expression. Such an expression is evaluated starting with the inner-most expressions. For this work, no constraints were placed on the size of the controllers' expressions.

The controllers that are generated are presented as a part of the results, except in those cases where they exceed more than a few terms, in which case they are noted as being complex (see below). To minimize the length of the controller program, the program is examined and automatically trimmed in such a manner as to preserve its semantics. However, due to the choice of representation, there will be program portions that should be reducible that cannot be reduced. Consider the following program fragment: `{(* 120 54) > 0 a b}`. Since 120×54 is greater than 0, code fragment `b` will never be executed, so this statement can be replaced by code fragment `a`. However, the product of 120×54 is greater than 255, the largest number that can be represented by 8 bits, thus this term cannot be replaced and therefore the reduction cannot take place. Where interesting controllers are developed, they will be reported.

The reports of the results of these experiments may make use of the two phrases:

- terrain adaptive - the controller uses the sensor data to modify the distance each frame moves. The majority of controllers developed are terrain adaptive. Those that are not will be high-lighted.
- controller is too complex to analyze - the controller cannot be reduced to some easily understood program. See Appendix B for these controllers.

5.4.1 Experiment from Section 5.2.1 with vehicle level controller

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: Each vehicle should have two frames, and the fitness for the vehicles should exceed those for the best cases previously reported.

ACTUAL RESULTS:

run	genome	fitness
1	F174 L247 9 L210 3 L235 5 L245 5 F206 L253 1 L240 3 L251 4 L233 1 C complex	21484
2	F240 L249 1 L253 1 F133 L248 1 L255 1 C s3	21372
3	F231 L248 1 L253 1 F114 L248 1 L253 1 C s3	21372

COMMENTS: The results do exceed those of the previous experiments as expected. The resulting vehicles belong to one of two classes:

- the vehicle in Run 1 - the controller commands large moves and the vehicle performs well because of its ability to traverse the terrain in only 79 steps. This can be

compared to the best results from Section 5.2.1 in which 77 steps were required. Since the genome of these vehicles can be much larger than for the fixed gait vehicles, (hundreds of nodes versus tens of nodes), it is understandable that the numeric values of the pads and gaps have not been fully optimized. That is to say that this vehicle can still be slightly optimized

- the vehicles in Runs 2 and 3 - these vehicles are essentially optimized versions of the vehicles that were developed in Section 5.2.1, CPM=2, Runs 1 and 2. That is, if the pad widths for the resulting vehicles in Section 5.2.1 were reduced, the fitness values of those vehicles would be the same as those for the vehicles developed in this section. The distinction between the vehicles from the different experiments is not found in the vehicle structure, but in their controllers. While the vehicles in Section 5.2.1 performed well for this perfectly periodic terrain, they would certainly fail on a more rigorous terrain. As will be seen, the controller developed by the vehicles in Runs 2 and 3 will appear frequently in the results, thus indicating that this controller, which commands the vehicle to move by an amount equal to the local terrain period, is a robust controller and these vehicles would be able to handle terrain disturbances.

As mentioned in the introduction, certain controllers are too complex to analyze. As a counter-example, consider the controller developed in Run 2: `s3`. That this controller commands the moving frame to move by a distance equivalent to the local terrain period is easily understood. The controller for Run 1, however, encompasses more than 350 terms and cannot be analyzed in a similar manner. It seems that this controller commands a large, non-terrain adaptive, move, but this is uncertain. This controller program is shown in Appendix B.

5.4.2 Experiment from Section 5.2.2 with vehicle level controller

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: Each vehicle should have two frames, and the fitness for the vehicles should exceed those for the best cases previously reported.

ACTUAL RESULTS:

run	genome	fitness
1	F4 L46 1 L246 1 L254 0 F0 L46 1 L211 1 L252 0 C s3	20062
2	F164 L39 1 L242 1 L248 0 F12 L248 1 L254 0 L234 1 C (reduces to) s3	20062
3	F56 L38 1 L246 1 L251 0 F41 L42 2 L245 1 L245 0 C s3	20049

COMMENTS: Although the results from these experiments do not exceed the best results from Section 5.2.2, they may actually be better results. The reason that these results are not as good is that the number of steps taken by these vehicles developed here (150 for all runs) far exceeds the number of steps taken by the best vehicles developed in Section 5.2.2, (76, 101 and 101 steps respectively for CPM=4). The reason the fitnesses of these vehicles is so high given the larger number of steps is that they are three legs smaller and the pad widths have been completely reduced. (See Section 5.4.12 for further discussion.)

The vehicles developed in Runs 1 and 2 are "optimal" solutions for this problem in the sense that they have the minimum number of legs per frame to allow optimal locomotion. To ensure that two legs contact the terrain in this problem, either the first gap distance must be less than 50 or the sum of the first three gaps (necessitating four legs per frame) must exceed 450. To allow steps of 500, the frame length must exceed 500, but since the first gap is less than 50, three legs per frame are required. Since the third leg is not needed, its pad width should be reduced to zero. This is exactly the configuration seen in Runs 1 and 2.

The designer, in this situation, would be faced with a difficult choice. The results from Section 5.2.2 are better than these results, and since a sensor is not required, probably less expensive to implement. However, these vehicles are more robust to terrain disturbances. It is for this reason that GD is a tool to *aid* the designer, not a tool to dictate designs. This decision must be made by the designer whose understanding of the full scope of the design problem would allow the proper selection to be made.

The result from Run 2 indicates that the controller reduces to `s3`. This reduction was carried out on the original controller (about 15 terms) using knowledge of the system that cannot be easily incorporated into the program that automatically trims the controller code. For example, the parameter s3 is always greater than zero, so multiplying it by a constant yields a term greater than zero. In this manner, the controller in Run 2 was shown to be equivalent to s3.

5.4.3 Experiment from Section 5.2.3 with vehicle level controller

# of terrain cycles: 150	*terrain period:* 1000	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 57277

EXPECTED RESULTS: Each vehicle should have two frames, and the fitness for the vehicles should exceed those for the best cases previously reported.

ACTUAL RESULTS:

run	genome	fitness
1	F181 L251 1 L250 5 L236 10 L249 2 F76 L230 5 L221 3 L244 12 L249 2 C (reduces to) (+ 109 (+ 163 s3))	84628
2	F199 L250 2 L237 14 L255 8 L250 8 F127 L226 11 L244 2 L255 10 L254 8 C (+ (+ 241 (+ (+ 10 s2) s3)) s3)	84568
3	F89 L252 19 L253 7 L243 10 L249 4 F150 L248 29 L253 1 L241 9 L240 5 C complex	84162

COMMENTS: The results from this experiment are approximately the same as those obtained in Section 5.2.3. The reason for the lack of improvement is that these controllers are not purely terrain adaptive in the sense that the commanded moves are not the terrain period. What appears to be happening is that the vehicles are growing to accommodate as large a step as possible, thus requiring larger vehicles. The results from Run 1 indicate that in generation 148, the results started to improve rapidly: a 1.1% increase in fitness for the last three generations as opposed to a 0.2% increase in fitness for the previous 27 generations. Thus, these results may actually exceed those from Section 5.2.3 if the program were run for even a few more generations.

The more interesting results, however, come from a set a vehicles whose fitness is two percent less than the maximum:

run	genome	fitness
1	F158 L250 1 L225 1 F 3 L232 1 L254 1 L254 0 C s3	82881

This vehicle, which is representative of a number of similar “optimal” solutions, presents the designer with the same conundrum as the previous experiment in which the choice must be made between the vehicle with the greatest fitness and a slightly less fit but more robust vehicle.

5.4.4 Experiment from Section 5.2.4 with vehicle level controller

# of terrain cycles: 150	*terrain period:* 1000	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 57277

EXPECTED RESULTS: Each vehicle should have two frames, and the fitness for the vehicles should exceed those for the best cases previously reported.

ACTUAL RESULTS:

run	genome	fitness
1	F52 L88 6 L254 12 L248 13 L212 2 L240 2 F7 L82 7 L255 3 L213 6 L255 7 L227 24 C {(- 64 s2) > 0 (- 230 s3) s3 }	79040
2	F8 L91 18 L253 5 L233 17 L215 4 L228 0 F252 L95 16 L245 4 L233 17 L215 4 L228 0 C s3	78944
3	F105 L66 13 L232 23 L243 6 L236 5 L242 7 F6 L255 8 L238 5 L249 5 L251 16 L125 11 C s3	78495

COMMENTS: The fitness for each of these vehicles exceeds that of the vehicles generated in Section 5.2.4. The reason for the improved performance is the reduced number of steps, 150 steps in this experiments versus 300 steps in Section 5.2.4. The reason for the reduced number of steps is seen in the vehicle configuration. In Section 5.2.4, the second frame took a big step, followed by a short step of the first frame. The summation of these two steps equals the terrain period. By having the first frame take a short step, the first frame could be much smaller (two legs) than the second frame (five legs). In this experiment, each frame moves forward by a distance equal to the terrain period, thus half the number of steps are required. However, the first frame must now be the same size (five legs) as the second. The results in this section may be improved with more generations as many of the pad widths are larger than necessary.

The controller program for each of the vehicles generated is s3. The controller for the vehicle in Run 1 may not appear to be equivalent to s3, however, the value of s2 is equal to zero at the beginning of the terrain, and since the l vehicle moves forward by s3, the value of s2, on this terrain, will always be zero.

5.4.5 Summary of Section 5.2 experiments with vehicle level controller

The results from each of these first four experiments typically show an increase in fitness as compared to the fixed controller experiments. In most cases, the vehicle level controller developed is terrain adaptive, although the meaning of a few of the programs is not always apparent. In certain experiments, the size of the control program was fairly substantial, on the order of 400 elements. This suggests that further improvements to the vehicles may be stymied because of the greater probability of selecting a controller node for cross-over and/or mutation. To mitigate this problem, a more sophisticated routine for simplifying the controller program is required. Simplifying the controller program will also decrease the execution time of the GD program.

As stated earlier, a fixed gait vehicle is capable of traversing a periodic terrain efficiently. In the next set of experiments, in which the terrain is non-periodic, the performance

of the vehicles with vehicle level controllers should show a greater increase in performance over the fixed gait vehicles than the last set of experiments.

5.4.6 Experiment from Section 5.3.1 with vehicle level controller

# of terrain cycles: 75	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 100	*terrain period:* 375	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: The vehicles developed should have greater fitness values than those in Sections 5.3.1 and 5.3.2, and possibly by a large margin because a terrain adaptive controller should be able to modify the gait patterns so “both” terrains are traversed efficiently. It is possible that a vehicle with three legs on each frame, the smallest sized vehicle capable to traversing a such as this, may be developed.

ACTUAL RESULTS:

run	genome	fitness
1	F247 L246 6 L132 1 L182 21 L195 4 L240 5 F53 L124 5 L255 6 L121 1 C complex	20330
2	F131 L205 9 L187 1 L228 1 L192 1 L190 8 F42 L191 4 L244 3 L214 5 L225 2 L228 2 C (- (* (- (* 4 (* 88 250)) 214) 242) s1)	20246
3	F242 L224 14 L228 4 L250 1 L246 5 L241 1 F184 L225 14 L244 4 L218 1 L164 4 L242 17 C (*29 205)	20245

COMMENTS: The performance of these vehicles is approximately the same as those in Sections 5.3.1 and 5.3.2. These used 102, 72 and 66 steps respectively to cross the terrain versus 100 steps from the previous experiments. The reason that so few steps were taken is that each of the controllers essentially reduce to a large constant, thus the vehicle moves each frame as far forward as possible.

Although these top-performing vehicles do not exhibit terrain adaptive controllers, more than half of the vehicles generated do. For example:

run	genome	fitness
4	F159 L118 1 L230 1 L236 0 F 24 L145 1 L163 1 L235 0 C s3	18918

Like several of the previous examples, the fitness of this terrain adaptive vehicle is slightly less, seven percent, than the non-adaptive vehicles, but it is “optimal” in the sense that it represents the smallest vehicle that can efficiently cross the terrain. Notice the great similarity between this vehicle and the ones generated in

Section 5.4.2. This suggests that this configuration with an adaptive controller may be robust across a variety of terrain types.

Because of the great similarity of the experiments in Sections 5.3.1 and 5.3.2, the experiment in Section 5.3.2 was not rerun with the vehicle level controller.

5.4.7 Experiment from Section 5.3.3 with vehicle level controller

# of terrain cycles: 75	*terrain period:* 500	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 100	*terrain period:* 375	*step percentage*: 10
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: The vehicles developed in this experiment should have greater fitness values and be smaller than the vehicles developed in Section 5.3.3 because of the ability of the controller to be terrain adaptive.

ACTUAL RESULTS:

run	genome	fitness
1	F252 L29 1 L235 1 L254 0 F95 L17 1 L255 1 L254 0 C s3	18918
2	F40 L16 1 L248 1 L245 0 F32 L16 1 L249 1 L253 0 C s3	18918
3	F212 L18 1 L254 1 L250 0 F145 L34 1 L254 1 L255 0 C s3	18918

COMMENTS: Given the results of previous experiments, it is not surprising that the fitnesses of these vehicles do not exceed those of the best vehicles generated in Section 5.3.3. Rather, we are seeing “optimal” terrain adaptive vehicles, vehicles that are essentially identical to those developed in Section 5.4.2 and in Run 4 of Section 5.4.6. What is somewhat unexpected is that no solutions of the type found in Runs 1 through 3 of Section 5.4.6 were found. This may be due to the increased ruggedness of the terrain which enables the adaptive solutions to more easily survive the earlier stages of development.

5.4.8 Experiment from Section 5.3.4 with vehicle level controller

# of terrain cycles: 50	*terrain period:* 1000	*step percentage*: 25
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of terrain cycles: 62	*terrain period:* 800	*step percentage*: 25
s.d. of step % variation: 0	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 25231

EXPECTED RESULTS: It is expected that the fitnesses of the vehicles developed in this experiment will exceed that of their fixed controller counterparts. Although this expectation has not been met in all of the previous experiments, the use of a VLC with the more difficult terrains does seem to yield consistently better results.

ACTUAL RESULTS:

run	genome	fitness
1	F114 L249 4 L242 3 L218 1 L185 0 L225 2 F122 L248 4 L240 1 L218 1 L147 2 L233 2 C (reduces to) s3	35074
2	F2 L190 8 L217 2 L252 3 L248 2 L203 2 F170 L223 1 L253 4 L149 0 L195 1 L238 2 C (reduces to) s3	35005
3	F206 L224 2 L239 5 L154 3 L215 1 L237 1 F6 L158 6 L246 4 L238 0 L192 5 L209 4 C (reduces to) {s1 = 0 {s2 = 0 145 (+ s3 1)} {s2 = 0 144 s3 } }	34921

COMMENTS: Although the best solution from Section 5.3.4 required only 100 steps to traverse the terrain, these solutions yielded larger fitnesses while requiring 112 steps. Since the vehicles generated here are the same size, two frame/ten legs, as the best result in Section 5.3.4, the increase in performance is due to a decrease in the average pad width, from 22.7 to 2.0. This reduction in pad width is made possible by the fact that these vehicles match their motion to the terrain period.

Note that the controller in Run 3 may not be as robust as the controllers in Runs 1 and 2. The reason that it works well in this experiment is that s1 equals zero at the initial position, and since the gait is equal to the period, it remains s1 (and the gait remains equal to the period). However, it is possible that small terrain perturbations would cause the vehicle in Run 3 to perform poorly.

The reduction of the controllers for the vehicles in Runs 1 and 2 was done “analytically”. That is, the controller programs were reduced “mathematically,” substituting simpler, but identical expressions, into the program. The controller for Run 3 was reduced numerically. That is, the controller was simulated over the range of values that can be assumed by s1, s2 and s3, and the controller shown above was deduced from the simulation. Although it is possible that the controller shown

above is in error, it is certain that the operative part of the controller commands moves equal to the terrain period.

5.4.9 Experiment from Section 5.3.5 with vehicle level controller

# of terrain cycles: 150	*terrain period: 500*	*step percentage*: 50
s.d. of step % variation: 10	*s.d. of period variation:* 0	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: The results from this experiment should exceed the results from Section 5.3.5 since a VLC should be able to automatically adapt to the varying terrain.

ACTUAL RESULTS:

run	genome	fitness
1	F239 L80 2 L254 4 L251 0 F15 L86 2 L242 7 L254 3 C (reduces to) s3	19833
2	F68 L78 2 L248 2 L234 11 F178 L85 4 L245 2 L254 2 C s3	19767
3	F174 L92 2 L254 11 L255 2 F130 L82 1 L252 2 L247 2 C s3	19761

COMMENTS: The results of this experiment are as expected. These vehicles required 151, 151 and 152 steps respectively to traverse the terrain compared to 101 steps required for the vehicle from Section 5.3.5. Even though many more steps were required, these vehicle have a total of six legs, compared to nine in Section 5.3.5 and an average pad width of 3.0 compared to 8.1. It is probable that with a few more generations, the average pad width for these vehicles would become even smaller. Notice that these vehicle are of the same configurations as those generated in several of the previous experiments, for example Sections 5.4.1 and 5.4.7. This indicates that for this problem domain, this configuration, two frames with three legs per frame and a terrain adaptive controller, is particularly robust.

5.4.10 Experiment from Section 5.3.6 with vehicle level controller

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 0	*s.d. of period variation:* 100	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: The results of this experiments should exceed that of Section 5.3.6, and like the experiment in Section 5.3.6, it is hard to determine a priori if the results from this experiment will exceed that of the last or not. It is expected that these results will not be as good as the previous experiment because the terrain variation in this experiment is 20%, compared to the 10% variation in the last experiment.

ACTUAL RESULTS:

run	genome	fitness
1	F184 L223 1 L248 3 L243 0 F253 L249 2 L240 3 L176 0 L239 4 C (reduces to) (+ {(/ s1 s2) = 0 216 (- (/ 117 s2) 162) } s3)	19259
2	F91 L253 0 L30 4 L244 1 F165 L215 0 L121 4 L242 4 L251 3 C s3	19226
3	F224 L176 1 L226 1 L247 0 F28 L239 3 L163 2 L194 1 L237 0C s3	19136

COMMENTS: As expected, the vehicles generated in this experiment are better than those in Section 5.3.6 This improvement is due to the decrease in the size of the vehicle, seven legs compared to an average of 10.6 in Section 5.3.6 and an average pad width of 1.9 as compared to 10.5. The number of steps to traverse the terrain increased, 151, 151 and 155 steps respectively compared to 82 steps. As was conjectured, the terrain in this experiment was more rugged, thus the vehicle fitness values were not as high.

The controller that was produced in Run 1 is interesting in that it is not purely periodic, like the controllers in Runs 2 and 3, yet for this variable terrain, it produced equally good results. This leads to the notion that the simple controller, s3, may not necessarily be the best controller for all possible terrains, although, in general, it does appear to be very good.

5.4.11 Experiment from Section 5.3.7 with vehicle level controller

# of terrain cycles: 150	*terrain period:* 500	*step percentage*: 50
s.d. of step % variation: 10	*s.d. of period variation:* 100	
# of generations: 150		*max possible fitness:* 14295

EXPECTED RESULTS: Since the fitness values of the vehicles in Section 5.3.7 were less than the fitness values achieved in either of its two preceding experiments (since the terrain is more rugged), it is expected that the fitness values of the vehicles generated in this experiment will be less than in the previous two. However, the fitness of the vehicles developed should exceed that of the vehicles developed in Section 5.3.7.

ACTUAL RESULTS:

run	genome	fitness
1	F103 L76 2 L166 2 L254 1 L250 1 F152 L119 2 L242 3 L209 2 L216 8 C (reduces to) { (/ 44 {s2 > 0 s3 220 }) = 0 s3 62 }	18793
2	F40 L75 2 L223 4 L235 1 L230 0 F140 L82 6 L221 12 L255 0 L237 5 C s3	18709
3	F104 L68 5 L251 1 L204 1 L223 5 F136 L64 2 L251 1 L209 2 L246 1 C s3	18693

COMMENTS: These vehicles performed significantly better, eleven percent, than those developed in Section 5.3.7 even though many more steps were required to traverse the terrain. These vehicles required 145, 145 and 148 steps respectively to cross the terrain as compared to the best vehicle from Section 5.3.7 which required only 75. By adopting terrain adaptive controllers, these vehicles were able to efficiently traverse the terrain with much smaller vehicles than in the fixed-gait experiment, eight legs compared to twelve and an average pad width of 2.6 compared to 23.8. This improvement is made possible because of the terrain adaptive controller.

Although the controller developed in Run 1 may appear strange, it is essentially equivalent to s3. If s2 equals zero or if s2 is not equal to zero and s3 is greater than 44, the controller expressions reduces to s3; otherwise it reduces to 62. If s3 is less than 44, by stepping a distance of 62, the vehicle will still be moving forward effectively since in this case, the distance moved forward exceeds s3.

Figure 25 shows the best vehicle for CPM = 1, 2, and 4 from Section 5.3.7 and the vehicle from Run 1 traversing a representative terrain. The terrain adaptive behavior of the vehicle with the VLC is clearly apparent. Figure 25 graphically demonstrates the added advantage of the VLC. (Since Figure 25 is a scale drawing, some of the feet may not be clearly visible because they are quite small.

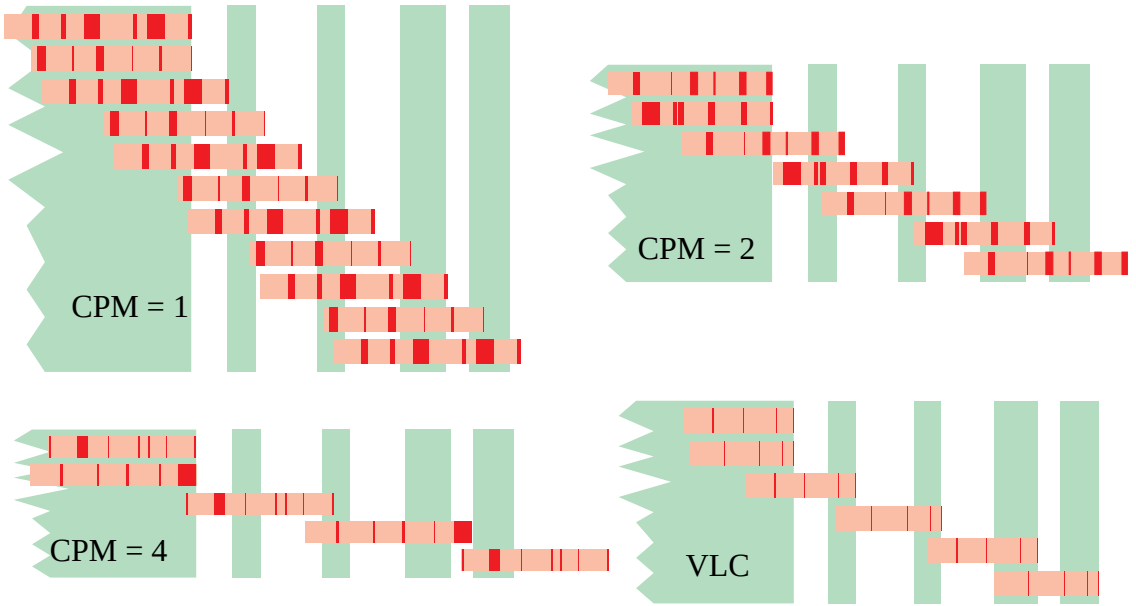


Figure 25. Vehicles traversing stochastic terrain

5.4.12 Summary of third experiments

The results of these experiments show the benefit of the simultaneous development of a capable controller and the vehicle. Although the fitness values did not always exceed those obtained with the fixed gaits, in general, they were better. Those cases where the fixed gait

machines had better results were on the simple, periodic terrains. That the fixed gait machines perform well on this type of terrain is expected. However, on the aperiodic terrains, the machines with the VLC significantly out-performed the machines with fixed gaits.

One explanation for the poorer performance of the vehicles with VLCs as compared to vehicles with fixed gaits in certain experiments, Section 5.4.3 for example, is the sensors made available to the VLC. In every experiment in which the fitness of a vehicle with a terrain adaptive controller was exceeded by a vehicle with a non-terrain adaptive controller, the vehicle with the non-terrain adaptive controller took longer steps. This suggests that making sensors s4 and s5 available to the controller, where s4 and s5 are the logical extensions to s2 and s3, the vehicles with VLCs might use this sensor information to take larger steps, thus resulting in vehicles with improved fitnesses.

An interesting set of additional experiments would be to develop fixed gait vehicles and vehicles with VLC for very rugged terrain. In this situation, it is possible that the vehicles with the VLC may do poorly. Based on the discussion of extinction, Section 5.3.8, only a few difficult terrains are needed to limit the genetic diversity of the population. Those vehicles that have poorly tuned terrain adaptive controllers may be more sensitive to early terrain variations, thus tending to eliminate vehicles with this type of controller from the population; whereas those vehicles with fixed-gait controller or VLCs that command large moves may be able to successfully traverse the terrain. Over the course of the experiment, vehicles with terrain-adaptive controllers may be eliminated from the population, although, based in the results in Sections 5.4.9 through 5.4.11, the vehicles with the VLC would be expected to perform better. An analogy to this situation would be rock-climbing: the climber who uses pitons will traverse the rock face every time, but a great deal of time and effort is required to set the pitons. This climber is analogous to the vehicle that commands a large move. The climber who uses chocks, while requiring more time to learn to use the technology, will eventually be able to traverse the same rock face more quickly than the piton user because using chocks is more efficient.

This brings up two concepts: learning and higher level planning. One way to improve the performance of the vehicles with the VLC would be to train the controllers on easy terrains before using them on the more difficult terrains. This process may be seen to be akin to child development in the animal world, where a youngster is first “taught” to traverse nominal terrain before graduating to more rugged terrains. One way to test this premise would be to start with a fixed terrain for some number of generations, then gradually start varying the terrain until the desired terrain parameters are achieved. By using this method, the terrain adaptive controllers should have time to develop before being subjected to rugged terrains.

Higher level planning requires the use of more complex sensors and completing plans of greater temporal duration to achieve success. For example, the successful hurdle runner is one who plans the jumps many steps ahead of the barrier, and not when the barrier is one

step away. Testing these higher level planners is outside the scope of this thesis work, but is considered to be an important part of any future work, Section 7.2.

5.5 Summary of stepping stone walker model

The results of this body of experiments indicates that the GD methodology works. In a number of experiments, the optimal solutions, based on the specified evaluation function, were found despite the fact that only a minute fraction of the design space was searched. For those experiments with a fixed-gait controller, Sections 5.2 and 5.3, the number of vehicles in the design space exceeds 10^{1000}. The design space is even larger for the experiments with vehicle level controllers. However, despite examining fewer than $1.8x10^{6}$ vehicles, reasonable results were obtained.

As stated previously, the results of GD are typically not going to be *the* ideal solution to a problem, but rather a set of good alternatives. To improve the results, a post-processor that takes the results generated and presents them to the designer, possibly in a graphical form, would allow the designer to see how the solutions are behaving and allow the designer to experiment with modifications in an attempt to improve the design. This tool should also allow the designer to evaluate manually developed concepts.

➤ The contention that the simultaneous design of the controller with the system leads to better design also appears to be supported by the results of these experiments. First, it is unclear how these experiments could have been carried out if the controller were not an integral part of the design. It is probably safe to state that in all design exercises, the means of control of the object being designed are developed along with the object. However, the controller design may frequently be assumed or highly restricted, thus limiting the potential of the design. Second, the limiting affect of the controller design is seen in comparing the results of the experiments in Section 5.4 with those of Sections 5.2 and Section 5.3 — by removing the fixed gait restriction, the vehicles with the more general gait planners were able to outperform the vehicles with fixed gaits. The vehicles with the general gait planners also had access to terrain sensors, which indicates the desirability of including sensor models into the artifact design problem as well.

5.6 Extension to rolling vehicles

The work presented in the previous sections of this chapter showed how GD could be applied to the design of an abstracted model of a walking vehicle. In this section, the extension to a similarly abstracted model of a rolling vehicle is presented. This presentation does not include any experimental results, rather it serves to show that the methodology developed is not restricted to a particular class of problem.

Like the stepping stone walker, the stepping stone roller is an abstraction of an actual wheeled vehicle. Like the stepping stone walker model, the analysis for the stepping stone roller model is a kinematic analysis. A kinematic analysis is chosen for practical reasons

— it is easier and quicker to perform than a dynamic analysis. GD does not place any requirements on the type of artifact analysis performed.

The stepping stone roller model is comprised of *n* wheels, each of which has a possibly unique diameter *d* and is separated from the next wheel by a distance *h*. A sketch of a stepping stone roller is shown in Figure 26 and the grammar for generating the vehicles is shown in Table 12.

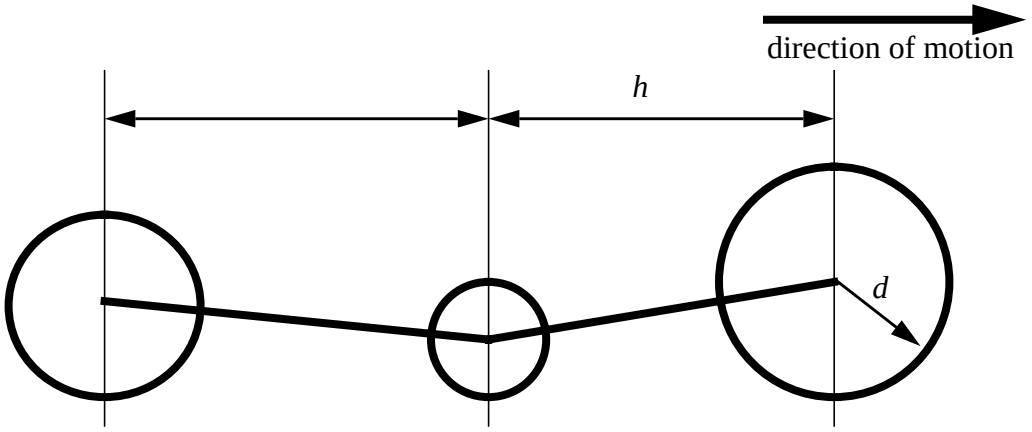


Figure 26. Stepping stone roller vehicle

vehicle →	*wheels*
wheels →	*wheel connector wheels* | *wheel*
wheel →	**W** *number*
connector →	**C** *number*
number →	8 bit number

Table 12: Stepping stone roller vehicle context-free grammar

The *number* associated with *wheel* is the wheel diameter. The *number* associated with *connector* is the separation distance. A vehicle with three wheels would be represented as:

W d1 **C** h1 **W** d2 **C** h2 **W** d3

The stepping stone roller uses the same terrain model as the stepping stone walker.

The evaluation of the stepping stone roller is a two step procedure. The first step ensures that the vehicle is viable, the second steps evaluates the model. A vehicle is viable if every separation distance is larger than half the sum of the two adjacent wheels. This requirement ensures that no wheels overlap.

The stepping stone roller is evaluated in a similar manner to the stepping stone walker. The evaluation is comprised of two components: the vehicles performance on the terrain and the vehicle's dimensions. The terrain performance is determined in the following manner. The vehicle is initially placed on the terrain with the center of the leading wheel

aligned with the world origin. The vehicle is then moved across the terrain, one terrain point at a time. At each point, there must be a portion of the step region beneath at least two of the wheels. Figure 27 shows a vehicle progressing past one set of terrain obstacles, but being halted by obstacles further along the path.

Figure 27. Stepping stone roller vehicle

In addition to measuring the distance traversed, the vehicle's parameter also effects its fitness evaluation. Like the stepping stone walker, smaller vehicles are considered to be better. Thus, the fitness evaluation will favor vehicles with fewer wheels, smaller wheels and smaller separation distances. These terms can be combined in a manner similar to those shown in Table 6 and Equation (12).

Since this vehicle cannot react to changes in the terrain, this evaluation is deterministic and a controller is not required. Were a dynamic analysis performed, the controller could be used to set the speed of the vehicle in order to cross the gaps — larger gaps would require greater speed. To prevent the controller from simply setting the greatest allowable speed, a speed dependent penalty would need to be assessed. This penalty is analogous to viscous friction power losses. The evaluation function for the dynamic model might also include a term to minimize the time of traversal.

This brief discussion should serve to show that GD can be applied to classes of vehicles other than the stepping stone walker. In this example, the same techniques can be applied to a simple rolling vehicles. Chapter 7 discusses other future applications of GD for vehicles.

6 Synthesis of planar mechanisms

➤ This chapter presents a method for synthesizing planar mechanisms using the GD methodology. This problem was selected for a variety of reasons, including:

- the problem is well understood and there exist well-known, classical solutions to certain problems.
- the metrics for determining the quality of the solution are more concrete.
- kinematic mechanism can be used as legs for walking vehicle, thus an extension of this work might combine the grammar developed in Chapter 5 for the walking vehicle with the grammar developed in this chapter for planar mechanisms to produce a grammar that describes vehicles with more detail than in Chapter 5.

Unlike the design of walking vehicles, the analysis and synthesis of planar mechanisms has been actively studied since the last century. The pioneering work in this field was explored by Reuleaux in the 1870's, and is still a field of active research. One of the great successes of this work was the development of the Burmester theory, discovered in 1888. This theory can be used to predict the number of different possible mechanisms that can be constructed to pass through a set of specified points (referred to as precision points). In brief, this theory shows that a four-bar mechanism can be built that will pass through up to five specified precision points, or through fewer points, but in a specified order. Although the solution is non-trivial, it is a closed form solution that can be implemented graphically or on a computer. [87] [91]

The mechanisms produced using Burmester theory are "perfect" in the sense that they pass thought the specified point exactly. But what if a four-bar mechanism is required to pass through more than five points? Burmester theory tells us that such a mechanism cannot be built, (except in some limited, special cases, such as symmetry. See the discussion on point position reduction in [37].) But what if the designer is willing to relax the inherent exactness of the Burmester theory results and accept a mechanism that comes "close" to the desired point? Also, what if a four-bar mechanism is not the appropriate solution to a problem, but a higher order mechanism is?

➤ This chapter is divided into two sections. In Section 6.1, a GA for four-bar mechanism synthesis is presented. The reason for presenting this work is to describe the evaluation function used, to compare and contrast the genetic technique with the classical techniques and to demonstrate its capabilities on some test problems. In Section 6.2, the GD methodology is applied to the general problem of planar mechanism synthesis. Through the use of GD, the type, number and dimension synthesis problems are all solved simultaneously to yield feasible solutions.

6.1 GA for four-bar mechanism synthesis

This section describes a genetic algorithm solution to the planar four-bar synthesis problem. The key component of this work is the evaluation function, described in Section 6.1.2, which is also used for the general planar synthesis problem, see Section 6.2. Before describing the evaluation function, the genetic representation of the four-bar is examined in Section 6.1.1. Section 6.1.4 describes some test results for a number of illustrative problems using the GA.

6.1.1 Four-bar representation

Figure 28 shows a sketch of a four-bar mechanism. This mechanism can be described by nine independent variables which are illustrated in the sketch. The small triangles represent revolute joints affixed to the ground, the open circles represent revolute joints that join two links and the filled circle is the coupler point. Link 1 is the input link, the crank; the large shaded triangle is link 2, the coupler; link 3 is the output link, the follower; and link 4 is the ground. Also shown are the offsets from the world origin and an angle that represents the angular offset of the coupler point.

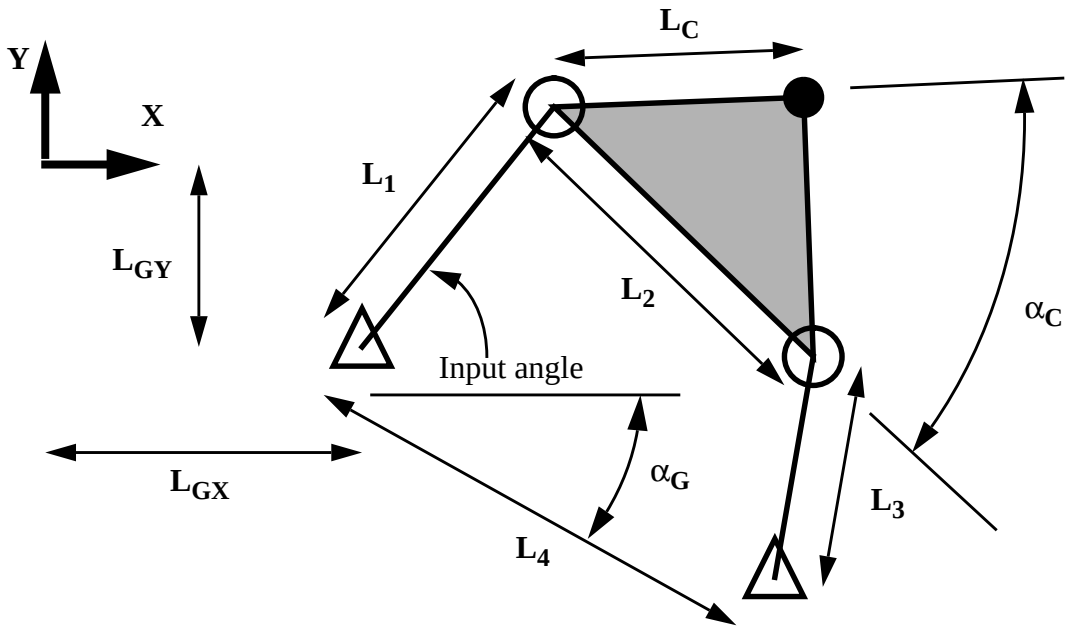


Figure 28. Four-bar mechanism

When representing an object as a bit string, the easiest representation is one that uses the same number of bits as predefined computer types. In the `C` programming language, the basic types are `char`, `short int`, `int` and `long int`, which are represented by 8, 16, 32 and 64 bits respectively. For this problem, representing the variables with eight bits did not seem to offer sufficient granularity to ensure good results. For example, if the largest length representable is 10, the granularity is approximately 0.04. By representing these lengths with 16 bits, the granularity is reduced to approximately 0.0002.

When representing a real quantity, such as length, as a bit string there are two important limitations: the range of values that the variable can assume and its granularity. If a solution requires values larger than the largest representable value, the solution cannot be found. If the solution is sensitive to small variations in parameter values, it is possible that good solutions may only be found if the number of bits is increased. For this problem, using 16 bits instead of 8 squares means that the number of possible mechanisms increases from 4.7×10^{21} to 2.2×10^{43}. In either case, the number of possible mechanisms is so large that an exhaustive enumeration of the space is impossible.

6.1.2 Kinematic analysis

There are numerous techniques for the kinematic analysis of a four-bar mechanism. One such technique, in closed form, is given in [38]. Although this closed form solution is the most efficient solution, it cannot be generalized to more complex mechanisms, thus, this method is not used. In [92], the authors present an algorithm for solving general, planar mechanisms. This is the algorithm used for both the four-bar GA synthesis and the general GD planar synthesis problems.

The algorithm for the solution of a general planar mechanism appears on pages 181-187 of [92]. Since the algorithm is well described in the book, only a brief summary of it is presented here. The algorithm is based on the Newton-Raphson numerical method for iteratively finding the zeros of equations. To implement the method outlined, the user need only supply the vector-loop equation(s) that describe the mechanisms being analyzed. The solution is found by reducing the error in the loop closure equation by using the Jacobian matrix of the mechanism. Although an iterative approach may seem to be inefficient, in practice, the equation is found to converge within five iterations - if a solution exists.

6.1.3 Evaluation function

The strength of using this GA method for solving the four-bar synthesis problem lies in the ability to fashion an evaluation function to the needs of the designer. For instance, although Burmester theory shows that there exist four-bar mechanisms that pass through five specified points, the order in which the mechanism passes through those points cannot be guaranteed. By using an appropriate evaluation function, a mechanism can be found that will almost pass through the five points, but in the order specified. The word “almost” is used because the solutions found cannot violate Burmester theory, thus the solutions are close approximations to the desired mechanisms. It should be noted that for certain cases (for example, five precision points with a prescribed order, or more than five precision points), the desired four-bar mechanism probably cannot be actualized.

Although a wide variety of evaluation functions are possible, this work focuses on one basic function. This function compares points along the path swept out by the coupler point of the candidate mechanisms with the desired points. The closer the points on the swept path come to the specified points, the higher the fitness of the mechanism. There are

two potential problems with this method. First, the points along the swept path must necessarily be discrete points. This leads to the possibility of having the desired points fall between the path points, causing a good configuration to be evaluated poorly. This problem is somewhat alleviated by selecting a finer granularity for the swept points, but this will increase the evaluation time. In this work, the mechanisms were evaluated at five degree increments of the input crank. The second problem is that in GA, individuals that perform better are scored higher than individuals that do not perform as well. However, with this scheme, point pairs that are closer, that is are better approximations, would receive numerically smaller fitness values. Three schemes for changing these small numerical values into higher fitnesses are presented below:

- Sum the difference of the position error and a user specified constant:

$$f = \sum_{i=0}^{\text{pts}} K - e_i \tag{15}$$

  where f is the fitness, K is the constant, pts is the number of specified points and e_i is the smallest distance between specified point i and a point on the mechanism's swept path. (Actually, for computational efficiency, e_i is the square of the smallest distance.) If $e_i > K$, the right hand side of the equation (for point i) is set equal to zero. The problem with this scheme is that if K is large enough so most points evaluate to a non-zero value, then K is also probably large enough that the function cannot clearly distinguish between swept points that are close to the specified point.

- Take the reciprocal of the sum of the position errors:

$$f = \frac{1}{\sum_{i=0}^{\text{pts}} e_i} \tag{16}$$

  This scheme eliminates the arbitrary constant and does correctly reward a mechanism for minimizing the error at each of the specified points. However, in practice, this scheme failed to produce good results because once several points were satisfied, the system would fail to satisfy the remaining points because moving to alternate solutions requires a decrease in the evaluation function. In other words, this scheme is prone to finding local maxima.

➤ - Take the sum of the reciprocals of the errors:

$$f = \sum_{i=0}^{\text{pts}} \frac{1}{e_i} \tag{17}$$

At first glance, this scheme seems to offer little advantage over the previous scheme — and simply used as shown in Equation (17), there is no advantage. However, reformulating Equation (17) as

$$f = \sum_{i=0}^{\text{pts}} \begin{cases} e_i \le C & \dfrac{1}{C} \\ e_i > C & \dfrac{1}{e_i} \end{cases} \tag{18}$$

and changing the value of C dynamically does offer significant improvement over the previous two schemes. To avoid the trap of the previous scheme, the value of C must initially be “large”. An initial value of $C = 0.10$ was found to produce good results. This relatively large value of C allows the system to explore other solutions because moving away from an existing solution cannot greatly reduce the fitness function. Once a user specified number of mechanisms satisfy $f = \text{pts}/C$, the value of C is updated using $C \leftarrow C/k$, where k is an user specified constant. Experimental results indicate that a relatively small value of k, around 1.2, yields the best results. This is the evaluation function and value of k used for all of the planar synthesis problems discussed.

These three evaluation functions all have one thing in common — they simply measure the distance between a user specified set of points and points generated by trial mechanisms. Other evaluation functions are also possible, depending on the specific requirements of the task to be solved. For example, although Burmester theory can tell you if a mechanism can be synthesized to pass through a set of five points, there is no way to know, and no way to specify, the order in which the points are visited. Using a modification of the scheme presented here, the relative ordering of the points can be specified. Another possible requirement would be to specify the relative crank angle as the mechanism passes through five or more specified points. Neither of these constraints can be explicitly satisfied using Burmester theory, however, the genetic approach outlined here is capable of dealing with them.

6.1.4 Implementation and test results

The GA technique for four-bar synthesis is implemented as a standard GA, Section 3.1, with two differences. First, this implementation incorporates elitism - the preservation of some user specified number of the best individuals into the succeeding generation. Second, this implementation uses a modified decimation procedure for generating the initial population. Decimation, as usually implemented, requires the generation of a larger than required initial population, followed by the selection of the best members of the larger population to form the actual initial population. In this implementation, an individual is not accepted into the population unless its fitness is non-zero. This eliminates from consideration those mechanisms that cannot be assembled.

Another benefit of this GA technique over Burmester theory is that with the GA technique, the user can specify the mobility class of four-bar mechanism created. For example, a mechanism created using Burmester theory might be double-rocker mechanism — one in which the crank cannot rotate through 360^{o}. Although such a mechanism is indeed a valid solution to the specified problem, in certain instances, implementing such a mechanism is difficult because the input may be required to be continuously rotating. In this case, a four-bar mechanism would be required to convert the continuously rotating input into the rocker motion input, thus converting the four-bar mechanism into a a six-bar mechanism. With the GA technique, mechanisms with continuously rotating inputs can be specified by assigning a fitness of zero to all mechanisms that do not have continuously rotating inputs. Similarly, the evaluation function can be formulated in such a manner to only accept double-rocker mechanisms. This specification of the mobility class is subject to the resolution at which the mechanism is being evaluated.

One of the difficulties inherent with the GA technique for the synthesis of four-bar mechanisms is that there is no guarantee that a four-bar mechanism can be built within the confines of the representation that will pass through a set of specified points. If the technique fails to find a solution for a set of points, it could be due to a failure of the technique or that no such solution exists. To show that the technique works, therefore, a set of points must be known to have a solution that is expressible within the confines of the GA representation used. To that end, the experiments reported in this section can be divided into three groups. The first four experiments use sets of points that are known to have solutions in the representational space. This is known because the points were generated by randomly generating a mechanism, using the same representation, and selecting, at random, some of the coupler points.

The second set of experiments attempts to synthesize four-bar mechanisms that follow special paths. The first set of examples is based on an involute curve; the second set on straight line motion.

The third set gives two examples - one showing a strength of this methodology and the other showing how it can fail. The first experiment synthesizes a mechanism by specifying a set of 24 points. These points are known to lie in the solution space because they are selected in a similar manner to the points in the first set of experiments. This experiment shows how the methodology can be used to specify the entire path for a mechanism. The second experiment highlights the need to add scaling to this methodology, because without scaling, the results produced can be erroneous. In this context, scaling refers to the ability to automatically modify the bit string to length translation to be most appropriate for a particular experiment. Table 13 lists the experiments performed:

	number of specified points			
specified points formed by:	3 or 4	*5*	8	11 or 24
four-bar mechanism	Section 6.1.4.1	Section 6.1.4.2	Section 6.1.4.3 Section 6.1.4.4	Section 6.1.4.10
involute curve	Section 6.1.4.5	Section 6.1.4.6	Section 6.1.4.7	——
straight line	Section 6.1.4.8	——	——	Section 6.1.4.9

Table 13: GA four-bar mechanism synthesis experiments

For each experiment (except for the straight-line motion experiments), the desired curve and the set of points to be passed through is shown. This plot is labeled “input data”. Then, the coupler curves for the three best generated solutions are shown. For each of the solutions, the mechanisms generated are shown, along with the maximum possible error. For each experiment, the scales of the plot are identical. Figure 29 is a legend for the plots in the following sections:

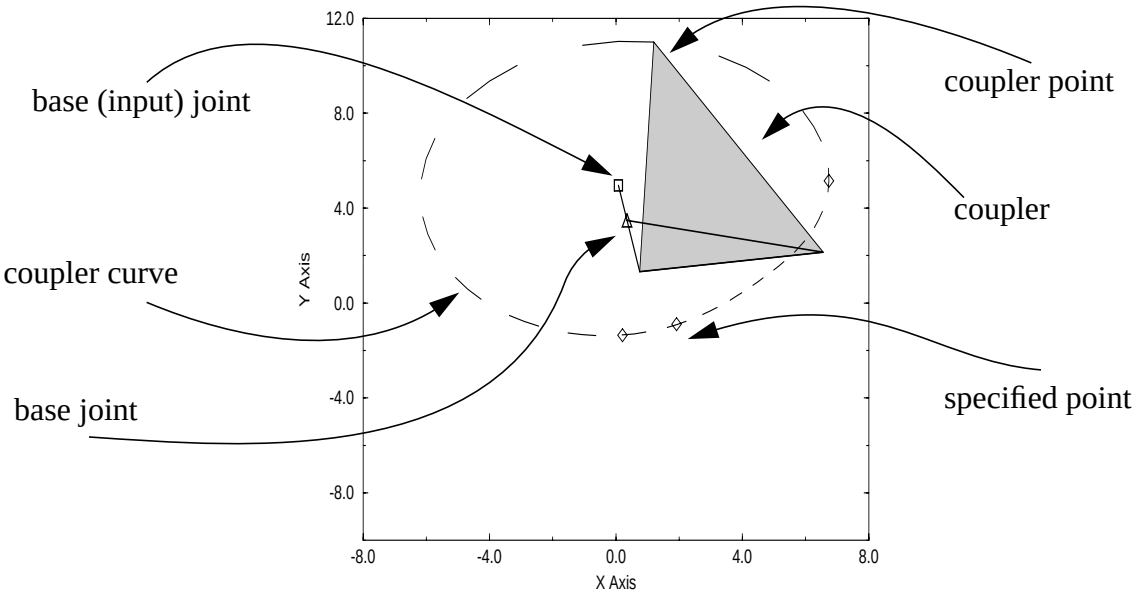


Figure 29. Interpretation of four-bar synthesis plots

The width of the dashes of the coupler curve shows the approximate velocity of the mechanism at that point. Since each dash represents 10 degrees of motion of the input link, wider dashes represent faster movement. Each mechanism is shown with the input link at zero degrees. The reason that the input link is not oriented with the world X-axis is that the mechanisms are placed in the world (position and orientation) after the motion of the mechanism has been simulated. The location of the coupler point on the dash indicates the

direction of movement for the coupler point of the mechanism. The coupler point shown in Figure 29 is rotating counter-clockwise.

The maximum possible error can be calculated for any solution whose fitness exceeds $(\text{pts}-1)/C$. If the fitness satisfies $(\text{pts}/C) < f < (\text{pts}-1)/C$, it is assumed that one point has all of the error. For example, consider the case where $\text{pts} = 5$, $C = 0.1$ and the fitness is equal to 42.0. This fitness is assumed to be achieved by having four points with $e_i < C$, and the last point has $e_5 > C$. For this example, e_5 can be found: $1/e_5 = 42 - 40 \quad \rightarrow \quad e_5 = 1/2$. Since the error is the square of the distance, the maximum distance error is $\sqrt{1/2} = 0.707$. In the case that $f = M \equiv \text{pts}/C$, the maximum distance error is given by $\sqrt{C}$. The formula for calculating the maximum possible distance error, *d*, given *f*, *M*, and pts is

$$d = \left(f - \frac{(N-1)\,M}{N} \right)^{-1/2} \tag{19}$$

If $f < (\text{pts}-1)/C$, the maximum error cannot be calculated since the number of points whose error is less than *C* cannot be determined and the maximum error is unbounded. It is important to note that the maximum possible error will likely increase in the generations immediately following an adjustment of *C*. For example, if in generation j, $\text{pts} = 5$, $C = 0.1$ and $f = 50$, the maximum error would be 0.32, and in generation $j + 1$, with $C = 0.1/1.2$, if the mechanism does not change, the new maximum error would be 0.71.

The question that arises from the discussion of the maximum possible error for a mechanism, is this: How does the reported value of the maximum possible error corollate to the quality of the mechanism? This is a difficult question to answer for a number of reasons. First, the number presented is the maximum *possible* error, meaning that the actual largest position error may be significantly smaller. The reason for this is that the distance metric is between two sets of discrete points — it is not between a curve and a set of points. Second, as currently implemented, the error is absolute, not relative. For a mechanism with links 10 units long, an error of 0.1 units may be acceptable. For a mechanism with links 1 unit long, this may be unacceptable. Finally, when attempting to find a mechanism to pass through more than five points, an exact solution cannot exist (except in special cases). Thus, the maximum possible error may in fact be the smallest achievable error. The answer to the question must therefore be this: It is up to the designer to decide whether the error is acceptable or not.

The GA parameters shown in Table 14 were used in the following experiments. These values were not generated by a meta-GA for two reasons. First, a reasonably sized population is required and the computing resources necessary were not available. Second, the results from the meta-GA were shown to be more sensitive to the random seed than the parameters themselves, unless each set of parameters is tested with a number of random

seeds, see Section 5.4.12. Thus, the parameters used are those commonly used for this type of work. The one exception is the relatively high mutation rate. This was chosen because of the large size of the genetic space:

cross-over probability	0.5	number of elitists	10
mutation probability	0.01	population size	200
required input rotation	360^o	number of generations	150
tightening parameter, C	5	fitness multiplier	3.0

Table 14: GA control parameters for four-bar synthesis

The power of the GA over random searching is evident when one considers how few mechanisms are really examined. The maximum number of distinct mechanisms that are evaluated is 900,000 (30 experiments with a population of 200 for 150 generations). However, many fewer are unique because of the inclusion of elitists and the low cross-over probability. These 900,000 mechanisms represent but 4.2×10^{-35} of a percent of the mechanism space. And even this small number of mechanisms greatly exceed the number of mechanism coupler curves available in the classical texts, such as the atlas compiled by Hrones and Nelson. This atlas, while representing a tremendous effort, only shows curves for 7300 different mechanisms. As will be seen in the following experiments, sometimes examining 900,000 mechanisms yields only marginally acceptable results.

6.1.4.1 Three specified points - generated by four-bar

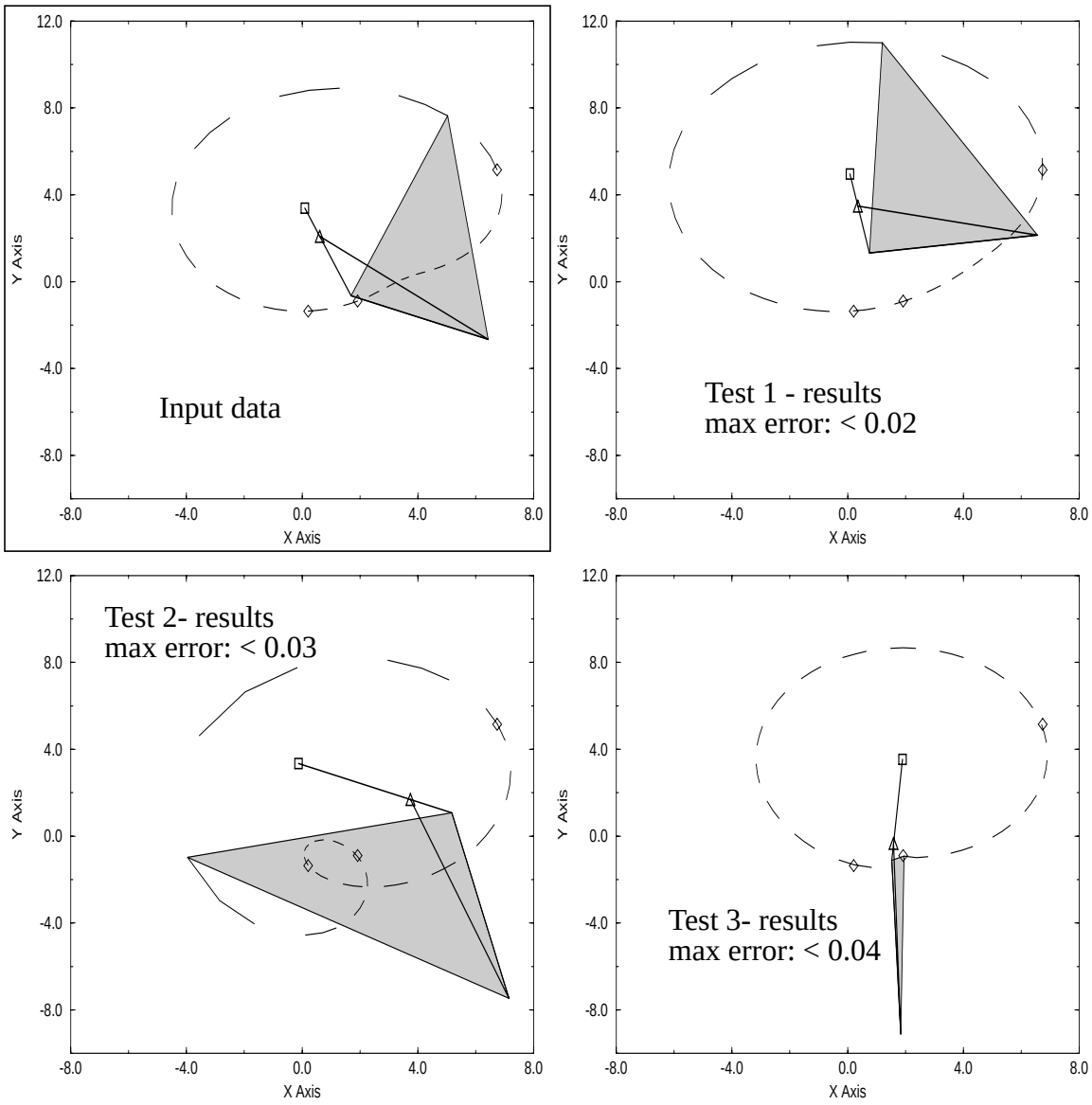


Figure 30. Planar four-bar synthesis, three specified points

Burmester theory states that there are an infinite number of four bar mechanism configurations that can be used to pass through three specified points. The results of this experiment show three rather different solutions to the same problem. This range of solutions suggests that specifying only three points may insufficiently constrain the mechanism configuration. It is interesting to note that all of the mechanisms are drag-link mechanisms. Some might object to these solution because exact solutions to three specified points should be found. However, finding those solutions assumes that the task is to fit a curve to the three points. The methodology developed here matches finite sets of discrete points, therefore it is not expected that an exact solution will be found.

6.1.4.2 Five specified points - generated by four-bar

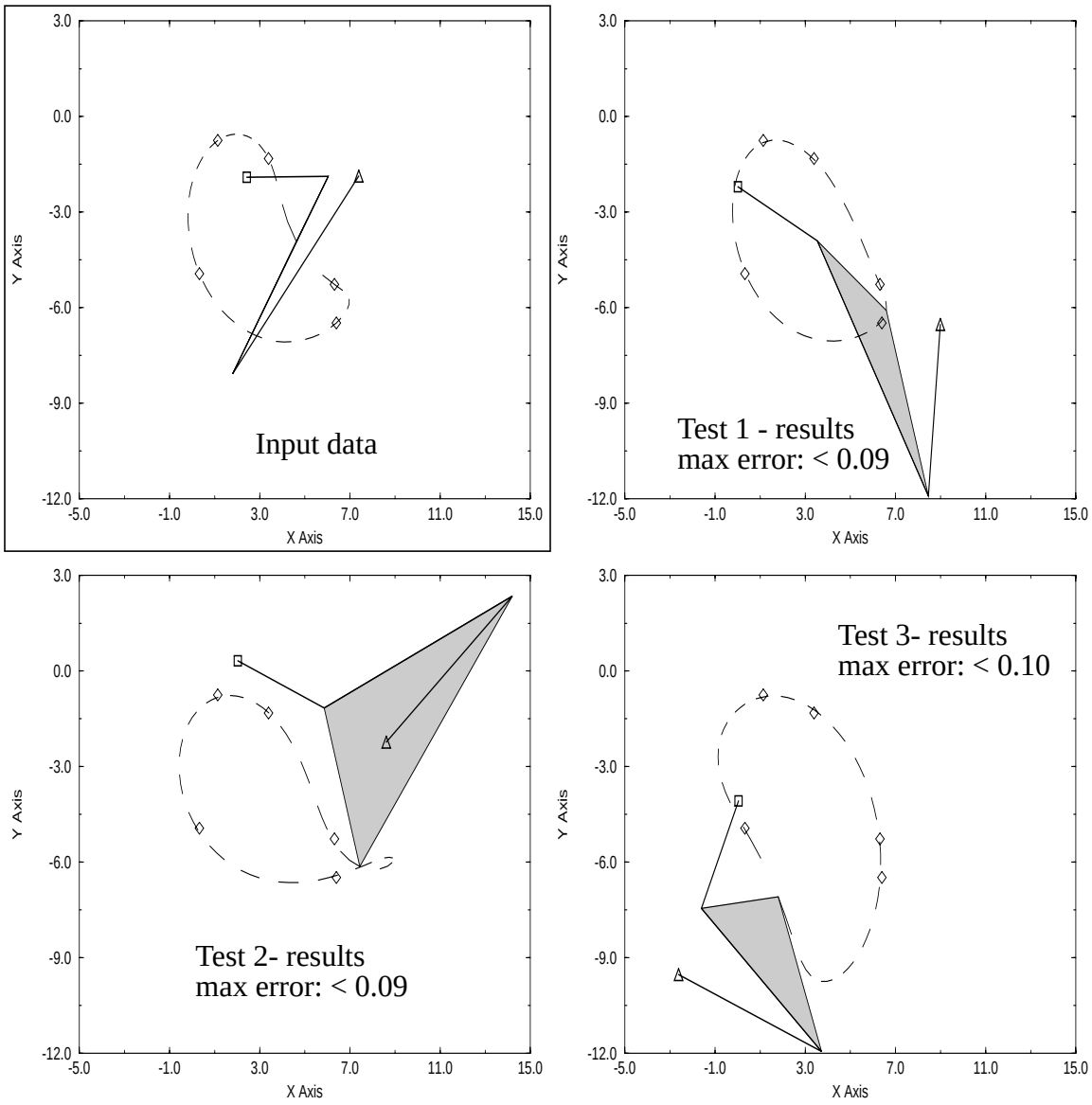


Figure 31. Planar four-bar synthesis, five specified points

There are two things to note about this set of experiments. First, the maximum error for this set of experiments is greater than that of the previous experiment. This is expected since this experiment is a more difficult problem, and more generations would be required to achieve similar results. Doubling the number of generations did result in a 10% improvement in accuracy for Test 1. Second, the shapes of the coupler-point paths of the solutions is more similar to the shape of the input data coupler path than the previous experiment. The reason for this is that there are more constraints, thus there are fewer mechanisms in the design space that can pass “through” these points. In this experiment, all of the mechanisms are crank-rocker type mechanisms. Note that the plot for the input data is correct - the coupler is thin and not clearly distinguished.

6.1.4.3 Eight specified points - generated by four bar

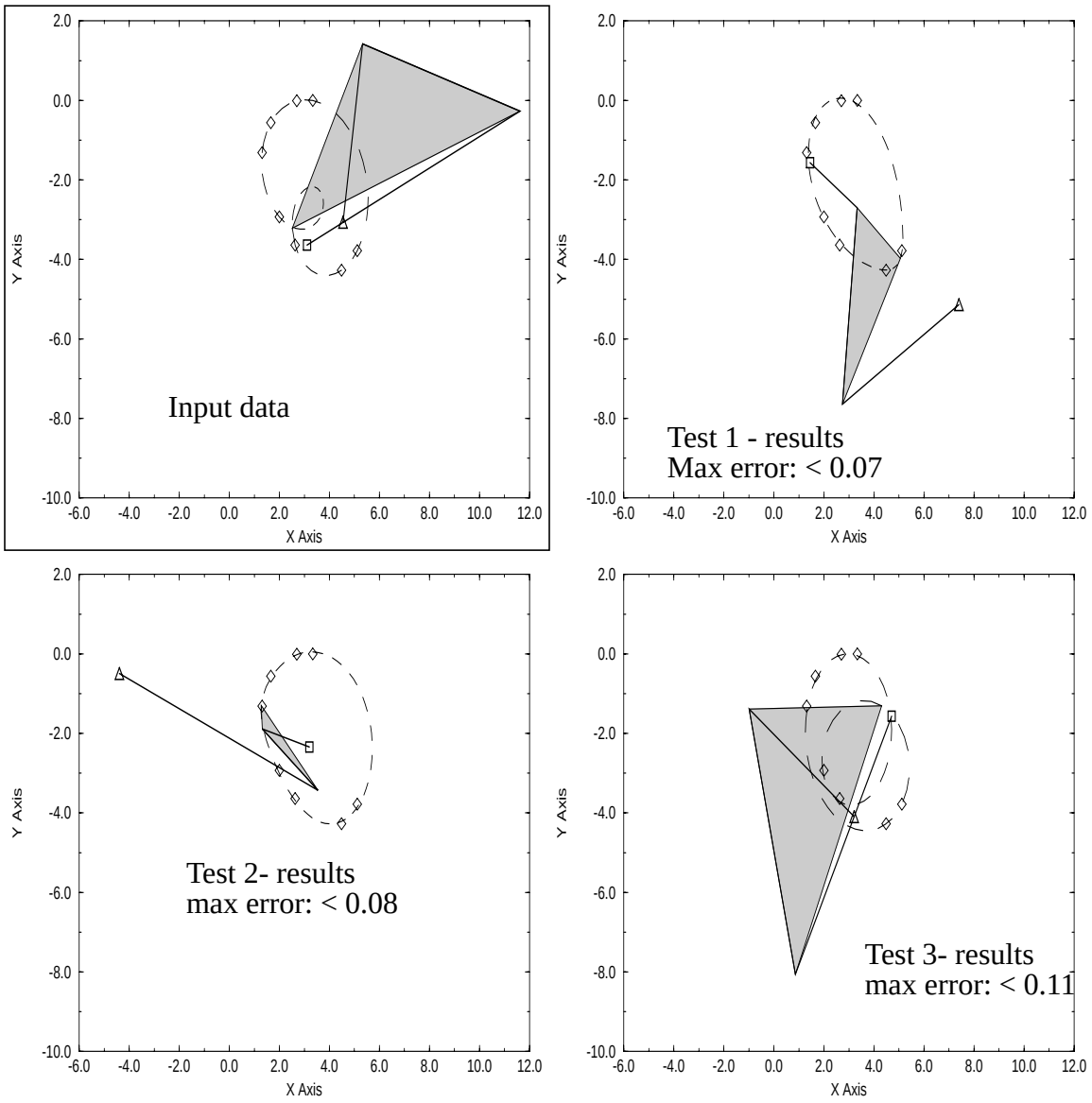


Figure 32. Planar four-bar synthesis, eight specified points

The input data points selected for this problem “disguise” the fact that the coupler curve of the mechanism that generated the specified points has a loop. Thus, two of the three test results shown do not have loops in their coupler curves. Although these results are good, it is expected that there is only limited potential for improvement since the basic shapes of the curves differ. This is not the case for the previous two experiments, because there are an infinite number of solutions for the first experiment and a finite number of solutions to the second experiment, in which cases solutions that appear to be quite different can indeed converge to the same solution. Also note that since the order in which the points are satisfied was not specified, the ordering in the three solutions presented are different. In this experiment, there are two drag-link and two crank-rocker mechanisms.

6.1.4.4 Eight specified points - generated by four bar - limited crank angle

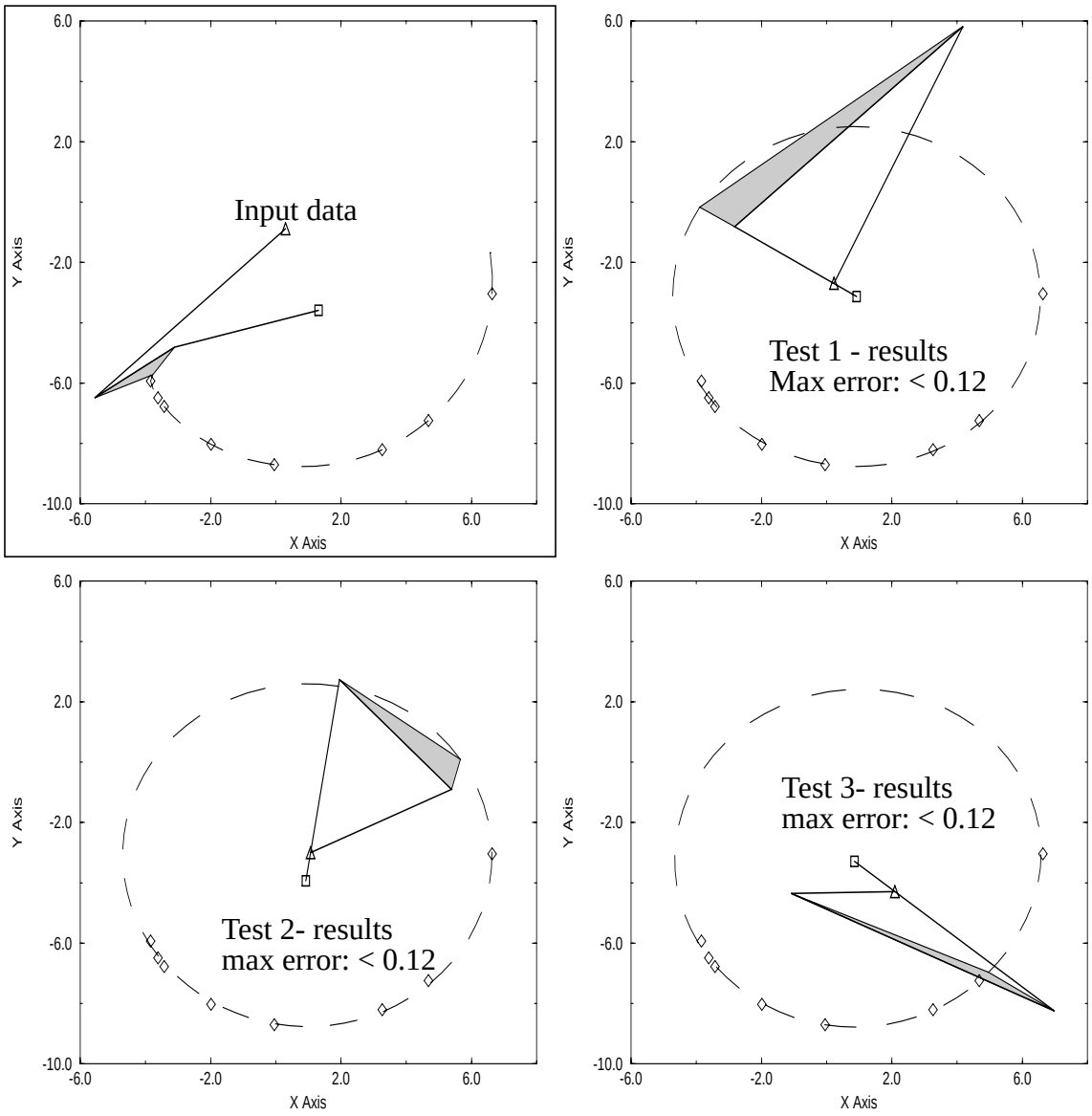


Figure 33. Planar four-bar synthesis, eight specified points, limited crank angle

What distinguishes this experiment from the previous ones is that the specified points were generated by a mechanism that only has 180^o of input crank angle rotation, a double-rocker mechanism, but the generated mechanism is required to have 360^o of crank rotation. The success of the algorithm in solving this particular experiment suggests that this algorithm might be appropriate for synthesizing mechanisms that mimic the motions of other mechanisms. This case might represent the desire to replace a six-bar mechanism (an input four-bar to generate the crank angle and an output four-bar to generate the path shown as the input data), with a single four-bar for purposes of reducing manufacturing costs. Each of the generated mechanisms is a drag-link mechanisms.

6.1.4.5 Three specified points - involute curve

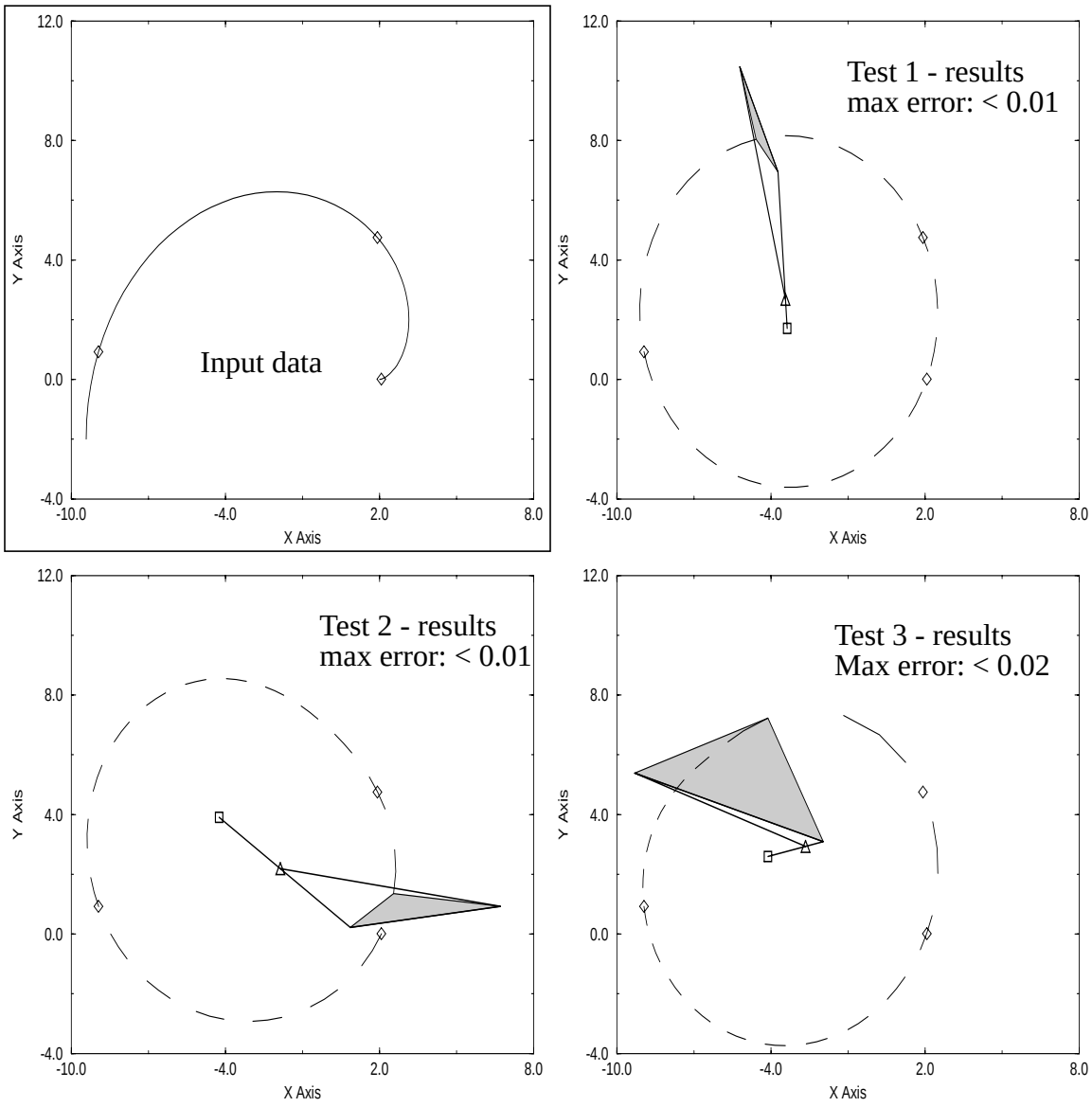


Figure 34. Planar four-bar synthesis, three specified points - involute curve

Since there exists an infinite number of four-bar mechanisms that pass through three specified points, the expected results of this experiment were three good solutions, even though there was no guarantee that any solutions, as constrained by the mechanism representation, actually exist. However, since three points form a circle, and the center of the circle formed by the three points lies well within the represented area, it was expected that a solution would be found. What is interesting is that the magnitude of the errors is similar to the previous experiment with three specified points. This suggests that the error for this methodology might be quantifiable. It is also interesting to note that the coupler curves generated do not resemble the curve of the input function. Each of the solutions shown uses a drag-link mechanism.

6.1.4.6 Five specified points - involute curve

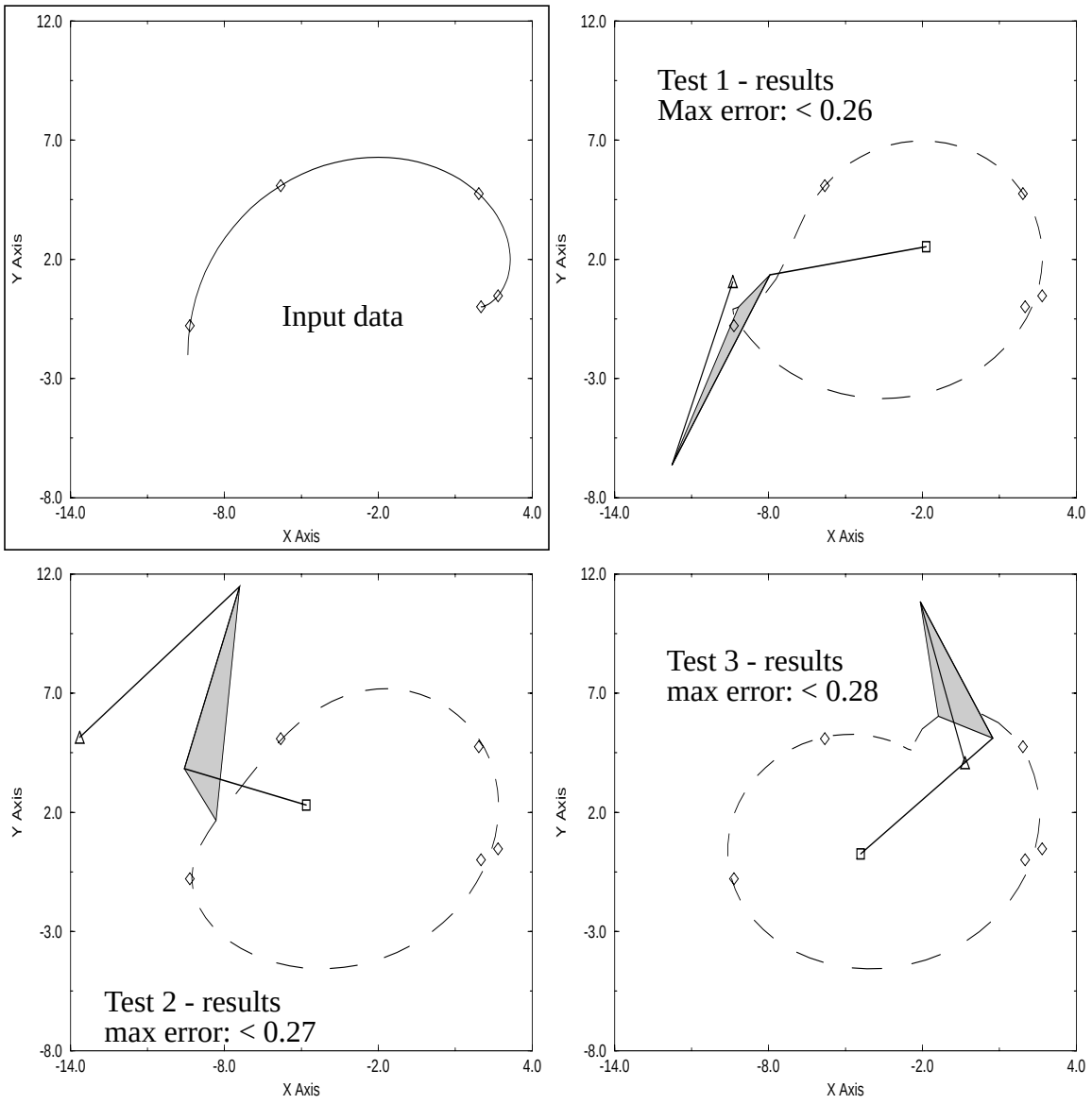


Figure 35. Planar four-bar synthesis, five specified points - involute curve

Since it was not known if this particular set of five points could be passed through by any four-bar mechanism, let alone one with the restrictions imposed, it was encouraging to see that a solution was found. However, the coupler curves from Tests 1 and 3 bear little resemblance to the involute curve. Since classical synthesis theory does not permit the specification of more than five points, the coupler curve generated by mechanisms that result from classical theory application may also bear little resemblance to the involute curve. More points need to be specified to mimic the curve more closely. The solutions shown are both drag-link and crank-rocker mechanisms.

6.1.4.7 Eight specified points - involute curve

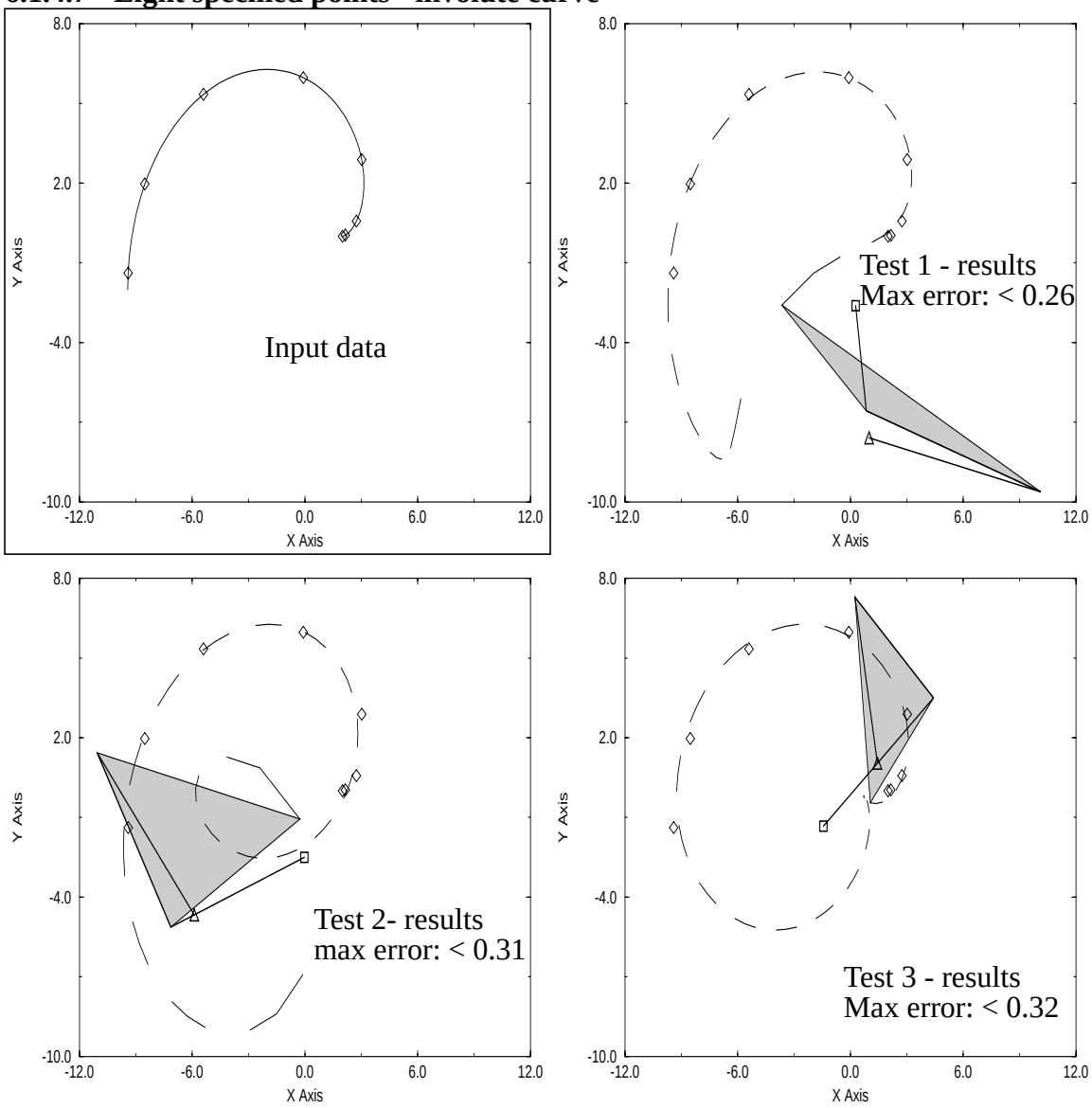


Figure 36. Planar four-bar synthesis, eight specified points - involute curve

It was encouraging to note that the methodology found a solution because classical theory cannot provide a solution to this problem. In addition, it was not known that a solution could be found within the representational space, however, the results of the previous experiments did suggest that solutions would be found. In addition, by specifying eight points, the coupler curves of these three mechanisms (in the region of interest) do more closely approximate an involute curve than the coupler curves from the previous solutions. This suggests that by specifying a fairly large number of points, it may be possible to synthesis a four-bar mechanism that follows a particular path, see Section 6.1.4.10. Both drag-link and crank-rocker mechanisms appear in these solutions.

6.1.4.8 Four specified points - straight line-motion

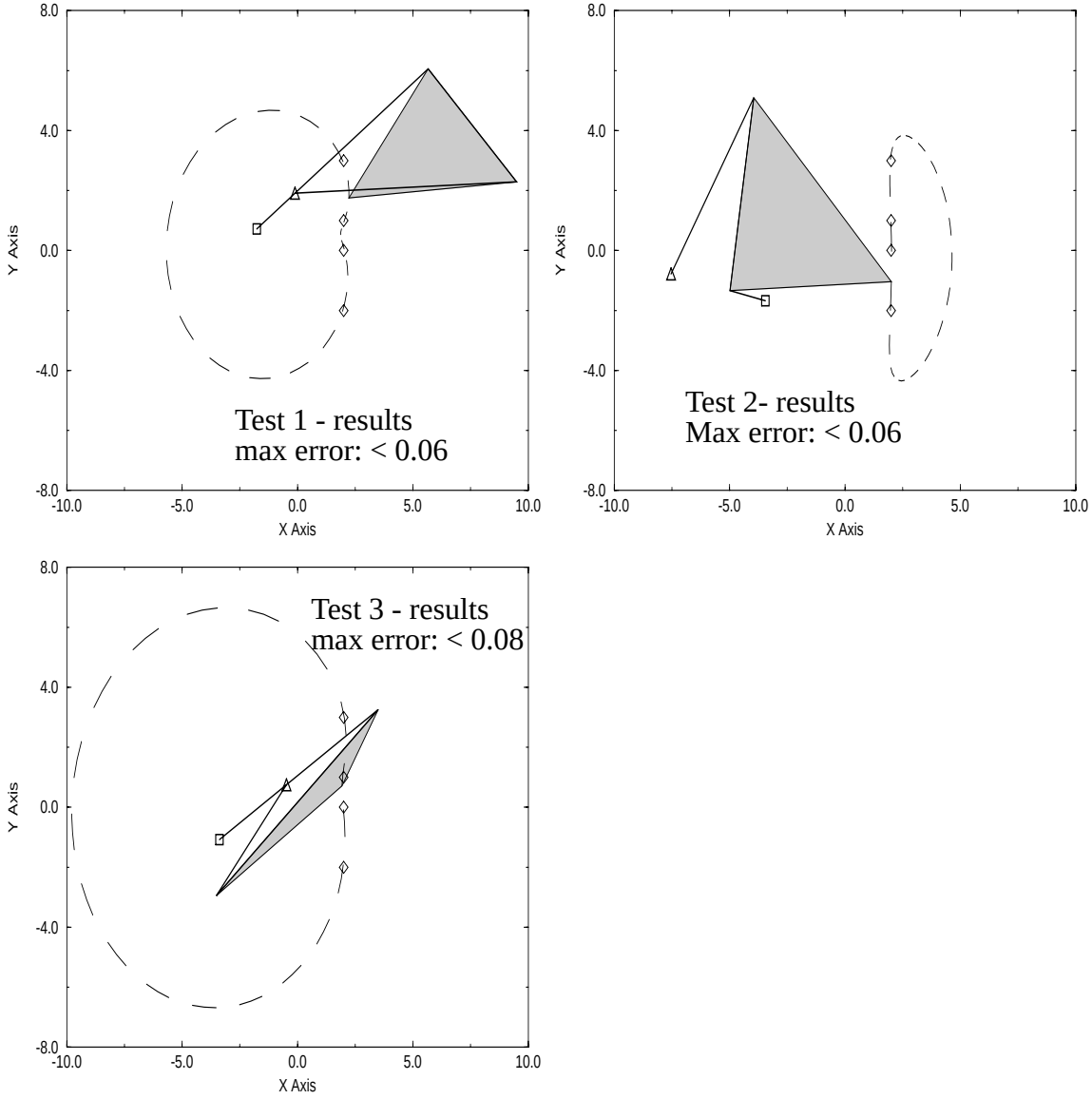


Figure 37. Planar four-bar synthesis, four specified points - straight-line motion

The purpose of this test was to see if the GA method could produce one of the “classical” solutions to straight-line motion generation using a four-bar mechanism. The result of Test 1 is quite disappointing in the sense that it failed to generate straight-line motion. The coupler curves generated for Tests 2 and 3 are similar to the curves generated by the classical Chebyshev straight-line mechanism, although neither of these mechanisms bear any resemblance to the classical mechanisms. These tests indicate the need to use more points to achieve the straight line motion. This is done in the next experiment. This test also raises the as-yet unsolved question of translating functional objectives into the design domain. The actual goal of this experiment is the generation of a mechanism that yields straight-line motion, not the generation of a mechanism that passes through a set of points. Since it is

not known how to specify the former problem, the latter problem is solved instead, which can lead to erroneous results, such as those in Test 1.

6.1.4.9 Eleven specified points - straight-line motion

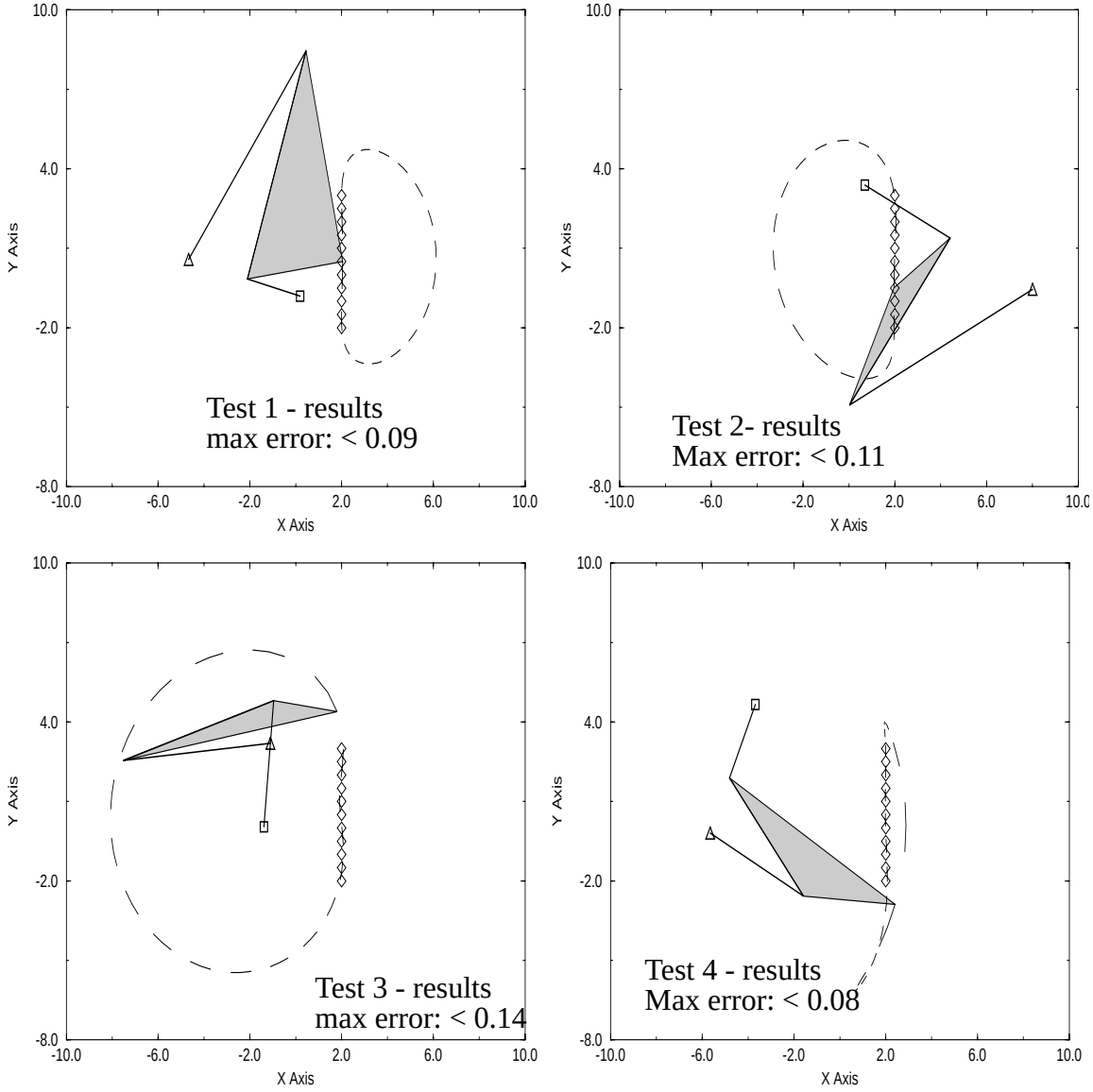


Figure 38. Planar four-bar synthesis, eleven specified points - straight-line motion

Increasing the number of specified points from four to eleven greatly improved the quality of the generated mechanisms. The shape of coupler curve from Tests 1, 2 and 3 are reminiscent of the coupler curve formed by the classical Chebyshev straight-line motion mechanism, but, as in the previous experiment, these mechanisms do not resemble the classical ones. The coupler curve in Test 4 is slightly reminiscent of the coupler curve of a Roberts straight-line motion mechanism. What makes the classical solutions so brilliant is that they were developed without the aid of computers. What makes them special is they typically have some special properties, such as symmetry. The GA method offers the

opportunity for discovering many new classes of straight-line mechanisms, classes that may not share any of these special properties with the classic solutions. Since the length of the straight-line segment is 5, the path deviation percent error is about 2 percent.

6.1.4.10 Twenty-four specified points - generated by four-bar

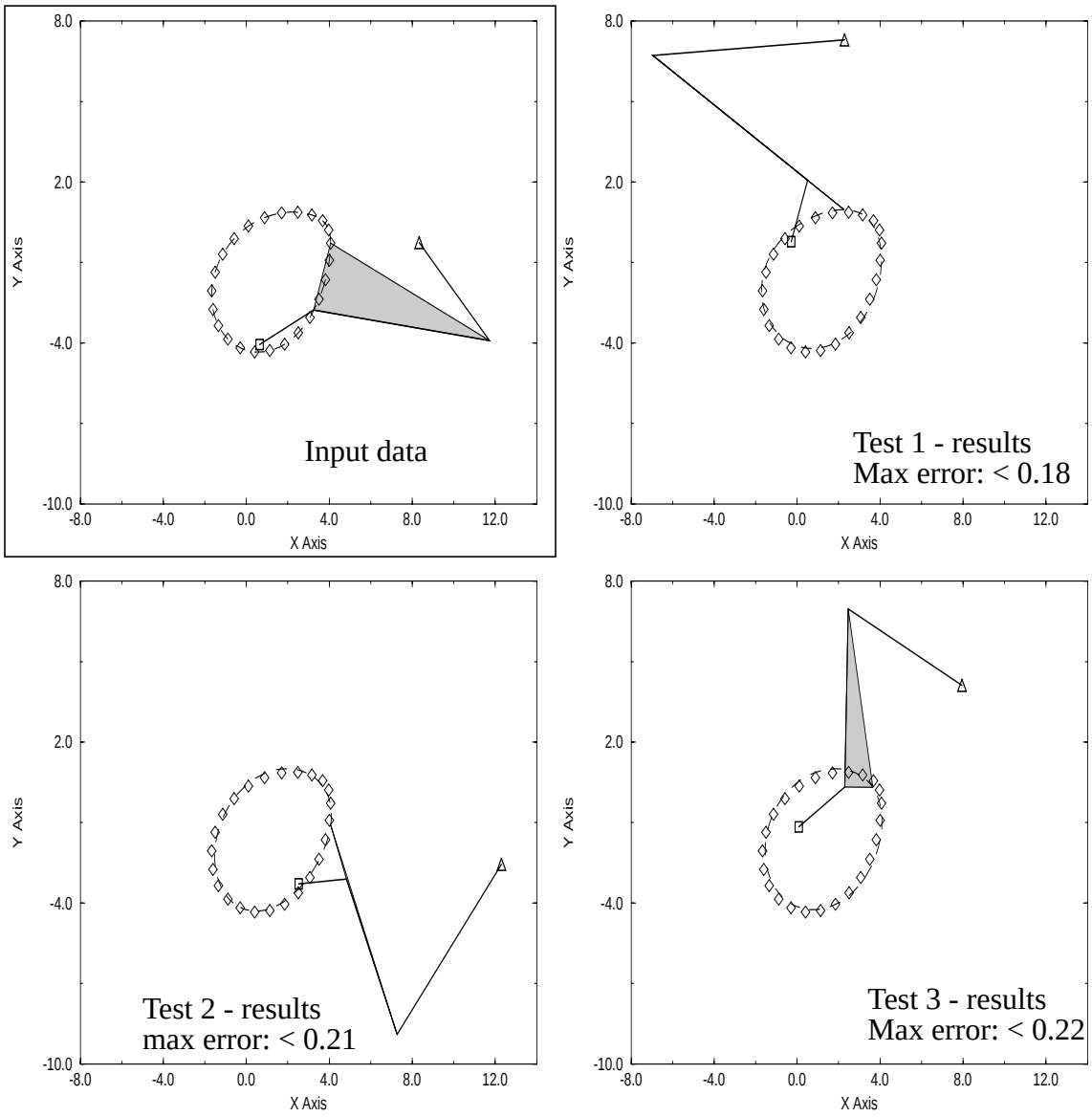


Figure 39. Twenty-four specified points - generated by four-bar

This experiment highlights one of the strengths of the GA synthesis method, the ability to specify the entire coupler path. Classical techniques, or even the GA technique, may not sufficiently constrain the path of the coupler if few points are specified. By specifying a large number of points, the path will necessarily be constrained. The results obtained by using the GA technique may yield “better” results than the classical (graphical) methods because, although the classical methods will guarantee that the precision points are achieved precisely, the coupler point path from such a mechanism may not be viable for the

particular application. With the GA method, although the specified points are not precisely achieved, the overall shape of the coupler point path can be guaranteed. Not only are all of the mechanisms shown crank-rockers, but they are much more similar to each other than the solutions presented in the previous experiments.

6.1.4.11 The need for scaling

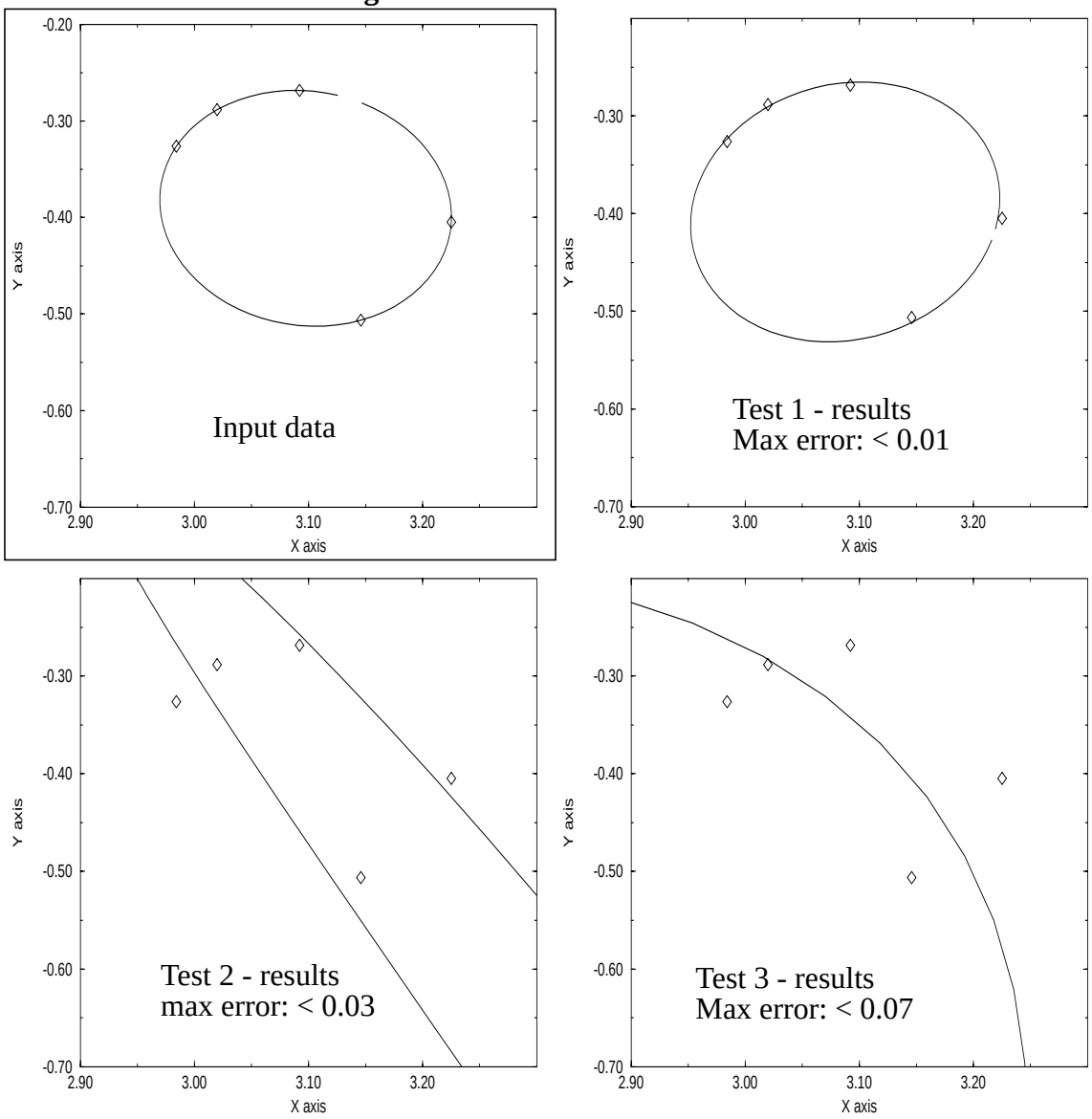


Figure 40. Planar four-bar synthesis, the need for scaling

The important thing to note in this experiment is the small size of the input data coupler curve with respect to the size of the space representable by the GA method. This curve occupies an area of approximately 0.25 by 0.25, or about 0.01 percent of the area covered by the representable space. Although the results from Test 1 are quite good, the results from Tests 2 and 3 show two solutions with high fitness values, but whose solutions are probably not acceptable. This indicates the need to specify more points if a path is to be followed and

also the need to scale the specified points (or the mechanism) so that the points occupy a larger percentage of the representable space. Scaling should be straight-forward if only positional constraints are imposed. If velocity and/or acceleration constraints are imposed, scaling may not be easy to implement. The mechanisms in this experiment are not shown because they do not provide any additional useful information.

6.1.5 Summary

The results of the experiments indicate that the evaluation function proposed, Equation (18), will yield acceptable results. The only concern raised is that of scaling, Section 6.1.4.11, however, this would be a concern for any optimization based synthesis technique. This raises the question of the applicability of other optimization techniques for solving this problem. Since most other optimization techniques require auxiliary information, such as local gradients, they may work well on this problem because certain small parameters changes can dramatically alter the mechanisms performance. However, combining a hill-climbing technique with the GA should provide better results than those shown here. With a technique based on a combination of GA and hill-climbing, the GA would be used to get a “coarse” answer, then the hill-climbing technique would refine that answer.

The only remaining question is whether or not this technique is viable in terms of resources required. The remainder of this summary explores the computing time required to generate some solutions.

There are two control parameters that strongly effect the run-time of this algorithm, and two that weakly affect it. The two that strongly effect the run-time are the population and the number of generations. For each of these parameters, the change in run-time should be a linear function of the change in the parameter. For example, doubling any one of these should result in a doubling of the run-time. The parameters that weakly effect run-time are the required input rotation and the number of specified points. The effect of the required input rotation parameter is unclear. Specifying a larger value means that more points need to be compared, although some mechanisms will not be checked because they fail to meet the requirement. However, a smaller value for this parameter means that more mechanisms will need to checked. Overall, a larger parameter value will probably increase run-time, but the relationship is not clear. It was also assumed that the number of specified points would have a significant impact on the execution time, but since the most expensive part of the experiment is the mechanism synthesis, doubling the number of specified points did not have the expected impact.

The following table presents some run-time results from running this program. These results were obtained on a Sparc 20/61TGX with 32 MB of memory. All parameters are the same as shown in Table 14, except as superceded here. Each entry shows the average time and standard deviation, in seconds, for three runs. These times were obtained by using the

Unix `/usr/bin/time` routine. Each time was obtained by running each experiment three times and averaging the results:

	number of specified points: 4			number of specified points: 8		
	generations			generations		
population	50	100	150	50	100	150
100	81.3	138.0	218.9	77.8	141.3	200.7
	2.0	12.5	28.1	2.6	6.2	17.4
200	155.0	271.1	400.3	150.8	260.2	410.2
	10.1	2.1	15.9	5.3	15.8	48.2
300	223.3	417.1	589.9	219.2	392.7	598.6
	4.4	37.2	29.7	5.3	8.5	50.5

Table 15: Run-times (seconds) for GA four-bar synthesis program

The results shown in Table 15 are as expected. The times do not scale exactly linearly, but this is expected since generating the first generation takes some fixed amount of time. The average time (and standard deviation) required to generate a population of 100, 200 and 300 mechanisms is 16.4 (2.1), 32.8 (1.4) and 50.6 (2.68) seconds respectively. Not only do these times agree with the claim that the size of the population has a linear affect on the execution time, but subtracting these times from the times shown in Table 15, makes those values more closely agree with the claim. For example, with a population of 200 and four specified points, the ratio of the times for 100 and 150 generations to 50 generations is 1.75:1 and 2.58:1. However, subtracting the time to generate the population changes the ratios to 1.95:1 and 3.01:1. Thus, the claim is well supported. This can be used by the designer to estimate the computing resources necessary to solve a specified problem.

The experimental results presented in this section indicate that the evaluation function proposed in Equation (18) can be used to generate four-bar mechanisms. Since the representation of a four-bar mechanism is fundamentally no different from higher order mechanisms, and since the task to be accomplished is the same, it is believed that this same evaluation function can be used for the more general planar synthesis problem.

6.2 GD for general planar mechanism synthesis

➤ This section shows how the GD methodology can be applied to systematically design planar mechanisms. A benefit of using the GD methodology is that it can perform type, number and dimension synthesis simultaneously. The work in this section is limited to revolute jointed mechanisms with one degree of freedom.

The previous section showed how a GA could be applied to the dimension synthesis of a four-bar mechanism. The underlying premise of that section, and indeed, the underlying premise of all four-bar synthesis techniques, is that a four-bar mechanism can provide a solution to the problem. However, Burmester theory shows that if more than five points are specified, a four-bar mechanism cannot, in general, provide a solution. (In the context of Burmester theory, solutions must be exact, so the solutions shown in the previous section for more than five points do not contradict the previous sentence.) To solve the general *n*-point problem, type, number and dimension synthesis must be employed. Type synthesis refers to the selection of the types of motion generating elements, revolute joints, prismatic joints, cams, etc. Number synthesis refers to selection of the appropriate numbers of components to achieve the required motion. Dimension synthesis refers to the determination of the sizes of the components to yield the motion required. The previous section dealt only with dimension synthesis — this section deals with all three.

The work outlined in this section presents a methodology for the simultaneous type, number and dimension synthesis of planar mechanisms of one degree of freedom. The problem solved is the same as the previous section: to design a mechanism to pass through a set of user specified points. In this work, the mechanisms are restricted to being four-, six-, or eight-bar mechanisms with one degree of freedom with each of the joints a revolute joint. There is nothing inherent in the methodology that requires this restriction — these restrictions are for pragmatic reasons. This work can be extended to the general synthesis of planar mechanisms of any size and with any type of joint. The extension into three dimensions should be possible, however, certain formulas (such as Greubler's criterion for the determination of degrees of freedom of a mechanism) do not properly translate to three dimensions in all cases.

This section is divided into three parts. The first part, Section 6.2.1, describes how the mechanism is described and how the representations are genetically manipulated. The second part, Section 6.2.2 describes some of the implementation issues and the third part, Section 6.2.3 describes the results from some experiments using this model.

6.2.1 Planar mechanism model representation

➤ The key difficulty encountered in solving this problem was devising a method to describe mechanisms such that they could be manipulated genetically using existing genetic representations and manipulations. The reason for this difficulty is that planar mechanisms are connected graphs, and connected graphs are very difficult to represent with context-free grammars. Consider the example of a Stephenson III six-bar mechanism as shown in Figure 41. (Note, the missing sixth link is the ground link - and is notated as link 1.)

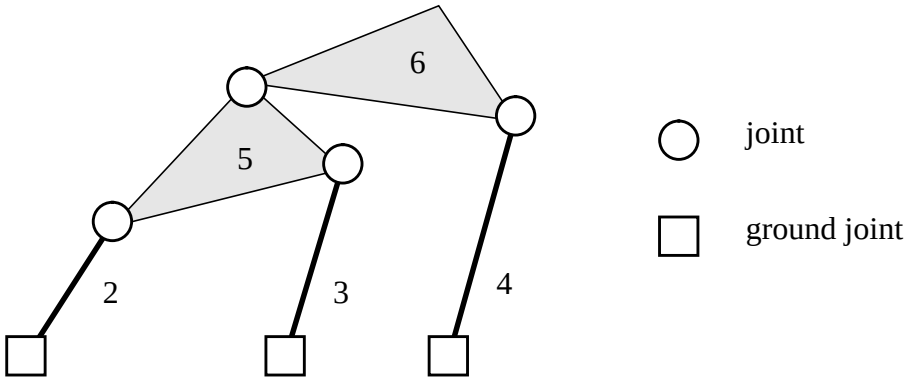


Figure 41. Stephenson III six-bar mechanism

To generate a tree (a graphical representation of a context-free grammar) that represents this structure directly is not feasible because at some point in the tree, a link is required to a previously defined node. Because of this link, traditional GP techniques cannot be applied to this type of a representation.

➤ To allow representing mechanisms as tree structures (that is, representable by a context-free grammar), an entirely different means of representation is required. This new representation can be envisioned not as the mechanism itself, but rather as a program that builds the linkage. By using this scheme, the mechanism is representable with a context-free grammar and can be manipulated using GD. The approach used will define commands, which require parameters, for building a linkage. The definition of a mechanism then becomes an instantiation of a program that calls these commands with certain supplied parameters.

To implement this scheme, the linkage is represented by its interchange representation instead of its direct representation [21] [30]. In the interchange representation, the nodes are links and the arcs are joints. Figure 42 shows the interchange representation of the mechanism shown in Figure 41, with the nodes labeled with their link numbers:

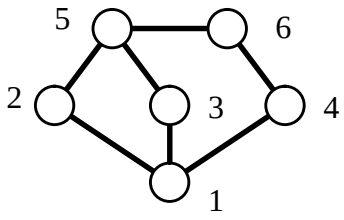


Figure 42. Interchange representation of a Stephenson III six-bar mechanism

The following two procedures are sufficient for designing any planar mechanism that is comprised of binary revolute joints and links. These procedures can be extended to other types of binary joints without changing the underlying theory. Non-binary joints can be modeled as a set of binary joints joined by a zero-length link:

```
link (n) [n = 2, 3, ..., N]
joint (Li, Lij, Lk, Lkl)
```

where

- `n` is the number of incident joints at the newly created link,
- `N` is the maximum number of incident joints,
- `Li` is the first link to be joined, where i ranges from 1 to the number of links in the mechanism,
- `Lij` is the jth joint on link i,
- `Lk` is the second link to be joined, where j ranges from 1 to the number of links in the mechanism and $j \neq i$,
- `Lkl` is the lth joint on link k.

To show that GD can be used to design planar mechanisms using this means of representation, the following four topics need to be addressed: how to create an initial population, how to represent the program as a tree (a grammar), how to perform cross-over and how to perform a mutation. Each of these is now considered.

The first step in creating an initial population is to determine the number of links in the mechanism. The number of links, however, cannot be chosen completely at random since each linkage in the population should have exactly one degree of freedom. Rearranging the Greubler formula: $F = 3(n-1) - 2f$, and substituting $F = 1$ yields $3n - 2f = 4$. This equation is then solved for all solutions that yield integral values of f with $n \leq 10$. (The solution $n = 2,\ f = 1$ is not considered as it represents a trivial solution. The upper bound is chosen for pragmatic reasons.) This yields the following combinations of links and joints:

link/joint combinations yielding 1 DOF			
n	j	n	j
4	4	8	10
6	7	10	13

Table 16: Valid link/joint combinations for 1 DOF mechanisms

After selecting the number of links, the link type for each link is determined. For this work, links will be one of three types: binary, ternary and quaternary. The definitions for each of these types of links is shown below:

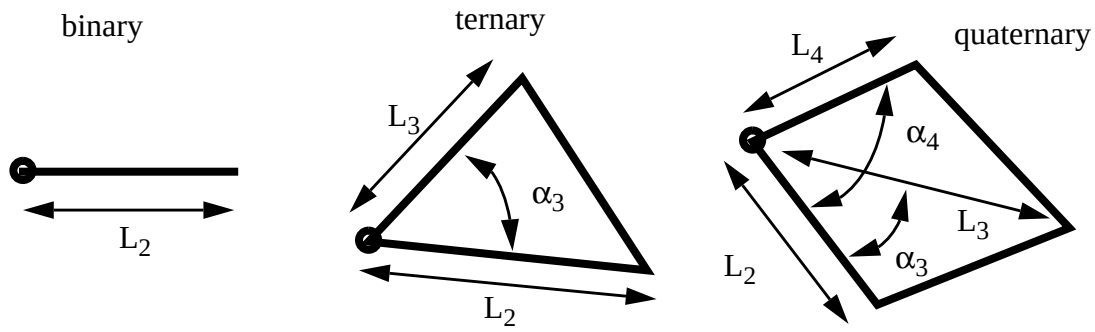


Figure 43. Link definitions

Note that this scheme is perfectly general - it does not enforce any restrictions on the shapes of these links. The small circle denotes the first joint-locus on the link. The distance to the ith joint-locus is given by L_i. Any necessary angular offsets for joint i are given by α_i, where the angle is measured counter-clockwise positive from the line segment defined by joint-locus 1 and joint-locus 2 to the line segment formed by joint-locus 1 and joint-locus i.

The choice of link type is not completely random. For a mechanism to be assembled, the sum total of joint-loci for all links must satisfy the equation $\sum L_i \geq j$, where L_i is the number of joint loci on link i, where L=2 for a binary, L=3 for a ternary and L=4 for a quaternary. In theory, satisfying this equation is sufficient to guarantee viable mechanisms, however, in practice, a fair amount of overhead was required to ensure that this equation was satisfied. To simplify the implementation, binary links are not used. This does not impose any restrictions on the methodology as a ternary with only two utilized joint-loci is in fact a binary link.

To ground the mechanism, the first link is defined to be the ground link. In addition to any link parameters, three additional parameters are needed for the ground link: the x and y position of the first joint and the orientation of the line segment formed by joint-locus 1 and joint-locus 2 from the world x axis.

Once the number and type of the links are determined, the joints are identified. The number of joints to be created is given by Greubler's formula, see page 124. The joints are created with a two step procedure. First, each joint is assigned a random pair of links to connect. Second, each link is checked to ensure that it is connected to two other links. If it is not, a joint is removed from another link (the one that has the greatest number of incident links), and is assigned to the link. Although this process uses random numbers, in practice it converges very quickly.

The creation of six-bar mechanisms was studied in detail to ensure that the mechanism creation process functions properly. Figure 44 shows the four topologies of six-bar mecha-

nisms that can be generated with ternary links. The type labeled D2 is a six-bar that has "degenerated" into a two-bar and the type labeled D4 is a six-bar that has degenerated into a four-bar. These degenerate mechanism topologies (and other like them in higher-order mechanisms) are identified and eliminated from consideration in the planar synthesis program.

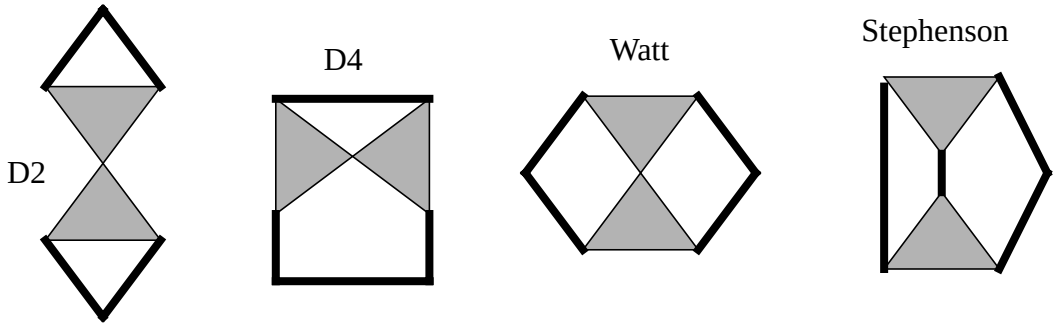


Figure 44. Six-bar mechanism topologies

A program was written that uses the mechanism generation routines from the planar-synthesis program. This program generated 1,000,000 mechanisms and sorted them by topology and type. Type is determined by the size (binary or ternary) of link 0, the ground link. For the two degenerate mechanisms, type 1 means that the ground link is a ternary link and a type 2 means that the ground link is a binary link. The traditional nomenclature is used for the Watt and Stephenson mechanisms. Since the differentiation among the types is the size of link 0, the ratio of types can be predicted. However, the ratio of the different topologies is very difficult to predict because of the "random" means of generation of the mechanisms.

D2	D4	Watt	Stephenson
T1:T2 = 1:2	T1:T2 = 1:2	W1:W2 = 2:1	S1:S2 = 1:1
			S2:S3 = 1:1

The results of running the mechanism generation program are presented below. The program was run three times, each time generating 1,000,000 mechanisms. The predictions listed above are met, thus there appears to be no programmatic bias. However, there does appear to be a small bias, about three percent, in generating Watt mechanisms instead of Stephenson mechanisms. Whether this bias is programmatic or expected, due to the method of mechanism generation, is unknown. However, the bias is small, so overall, the planar synthesis program probably generates a fair cross-sampling of mechanism types.

	D2	D4	Watt	Stephenson
T1:T2	1:1.996	1:.998	2.002:1	0.998:1
T2:T3				1.000:1

Table 17: Ratios of types of six-bar mechanisms generated

Once the mechanism is completely assembled, the joint for the input angle is specified. This is done by randomly selecting one of the joints incident to the ground link.

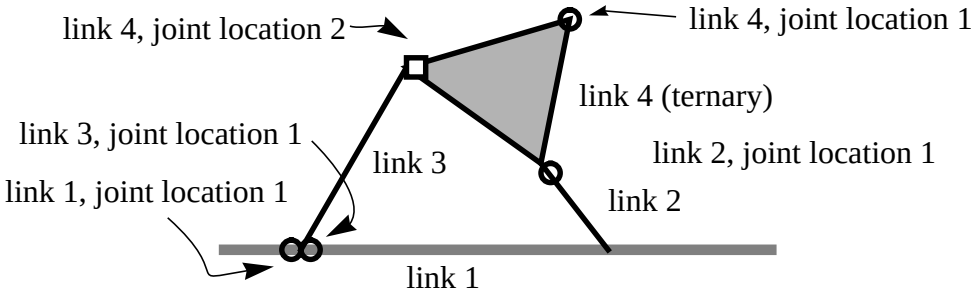
The next step is to create a grammar that embodies these ideas. This grammar should be context free since CFGs are typically unambiguous and are easily translated into rooted tree representations. It is important to note that the grammar need not embody the ability to create the mechanism nor must it only be to represent legal mechanisms. Rather it must embody the ***program*** that was developed to describe the mechanism. The following grammar is proposed:

mechanism →	**L** *links* **J** *joints* **G** *ground*
links →	*link_type links* | *link_type*
link_type →	**t** *ternary* | **q** *quaternary*
ternary →	*real_1 real_2*
quaternary →	*real_1 real_2 real_2*
joints →	*joint_description joints* | *joint_description*
joint_description →	**(** *integer_4* **)**
ground →	*real_3*
real_1 →	`real_number`
real_2 →	`real_number`**,** `real_number`
real_3 →	`real_number`**,** `real_number`**,** `real_number`
integer_4 →	`int_number`**,** `int_number`**,** `int_number`**,** `int_number`

Table 18: Planar mechanism context-free grammar

The grouping of `real_number` is used to allow a wider variety of cross-overs. For all links, the *real_1* represents the distance L_2. The *real_2* for the ternary and quaternary links represent the distance and angular offset, respectively, to the other joint locations. For *joint_description*, the *integer_4* represent the first link, the joint-locus on the first link, the second link and the joint-locus on the second link, respectively. For *ground*, the *real_3* represents the x and y coordinates of the first joint location on the ground link and the angle, measured counter-clockwise positive between the world x axis and the line segment formed between joint location one and joint location two on the ground link. The tokens, **L**, **J**, **G**, **b**, **t**, **q**, **(**, **)** and **,** are used for clarity.

To show how this grammar works, consider the following example, Figure 45. (Although binary links are not included in the grammar, this example uses three binary links for succinctness.):



L b L1,1 **b** L2,1 **b** L3,1 **t** L4,1 L4,2 a4,2 **J (**1,1,3,1**) (**1,2,2,2**) (**4,2,3,2**) (**2,1,4,3**) G** px, py, a1,2

Figure 45. Grammatical representation of a four-bar mechanism

To specify the lengths, the notation Lu,w will be used, where u is the link number and w is the length. To specify the angle, the notation αu,w will be used, where u is the link number and w is the length (see link definitions, Figure 43). Also, as in Figure 43, joint location 1 is identified with a small circle and joint location 2 (on the ternary only) is represented with a small square.

This grammar can be mapped into a rooted tree. Continuing with the previous example:

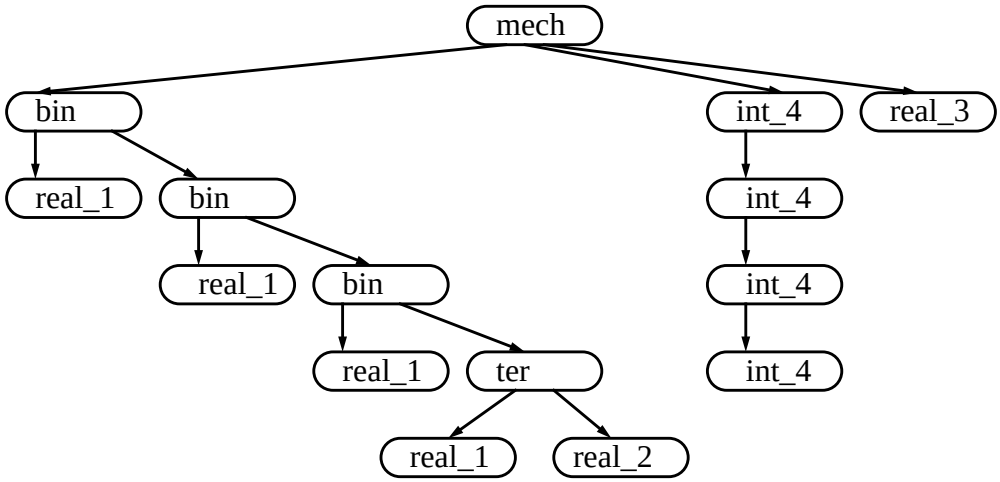


Figure 46. Tree representation of a four-bar mechanism

The next paragraphs discuss manipulating the mechanism descriptions genetically. One of the difficulties with a direct representation of the mechanism is that the concept of cross-over, although seemingly intuitive, does not make sense. Consider the Stephenson III mechanism shown in Figure 41. If links 5 and 6 are removed, that which is left is an open

chain, and it is unclear how to attach links from the other mechanism to this mechanism fragment. Instead, the programs for generating these mechanisms are manipulated.

To accommodate the strongly-typed genetic programming (STGP) concept, the following types are defined: link, joint, real_2, real_3, integer_4, real_number and int_number. Cross-over between two trees is allowed when the nodes types in both trees are of the same type. Cross-over between the numeric types is straight-forward. However, when the cross-over changes the description of the mechanism (the cross-over node is of type links or joint), the resulting program might be an illegal representation. The problem that can arise is that the number of links/joints is no longer valid (see Table 16). Cross-overs that create illegal mechanisms are disallowed — the mechanisms are left unchanged.

Another potential problem is that the joint relationships may no longer be valid. Since this problem will frequently occur, counting as illegal the mechanisms that no longer have proper joint descriptions will reduce the efficacy of cross-over to the point where generating new mechanism configurations will be almost impossible. Instead, after any genetic operation that can effect the joint relationships (including mutation, see below), the joint relationships will be updated. The procedure for updating joint relationships is:

1. Remove any relationships that reference links or joint locations on links that are not part of the mechanism.
2. Remove any relationships that reference joint locations that have been previously referenced.
3. Add joint relationships (using the same procedure as for generating a new mechanism) until the required number of relationships exist.

By using this procedure, the greatest number of relationships are preserved. After updating the joint relationships, the input joint will need to be identified. If the old input joint is still a legal alternative it is used, otherwise, a new input joint is selected (as described previously).

For computing efficiency, GP style cross-over is performed in the following manner. A node is selected at random from one parent and its type is identified. All nodes with the same type in the second parent are identified. If no nodes of that type exist, the cross-over does not occur. If it does, the node in the second parent is selected from the group of nodes of the same type. This method is more efficient than randomly selecting nodes in the second parent until one of the same type as the cross-over node in the first parent is selected.

In addition to the GP type cross-over just discussed, GA type cross-over can also occur. This type of cross-over occurs only between two real_numbers, one in each tree. (Cross-over between integer types is not considered because simply changing one of the four joint description parameters in each mechanism does not seem to make sense. In addition, such a cross-over could create an illegal number - such as a link 5 for a four-bar.) This GA-type cross-over is carried out in the manner discussed in Section 3.3.

There are several types of possible mutations. These mutations fall into one of two broad categories: numerical and structural. Numerical mutations are the typical GA-type, bit-wise mutation. Each bit representing a real number is subject to mutation. There are three types of structural mutations: the addition/removal of a dyad, changing a link type and changing joint relationships. The type of structural mutation that occurs depends on the type of node at which the mutation is called for: a link node generates a link-type mutation or a dyad mutation; and a joint relationship node generates a joint relationship mutation. For any mechanism, only one structural type mutation of any type is permitted per generation.

To add a dyad to an existing mechanism, two links are added to the mechanism, and the joint relationships are updated as outlined above. To remove a dyad, two links are chosen at random to be removed. If the removal of these links would result in an illegal mechanism the mutation does not occur. Otherwise, the two links are removed and the joint relationships are updated as outlined above.

A link type can be mutated into one of the two other types. To change to a "larger" link (binary→ternary or ternary→quaternary), the link type is changed and new parameters randomly generated. To reduce a link, first, it has to be determined if the mechanism will still be legal. If not, the mutation does not occur. If it is legal, the link is reduced and the joint relationships are updated as outlined above.

To mutate a joint relationship, the selected joint relationship is deleted and the joint relationships are updated as outlined above.

As mentioned previously, page 126, there is a large probability that a randomly generated mechanism will be either a degenerate mechanism or one that fails to assemble. Mechanisms with more links are more prone to these problems than those with fewer links. What was observed during the early stages of development of this program is that there was a very strong bias towards creating four-bar mechanisms. To overcome this bias, a genetic operation is accepted with a probability equal to the fitness of the newly generated mechanism (or sum of the mechanism fitnesses in the case of cross-over) divided by the fitness of its parent (or sum of the fitnesses of their parents in the case of cross-over). Should a genetic operation not succeed, the genetic operation is repeated until a successful genetic operation occurs or until ten attempts are made. Using this method, it is important to set the parameter of non-terminal cross-over to terminal cross-overs to be large, otherwise few structural cross-overs will occur. Setting the value of this parameter to four yields a numeric cross-over to structural cross-over ratio of about 10:1 and a numeric mutation to structural mutation ratio of about 3:1.

This method of representation is sufficient for representing and genetically manipulating all planar mechanisms that satisfy the following constraints: all joints are binary joints; all joints are revolute; and all mechanisms have exactly one degree of freedom. Future work can include other types of joints. The extension to 3D may not be possible

since some of the basic tools, such as Greubler's number, do not always yield predictable results for 3D.

6.2.2 Implementation

The previous section presented the means of representation for planar mechanisms. This section describes how the tree representation is converted into a form that can be evaluated. The implementation scheme will be explored through the use of an example.

The evaluation scheme, Section 6.1.3, requires a kinematic analysis of the mechanism, Section 6.1.2. This kinematic analysis is based on the idea of reducing the error in the vector loops of the mechanism to zero. A vector loop is a closed loop formed by traversing each of the links in the mechanism. Figure 47 shows the vector loops in a Stephenson III mechanism.

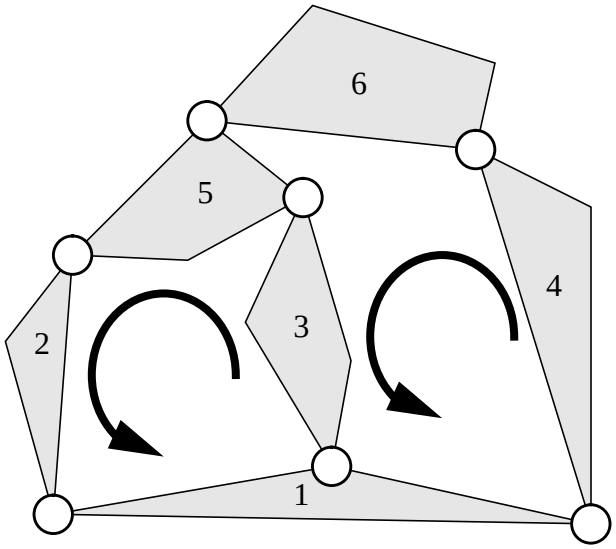


Figure 47. Vector loops in six-bar mechanism

These loops are constructed of the following vectors:

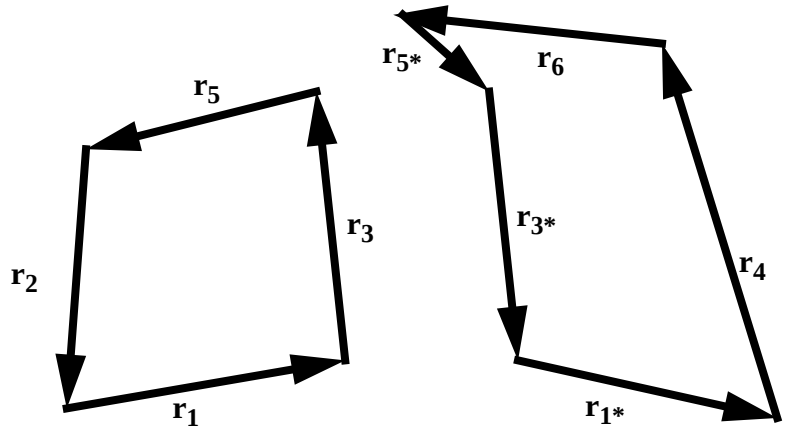


Figure 48. Vector loops definitions

The two vector loop equations are written as:

$$\mathbf{r}_1 + \mathbf{r}_3 + \mathbf{r}_5 + \mathbf{r}_2 = 0 \tag{20}$$

$$\mathbf{r}_4 + \mathbf{r}_6 + \mathbf{r}_{5*} + \mathbf{r}_{3*} + \mathbf{r}_{1*} = 0 \tag{21}$$

where the bold-faced type represents vector quantities, the subscripts represent the link number associated with the particular vector and the * indicates a second vector associated with the particular link. These two vector equations, which are four scalar equations, have four unknowns, the angles of four of the links. By convention, the angle of link 0 is fixed and given and the angle of one of the links attached to link 0 is designated as the input link, and its angle is iterated through some specified range.

For an engineer to write Equations (20) and (21) is a relatively straight forward task. To write these equations automatically, given the description of the mechanism, see Figure 45, requires a two step procedure. First, the vector loops must be identified. Second, the lengths and angles associated with each of the vector must be identified. In addition, vector paths to each of the joint-loci on each of the links must also be identified. Each of these steps is explained below.

Since a mechanism, such as the one depicted in Figure 42, is a network, graph theory can be applied to the problem of finding the vector loops. A graph (what has been referred to as a network) is defined as a set of points (nodes - links) that are connected by a set of lines (edges - joints). A path is an alternating sequence of vertices and edges. A circuit (vector loop) is a path in which the first and last nodes are the same. A tree is a path with no circuits and a spanning tree is a tree that is incident with all nodes in the graph. [31] The number of circuits in a graph is given by

$$C = 1 + v - e \tag{22}$$

where v and e are the number of nodes (links) and edges (joints) respectively in the graph.

To find these circuits, the following procedure is used. First, a spanning tree of the graph is found. This procedure executes in O(e) time. Since there are at least as many nodes (links) as edges (joints), there will be at least one edge (joint) that is not in the spanning tree. Second, the circuits are found by finding the unique path in the spanning tree between the nodes that are incident to the edges that are not in the spanning tree. Figure 49 shows both the interchange representation of the mechanism shown in Figure 41 and the spanning tree that is used to generate the vector loop equations, Equations (20) and Equation (21).

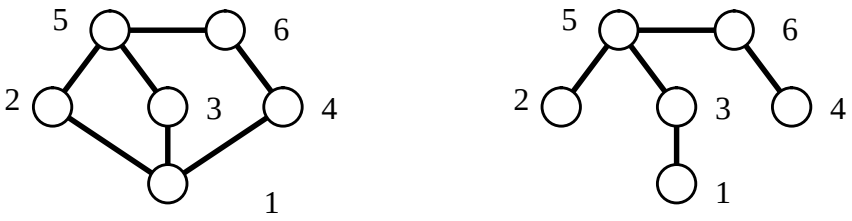


Figure 49. Spanning tree for Stephenson III six-bar mechanism

Identifying the links that the loops traverse is insufficient for specifying the vector loops unambiguously because certain links are traversed by more than one loop. In addition, the vectors that traverse a particular loop may have an angular offset with respect to each other. This problem is overcome in the following manner. (See Figure 50 which shows the mechanism of Figure 41, however, the joints have been exploded so the joint-loci number could be displayed unambiguously.)

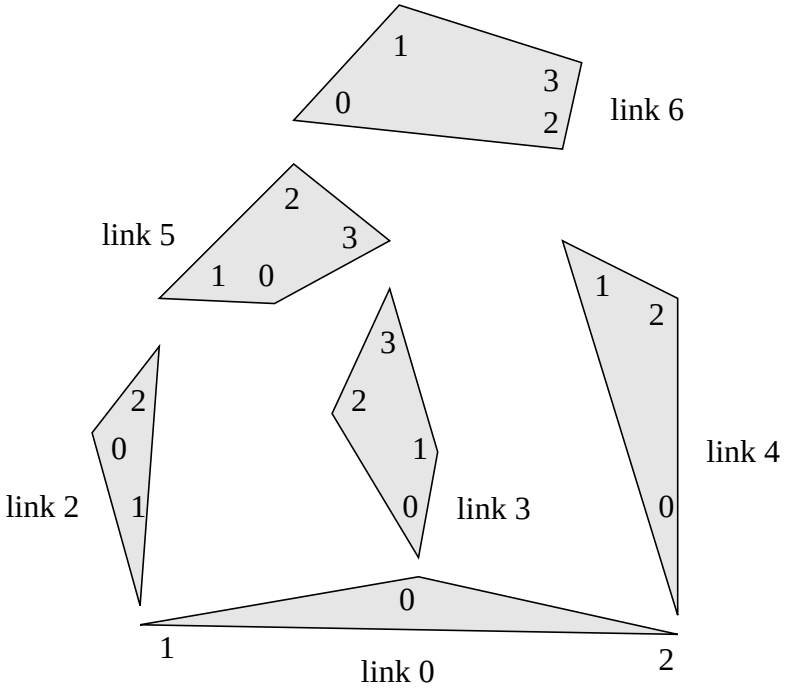


Figure 50. Joint-loci for a six-bar mechanism

Disambiguating the link vectors is a two step procedure. First, the GD representation of the link parameters, Figure 43, is converted into a local Cartesian representation, with the positive x- axis defined as the unit vector pointing from joint-locus 0 to joint-locus 1, and with the origin located at joint-locus 0. Then, when the vector loops are defined, the joint-loci for the link are used to find the actual distance between the loci (Cartesian distance) and the offset angle between the vector formed by the two loci and the local x-axis. During the analysis of the mechanism, the offset angle is added to the link angle to yield

the proper orientation of the particular vector. This scheme yields a consistent representation for all mechanisms under consideration.

For example, refer to Figure 48 and Figure 50: the vector $\mathbf{r_1}$ is defined as traversing link 0 from joint-locus 1 to joint-locus 0. Thus, the offset angle is 180^o.

To locate the coupler point position, vector equations are constructed from link 0, joint-locus 0, the origin of the world coordinate system, to the unused joint loci on each of the links. The construction of these paths is very similar to the construction of the vector loops.

The model description and genetic manipulations described in Section 6.2.1 and the further details provided in this section are sufficient for evaluating planar mechanisms and genetically operating on their descriptions.

The planar mechanism synthesis experiments use the same set of control parameters as the stepping stone walker model, Section 5.1.4, with the addition of two model specific control parameters from Section 6.1.4. The values for these parameters were not obtained by using a meta-GA method because each synthesis experiment takes more than five hours on a Sparc 20 and a meta-GA requires on the order of 1000's of evaluations. The values used were estimated based on the results of the stepping stone walker and the four-bar GA model.

The control parameters used are:

genetic algorithm cross-over probability	0.20	genetic programming cross-over probability	0.75
genetic algorithm mutation probability	0.02	genetic programming mutation probability	0.01
fitness multiplier	2.0	number of elitists	8
population	400	ratio of non-terminal xovers to terminal xovers	4
required input rotation	90 degrees	number of close points before tightening, *C*	4

Table 19: GD parameters for planar mechanism synthesis

6.2.3 Test results

The amount of planar mechanism synthesis testing done is quite limited compared to the other experiments performed because of the computing resources required. Thus, only three experiments have been run. In the first experiment, there are eleven specified points equally spaced on a circular path. In these experiments, the results should be very good because only a few of the mechanism's parameters are needed to satisfy the requirements. In the second experiment, there are five points on the path of the coupler curve of an eight-bar mechanism. The results from this set of experiments should be good because there

exists a large number of mechanisms that can pass through five points. There are two parts to the third experiment In the first, there are thirteen points equally spaced on a straight line. In the second part, there are 25 points equally spaced on a longer straight line. The results of this experiment might not be good because of the large number of parameters that need to be satisfied in order to get a mechanism to pass through these points.

Some interesting observations were made in reviewing the results of these experiments.

- Each run of each experiment “settled in” on mechanisms with the same number of links. That is, after some number of generations a large fraction of the population, typically greater than 90%, had the same number of links. This settling in can be controlled somewhat by reducing the fitness multiplier.
- The average number of links determined the run time. For those cases in which the average number of links approached four, the program ran to completion in less than six hours. In those cases where the average number of links approached ten, the run time was almost 30 hours. (All times reported for a Sparc 20/61.) This suggests that there are some practical limitations to this methodology with respect to this problem.

For each of the experiments, two of the best alternatives generated are displayed. For those cases in which the mechanism is a six-bar, it will be identified as a Watt or Stephenson mechanism. In each of the pictures, the ground link is the link with the heavy black lines. The other links are drawn as colored outlines. In Sections 6.2.3.2 and 6.2.3.3, the coupler curve of the mechanism that generated the coupler path is shown as a solid, dark green line. Other than these differences, these pictures should be interpreted in the same manner as the results from Section 6.1.4, see Figure 29.

6.2.3.1 Planar mechanism synthesis, eleven points on a circle

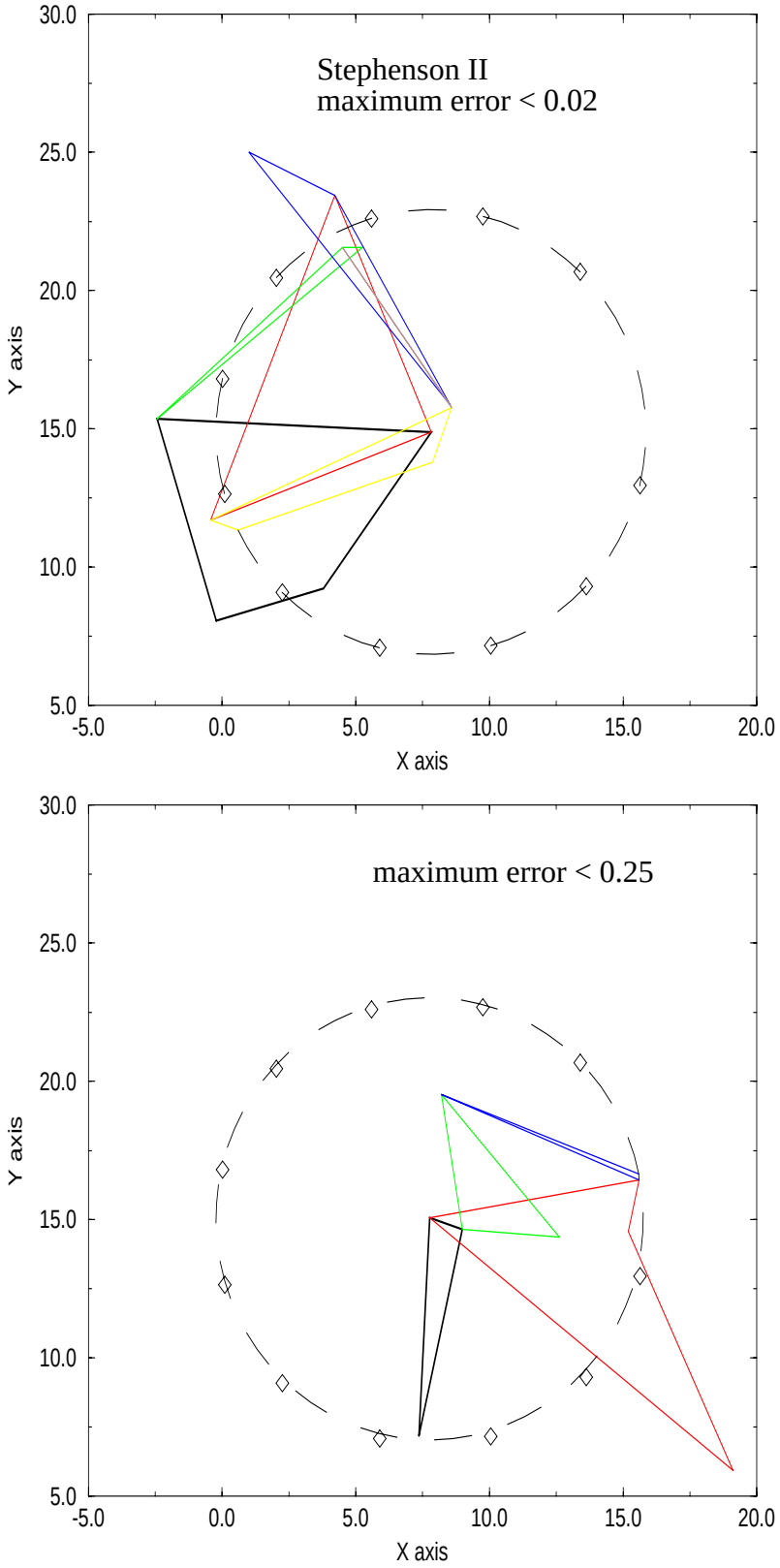


Figure 51. Planar mechanism synthesis, eleven specified points on a circle

6.2.3.2 Planar mechanism synthesis, five points on an eight-bar coupler curve

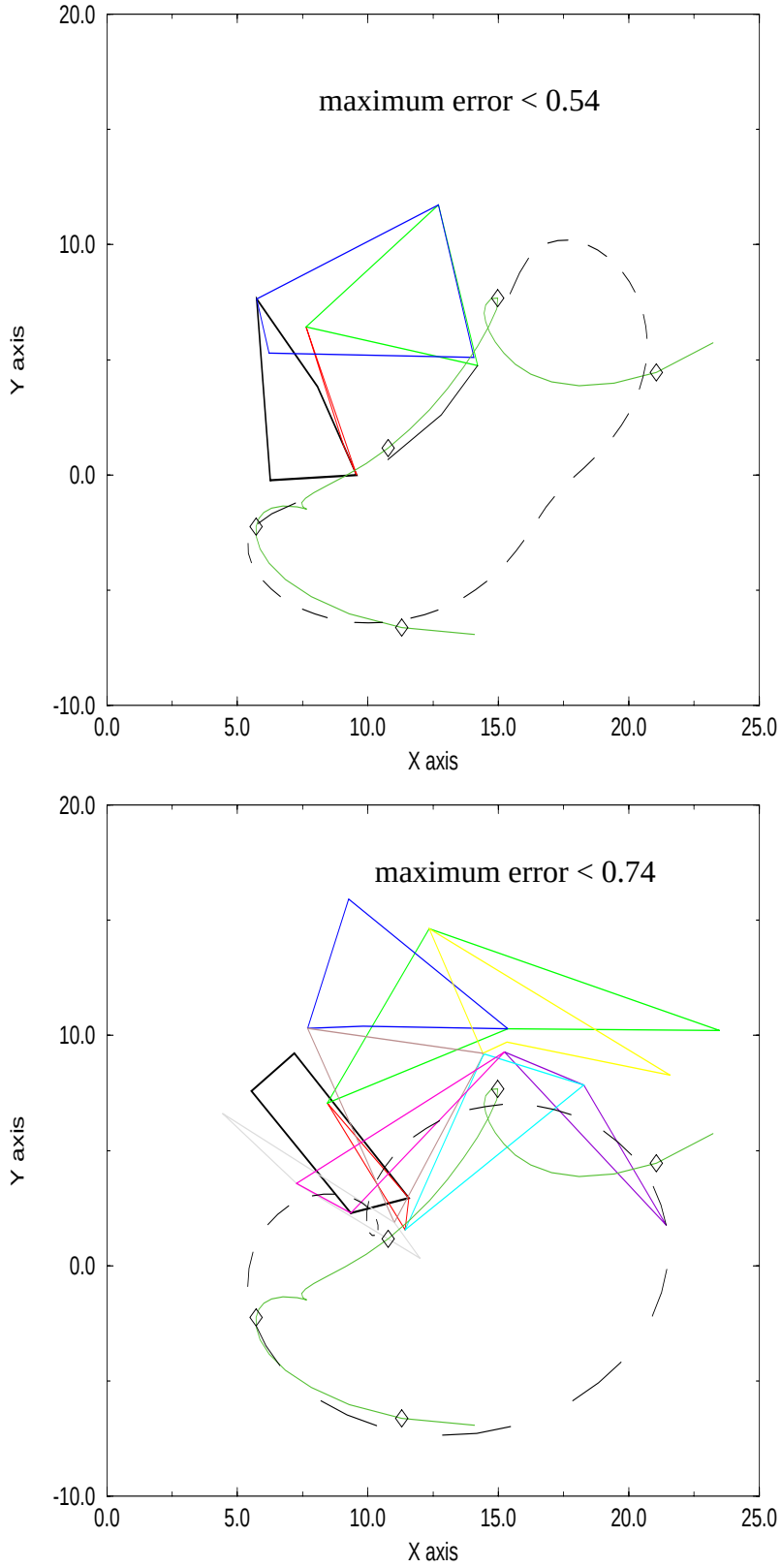


Figure 52. Planar mechanism synthesis, five points on an eight-bar coupler curve

6.2.3.3 Planar mechanism synthesis, thirteen specified points on a straight line

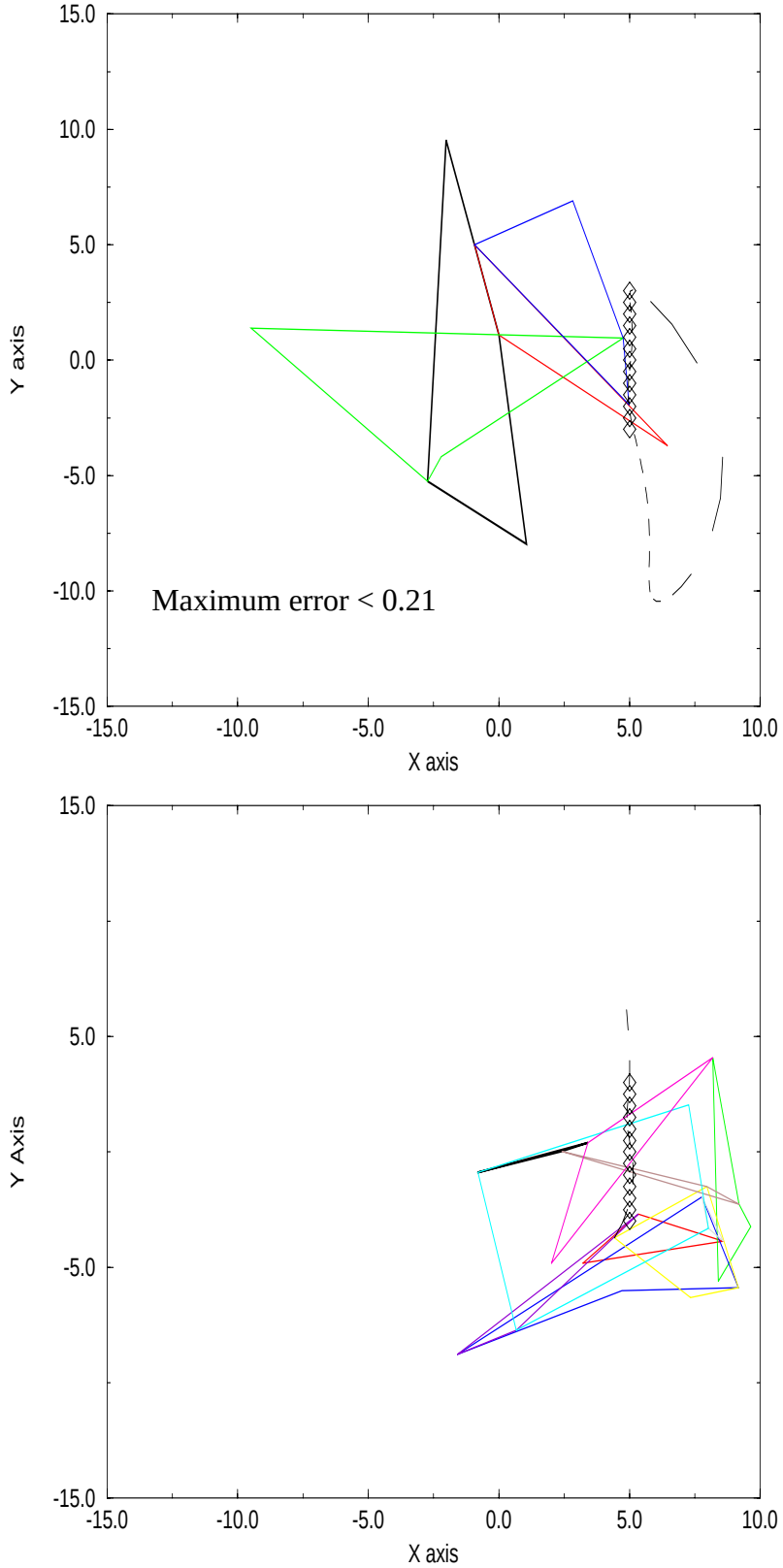


Figure 53. Planar mechanism synthesis, thirteen specified points on a straight line

6.2.3.4 Planar mechanism synthesis, 25 specified points on a straight line

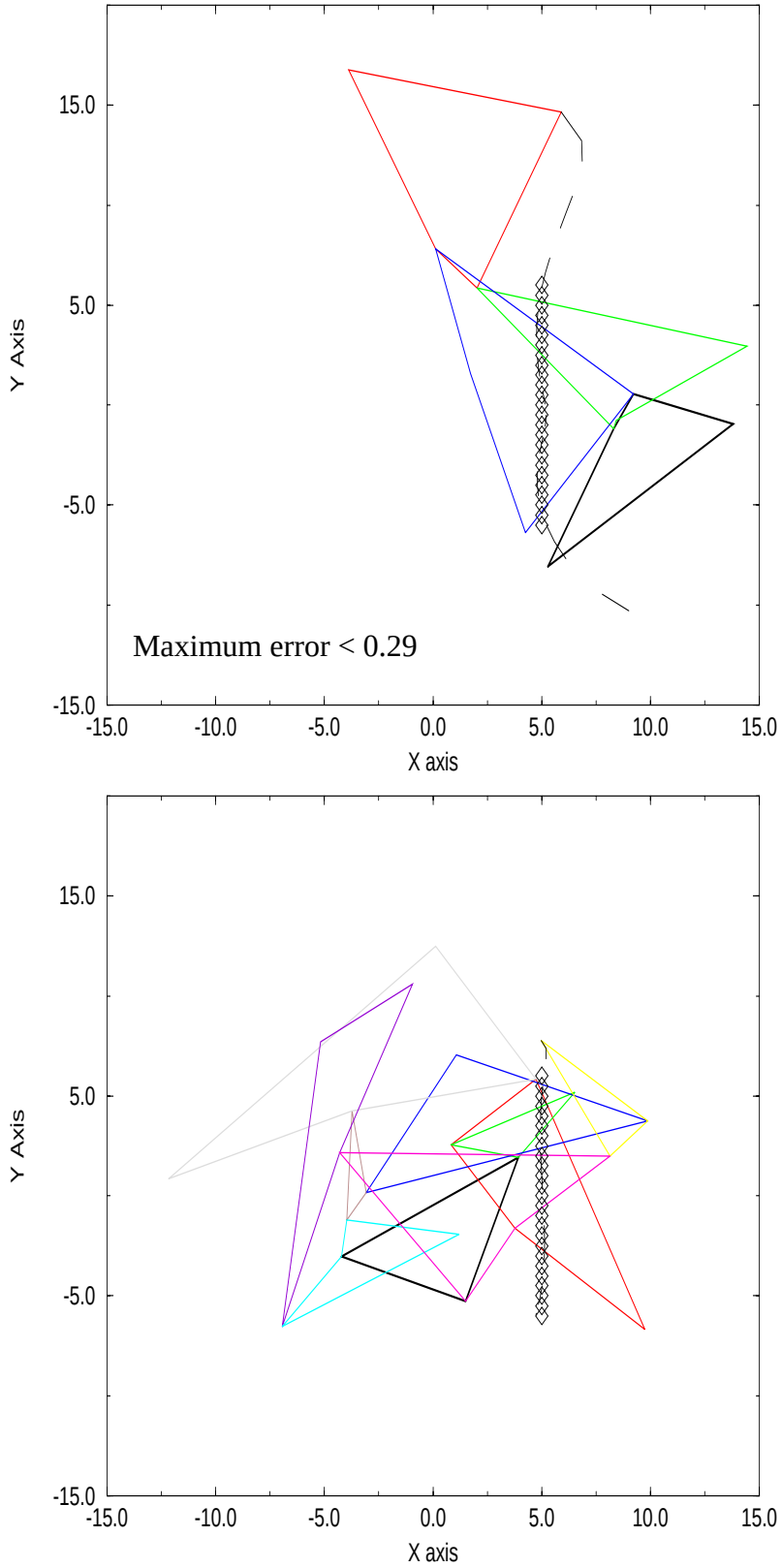


Figure 54. Planar mechanism synthesis, 25 specified points on a straight line

6.2.3.5 Discussion of experimental results

The results of the three experiments are reported in this section.

Planar synthesis, eleven specified points on a circle

The reason that good results were expected for this experiment is that it can be solved exactly with a two bar mechanism, the ground and a single rotating link. Thus, the expected result for this experiment would be to have one end of the ground link at the center of the circle with an appropriate link attached at that point to sweep out the circle. The rest of the mechanism is needed for generating the motion, and there are an infinite number of such mechanisms. However, to reduce computation time (for the general case), coupler points on links incident to the ground link are not examined. Thus, this type of solution cannot be found.

The first solution shown takes the form of a Stephenson II mechanism. Although it is not easy to discern, the brown link is a ternary link — it connects with the green, yellow and blue links. So, how does this mechanism generate a circular path? The left-hand sketch in Figure 55 shows a schematic representation of a Stephenson II mechanism. The links are labeled with the colors that correspond to the results being discussed.

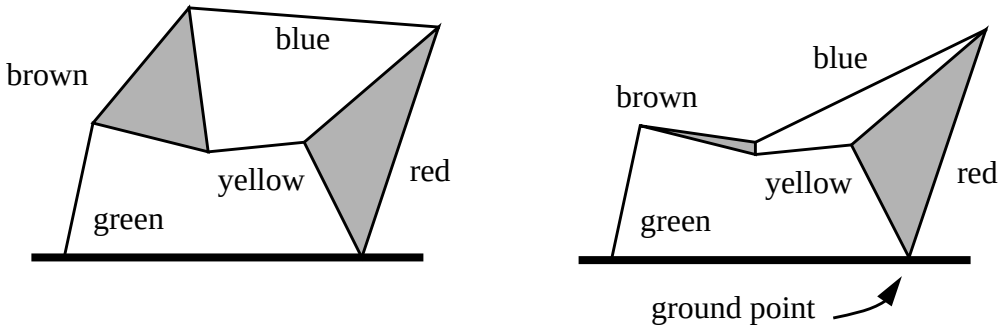


Figure 55. Stephenson II mechanisms - normal and degenerate

The right-hand sketch shows what has happened in this solution. The separation between the blue and yellow links, at the point of connection to the brown link, has become so small that these links can almost be considered to be joined together. This has the affect of “reducing” this six bar mechanism to a four bar mechanism. Thus, any point on the blue, red or yellow links can be considered to be rotating about the ground point, and this solution is equivalent to the form of the solution discussed in the previous paragraph.

The second solution shown is a four-bar mechanism. This mechanism approximates the circular path by reducing making the distance from the coupler point, which is on the blue link, very close to its connection point with the red link. As in the previous solution, this has the effect of reducing the mechanism to be kinematically approximate to the simple solution discussed previously.

Planar synthesis, five points on an eight-bar coupler curve

For this experiment, some of the control parameter values were modified. The number of elitists was increased from eight to 16 and the fitness multiplier was reduced from 2.0 to 1.8. In addition, the initial value for C was increased from 0.1 to 0.5. Since the initial neighborhood was made relatively large, the method for decreasing C was also changed. Instead of decreasing C by a fixed amount, see page 104, the amount of the decrease is given by k^n, where n is the number of times that the neighborhood has been reduced.

Unlike that data set used in the previous experiment, the points chosen for this experiment lie on a rather unusual coupler curve. Points on this curve were chosen because it seemed that this would provide a particularly challenging problem. A total of five experiments were performed. Two of the experiments were performed with the initial set of parameters, Table 19. Both experiments converged to eight-bar mechanisms and both only succeeded in satisfying four of the five specified points. After changing the parameters, as noted above, the results were better. The other three experiments yielded a four-bar, eight-bar and ten-bar solution, all of which succeeded in satisfying all five specified points. The maximum error reported for the ten-bar solution is larger than the actual maximum error, see explanation page 107. A better estimate of the maximum error for this experiment is 0.65.

One possible reason for the large errors is the relatively short number of generations used in these experiments. Examination of the output data indicates that the same mechanism is the best mechanism for an average of 20 or more generations. Compare this to an average of just more than one generation for the four-bar synthesis problem, Section 6.1. Three possible explanations for this include:

- The mechanisms are sensitive to parameter changes,
- Repeating the genetic operations until they succeed, see page 130, has the affect of increasing the probabilities of the genetic operations, thus the vast majority of the mechanisms are genetically operated on each generation,
- The technique devised of representing networks as trees increases the sensitivity of the mechanisms to parameter changes.

It is believed that the effects of all three explanations are occurring.

Planar synthesis, straight line motion

For these experiments, the same modifications to the control parameters mentioned in the previous experiment were used.

The results from this experiment were not as good as the results from the previous experiments or from the GA straight line experiments. Although the mechanisms shown in Figure 53 and Figure 54 appear to be good, the coupler curves through the area of interest are not particularly straight. That these results are not as good as the GA synthesis experiment results is expected because of the increased size of the solution space and the need to

establish the number and dimension parameters of the mechanism. Also, unlike the GA experiments, the input crank was not required to move through a full rotation, thus the coupler curves are not necessarily closed paths.

6.2.4 Planar mechanism synthesis summary

These experiments represent the most arduous testing of GD. This testing revealed several things, including:

- The need for copious computing resources. The computing time required is strongly affected by the total number of links in the population. For populations comprised primarily of four-bar mechanisms, the time required to generate a solution with a population of 400 for 150 generations was approximately five hours on a Sparc 20/61. For population comprised primarily of ten-bar mechanisms, the computing time exceeded 30 hours.
- How the representation can bias the results. As previously mentioned, it appears that a judicious choice of the control parameters can mitigate this somewhat.
- That within a class of artifacts, some are more likely to be generated than others. In these experiments, this came about for two reasons. The first is illustrated in Table 17, in which the means of randomly generating the artifacts can introduce a bias. The second is explained on page 130, in which the genetic operations can amplify the previously mentioned bias. Engineers using GD must be aware that biases potentially exist and must endeavour to structure the representations to minimize their affects.

The need for scaling has not been previously mentioned with respect to these experiments, but as with the GA synthesis, it is a very important issue. Consider a case similar to that shown in Figure 40. If the specified points are close together, a single point on a coupler curve might satisfy all of the points, but the mechanism thus generated will certainly not provide the desired motion. Extrapolating from this idea, it is suggested that the closer together the specified points are to each other, the higher the probability of finding a solution. As the points move further apart, the probability of finding a solution may decrease.

Consider the following example of trying to find a mechanism to pass through eleven specified points.

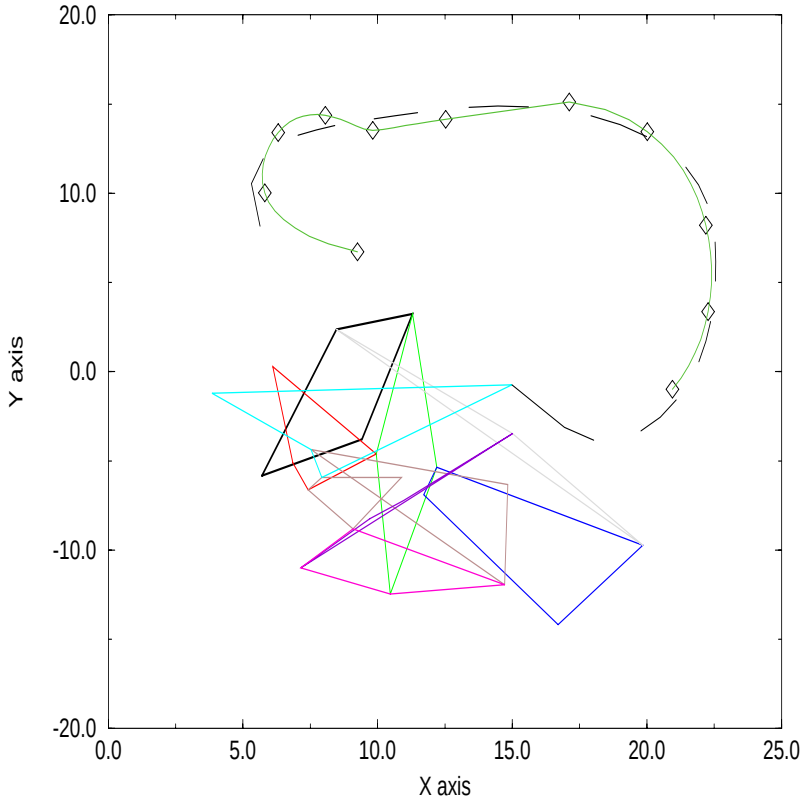


Figure 56. Planar mechanism synthesis, the need for scaling

Unlike the straight line experiments in which the points were space 0.5 units apart, in this experiment, the average spacing is approximately 4 units. Scaling alone may account for the fact that one of the points is not satisfied.

One possible way to overcome some of these difficulties is to specify the same point more than once. For the example shown in Figure 56, specifying the missed point twice would increase the “penalty” for missing this point, thus increasing the likelihood that it would be satisfied. Of course, this may cause some other points to be missed.

7 Summary and conclusions

7.1 Summary

The thesis presents a new methodology for the design of real-world systems. Although there already exists a large number of design methodologies, genetic design is unique in its ability to model complex systems, to automatically evaluate these models and to present the designer with an array of viable design alternatives. This is done by using a formal grammar representation of the object being modeled, incorporating the artifact controller into the model and using genetic programming techniques for model manipulation.

The genetic design methodology was tested in two problem domains: the design of a walking vehicle and the design of planar mechanisms. In both problem domains, the methodology provided good solutions to the specific design problems. Although neither problem domain is particularly challenging, the ability of the methodology to solve these problems is indicative of its ability to solve more complex problems, see Section 7.2.

In the course of this dissertation work, contributions in several fields were made:

- The genetic design methodology for the design of real-world systems, Chapter 4.
- The need for the simultaneous development of a system and its controller to achieve optimal performance, Section 4.3.
- The hybrid GA/GP technique for numerical parameter representation and manipulation, Section 3.3.
- The explicit coupling of formal grammars with genetic manipulation through the use of strongly-typed genetic programming, Section 4.2.
- A genetic algorithm based technique for four-bar mechanism synthesis, Section 6.1.
- A genetic design based technique for the simultaneous type, number and dimension synthesis of planar mechanisms, Section 6.2.

7.2 Future work

The main thrust of this thesis work has been to present a new design methodology. This methodology was tested in two different problem domains, the stepping stone walker and planar mechanism synthesis. These models were selected for two reasons: relevance to robotics and relative simplicity. Although these are relatively straight-forward problems, they do show that the methodology is viable. However, to be more widely used, this methodology requires further testing and it must be applied to more complex problems. In addition, benchmarks must be developed to help predict computing requirements for potential applications. This section will focus on future testing and applications of this methodology. This section will conclude with some speculative comments about the GD methodology.

Further experimentation is necessary to determine if this methodology is truly capable of handling such complex tasks as vehicle design. Future experimentation can be roughly divided into three categories. The first is that work which follows directly from the

work presented here. During this period, model complexity should incrementally increase to ensure that the methodology continues to function properly. The second is that work which represents new application fields in which the GD methodology can be applied. To effectively apply GD to other problem domains, tool sets that allow the methodology to be broadly applied will need to be developed. The third is to apply the latest research techniques being developed for GP to GD.

In continuing this body of research, there are many viable, alternative courses of action that can be pursued. The following prescription is just one of many.

Further testing

One of the first continuations of this work would be to extend the stepping stone walker into a two dimensional model. In this model, another degree of freedom would be added to the frames, the ability to displace vertically with respect to each other. Another change might be to change the meaning of the pad width parameter into a leg length parameter. Similarly, the three “sensors” used would be replaced by a sensor that reports the terrain elevations of the terrain in front of the vehicle. To simplify the model, the constraint that the body must be kept level at all times could be applied. This constraint derives from real-world considerations of leg sizing and planning simplicity. The controller for this vehicle would have to specify both the vertical and horizontal displacement of the moving frame. This model is actually a fairly good representation of an actual frame walking robot because the path traversed by an actual rover can be envisioned in two-dimensions (by taking a cross-section of the terrain). Further enhancements could include foot/terrain interactions. Some work has already been done in the area of genetically developing gait planners, see [48], chapter 15.

A possible next phase of development would be to expand the model into three dimensions. For simplicity, this model may continue to abstract the actual means of attachment of the vehicle’s frames and may continue to require that the body be kept horizontal. Since all testing to date indicates that two frames are sufficient for all terrains, the model may be limited to two frames. The controller for this model would have to command the horizontal, vertical and angular displacements for each of the frames. The controllers for the one and two-dimensional models have not had to consider where to move - they simply tried to move the vehicle across terrain. As such, these controllers are gait planners. For the three dimensional model, the controller needs to be able to plan both the gait and the path that should be traversed.

Incorporation of higher level controllers

It is not reasonable to assume that by starting with simple mathematical primitives that deriving a path planner, using GP, will occur. However, in the same manner that the sensors are modelled and not required to be generated from primitive elements, path planners could be generated similarly. That is, the planner could have access to global terrain

maps and other sensor inputs. The primitives for the path planner could be different types of planning primitives, such as obstacle avoidance routines. The outputs from this planner would be paths, expressed in global coordinates, for the rover to follow. The gait planner would then be required to generate vehicle motions to allow the vehicle to traverse the commanded path. Research in the area of path planning for both wheeled and legged vehicles is being actively pursued. [100] [105]

Vehicle design

The next phase of development would be to incorporate configurations other than frame walkers into the model. This phase is potentially the most difficult because different configurations will require entirely different controllers. What this implies is that the connection between the controller and the actuators will have to be made more explicit than it has been in the earlier models.

Another challenge that arises with the three dimensional model is the model evaluation itself. For the two dimensional model, the dynamics of the vehicle/terrain interaction can be managed quite easily [4]. In moving to a generic vehicle in three dimensions, the problem is significantly more challenging. First, rigid body dynamics is far more complex than planar dynamics. Second, developing techniques for calculating the dynamical interactions of general objects in three dimensions is a topic of current research.

If this phase of development succeeds in producing vehicles that traverse terrain models (based on actual terrains) the methodology can be considered to be a success.

One real-world problem that could be solved at this stage is the vehicle configuration problem for a lunar rover. The constraint for this problem include vehicle mass, stowage volume, average ground speed, locomotion power consumption and antenna pointing. The requirement for antenna pointing is to aim the antenna at some spot (typically the Earth) and to maintain lock with some fairly high precision. For a stationary vehicle this is a simple task, however, for a vehicle moving over an unstructured terrain, the task is quite challenging. What makes this problem even more interesting is that there is a trade-off between transmitter power and antenna accuracy, thus the model would need to incorporate the power consumption of the transmitter and the antenna pointing system.

GD methodology development

In the course of developing these models (and for that matter, during the course of all future work), two fundamental questions need to be addressed: What are the appropriate functions to evaluate and how are input disturbances handled? For the stepping stone walker problem, the evaluation incorporated functions relating to the vehicles performance, size and speed. Since this is an abstract problem, the designer was free to choose those functions that seemed appropriate. However, in real-world designs, the choice of evaluation functions may be far more difficult. For the lunar roving vehicle, although the primary goal may be to accurately point the rover's antenna, the ability to traverse the terrain in a power

efficient manner and the necessity of building a vehicle that fits within the confines of a launch vehicle are also of paramount importance.

For a system to be robust, it must be capable of handling input disturbances. However, as demonstrated in Section 5.4, the introduction of disturbances must be done in a controlled manner to achieve reasonable results. In Section 5.4, two means of introducing disturbances were explored and a third discussed. Each of these methods must be explored in greater detail. In addition, the entire scheme of controllers may need to be revisited to allow greater flexibility. Consider a mountain-climber. To reach the base of the mountain, the climber uses an ordinary walking gait. To climb the surrounding foothills, a slower, more deliberate gait is used. Finally, to ascend sheer faces, a "four-footed" gait is employed. Similarly, systems under-going development may require a selection of alternative control modes to effectively reject input disturbances.

Artifact evaluation

One of the central issues in this work is the function for evaluating the artifact. In nature, the evaluation is the ability of the organism to procreate; in design, it is some user defined function. The difficulty comes in determining not only what artifact outputs should be evaluated, but how multiple objectives are to be combined into a single evaluation function. While it is probable that determining general rules to apply to the evaluation of all conceivable artifacts may not be possible, some work should be done to determine if some general guidelines can be discovered. The ideas presented in Section 4.4 may serve as a basis for these guidelines.

In Section 5.4, the model of the stepping stone walker was augmented with a vehicle level controller. To be effective, this controller should be terrain-adaptive, a requirement that necessitates some means of gathering information about the terrain, i.e., sensors. In Section 5.4, three sensors were made available to the vehicles at no cost to the vehicle. The term "no cost" implies that the inclusion of sensors did not cause a reduction in the evaluation function. This is reasonable since the function does not evaluate a vehicle cost. For real-world systems, the inclusion of sensors can have a major impact on the system design and cost. For example, the inclusion of cameras to perform stereographic terrain mapping necessitates the addition of fairly significant amounts of computing hardware and software. For certain systems, reduced performance may be acceptable given the greatly increased cost for the inclusion of sensors with it associated increase in performance. To model realistic system, some notion of cost should be incorporated into the evaluation and the cost of the sensors appropriately accounted for.

One of the limitations of GD is the large computing resource required to solve even moderately complex problems. As problems become more complex, it can be expected that the resources needed to solve the problem will also increase. In addition, as the problems get more complex, and the size of the design space increases, it may become necessary to work with larger populations to ensure a proper sampling of the design space. Unfortu-

nately, the continuing increase in computing power may not be sufficient to meet future needs. Consider the planar synthesis problem, Section 6.2. In those cases where the population became dominated by higher-order mechanisms, the evaluation time for a single experiment was upwards of 20 hours on a Sparc 20/61. The reason for this long execution time is that the Sparc is a serial computer — it can only execute a few instructions at a time (neglecting its super-scalar capabilities which have little impact globally). To improve performance, parallel processing machines are needed. On way to take advantage of these parallel processing machines is to run each individual on its own node, thereby tremendously reducing the time to produce an answer. Since the vast majority of the computing resources are used for evaluation, if the evaluation procedure is necessarily complex, there does not appear to be a means for reducing the amount of computing resources needed.

One of the unexpected results of this work was to discover that the higher-order mechanisms, Section 6.2, were more sensitive to genetic operations than the lower-order mechanisms. This discovery required some changes to the planar synthesis program that would not unfairly bias the results in favor of the lower-order mechanisms while at the same time not introducing any artificial means for encouraging higher-order mechanisms. Although this problem of biases was not observed in the other experiments performed, it may be a fairly common problem when designing complex systems. Before attempting to develop guidelines for dealing with biases, it must first be determined if these biases occur frequently in other problem domains. If they do, the following two questions will need to be addressed. First, are the biases due to the means of representation/evaluation or are they truly representative of the problem? For example, if the stepping stone walker problem had shown a bias towards vehicles with two frames, this may have been acceptable since vehicles with two frames are the preferred solutions to the problems studied. Second, can some means be implemented to reduce the impact of biases on the problem without introducing artificial means? For example, an improper means to reduce the biases in the planar synthesis problem would have been to multiple the fitness by the number of links.

One of the current areas of research in GP is in the area of automatically defined functions (ADF). ADF are an extension of encapsulation, Section 4.5, page 54, in which the encapsulated functions are capable of further evolutionary change. The idea of ADF can be applied to the design of artifacts. For example, for the planar synthesis problem, a particular dyad could be encapsulated and higher-order mechanisms could be built from these dyads as well as from links. For more complex systems, the use of ADF would seem to offer great advantage in the sense that successful groups of components would be available throughout the population, not just the components themselves. For example, in the biological world, the eukaryotes, which are believed to have been formed from prokaryotes acting symbiotically, have become the basis for the evolution of more complex species as opposed to the more basic building blocks, the prokaryotes.

Long-range goals

A final area of further work is to consider alternate types of genetic operators. In the biological world, the result of a genetic operation produces a viable organism of the same species as it parents. This behavior is also seen in GP/GD where the “organisms” that evolve tend to do so in such a manner as to make the “organism” more capable of surviving genetic operations. This tendency limits the design space examined by the genetic operators. To overcome this tendency, either new operators should be developed or some alternate means of exploring wider areas of the design space should be employed.

Of course, the ultimate design tool would allow the tool itself to evolve. This idea is not as far-fetched as it may seem as this is what occurred in the biological world. By using sufficiently general means of representation, such as programs that represent the means to build an artifact, as opposed to representing the artifact itself, the future capabilities of GD are limitless. Of course, at some point in the future, the design system used will not be GD, but rather some system that may have evolved from GD. In that sense, GD could represent the first tentative step down a new path of discovery and invention.

A Stepping stone vehicle terrain generation algorithm

This appendix shows the algorithm used for generating the terrain for the stepping stone vehicle experiments, presents some graphical examples of these terrains and shows how these terrains are related to actual terrains.

Figure A1 shows a fragment of the actual computer code used to generate the terrain. Some of the details, such as type casting has been omitted for simplicity.

```
for (i=0, tc=0; i<numberOfTerrainSegments; i++) {
    for (j=0; j<trData[i].cycles; j++) {
        tc += trData[i].period + trData[i].periodBlur * gasdev ();
        tw = trData[i].period * (trData[i].pcntg + trData[i].pcntgBlur * gasdev ());
        if (tw < 0)    tw = 0;
        ts = (tw + trData[i].period * trData[i].pcntg) / 2;
        for (k=tc-ts; k<tc+tw-ts; k++)   terrain[k] = 1;
    }
}
```

Figure A1. Code fragment for terrain generation

The data type trData is a structure that holds the five terrain parameters, Chapter 5, Figure 23, for each of the terrain regions. These parameters are, for the *i*th terrain segment:

- trData[i].cycles - the number of terrain cycles
- trData[i].period - the terrain period
- trData[i].perriodBlur - the standard deviation of the terrain period
- trData[i].pcntg - the percentage of the terrain that is in the step region
- trData[i].pcntgBlur - the standard deviation of the percentage of the terrain that is in the step region

In addition to these parameters, there are seven variables

- i - keeps track of the current terrain segments
- j - keeps track of the number of cycles in the current segment
- k - counter for assigning terrain
- tc - the location of the nominal end of the current terrain cycle in global terrain coordinates
- tw - the width of the current terrain cycle
- ts - the location of the start of the current terrain cycle relative to tc
- terrain - an integer array that is the terrain

and one external routine

- *gasdev* - returns a Gaussian distributed random variable with a standard deviation of 1.0.

To generate a terrain, first, the value for each terrain location is set to zero, except for the two end-zones. Then, the above procedure is repeated until tc exceeds the specified terrain size. One non-obvious construct is the use of the parameter ts. This parameter could be eliminated entirely by starting the step-region a distance of tw before tc. This would have the affect of keeping the right edge of the step region fixed, that is, if $tw > 0 \forall tc$, then $terrain[tc] \neq 0 \forall tc$. What is done instead is to keep the midpoint of the step region fixed, that is, if $tw > 0 \forall tc$, then terrain[tc-%age*period] $\neq 0 \forall$ tc. This latter method was chosen because it was felt that it would produce more interesting terrains.

To illustrate the terrain generation procedure, four examples are considered. Table A1 summarizes the terrain parameters for each of these examples:

Example	Segments	Cycles	Period	PeriodBlur	Pnctg	PnctgBlur
1	1	3	2	0	50	0
2	1	3	2	0	50	25
3	1	3	2	0.5	50	0
4	1	3	2	0.5	50	25

Table A1: Terrain parameters for example terrains

For each of the example, the values for the variables, tc, tw and ts are listed in Table A2. All values have been rounded to the nearest 0.05 so that the results can be displayed to scale:

Example	Cycle	tc	tw	ts
1	1	2.00	1.00	1.00
	2	4.00	1.00	1.00
	3	6.00	1.00	1.00
2	1	2.00	0.85	0.90
	2	4.00	0.90	0.95
	3	6.00	1.35	1.20

Table A2: Variable values for examples

Example	Cycle	tc	tw	ts
3	1	2.35	1.00	1.00
	2	4.65	1.00	1.00
	3	6.15	1.00	1.00
4	1	1.30	1.25	1.15
	2	3.55	0.70	0.85
	3	5.00	0.90	0.95

Table A2: Variable values for examples

These terrains are shown in Figure A2, where the step regions are shown as dark lines:

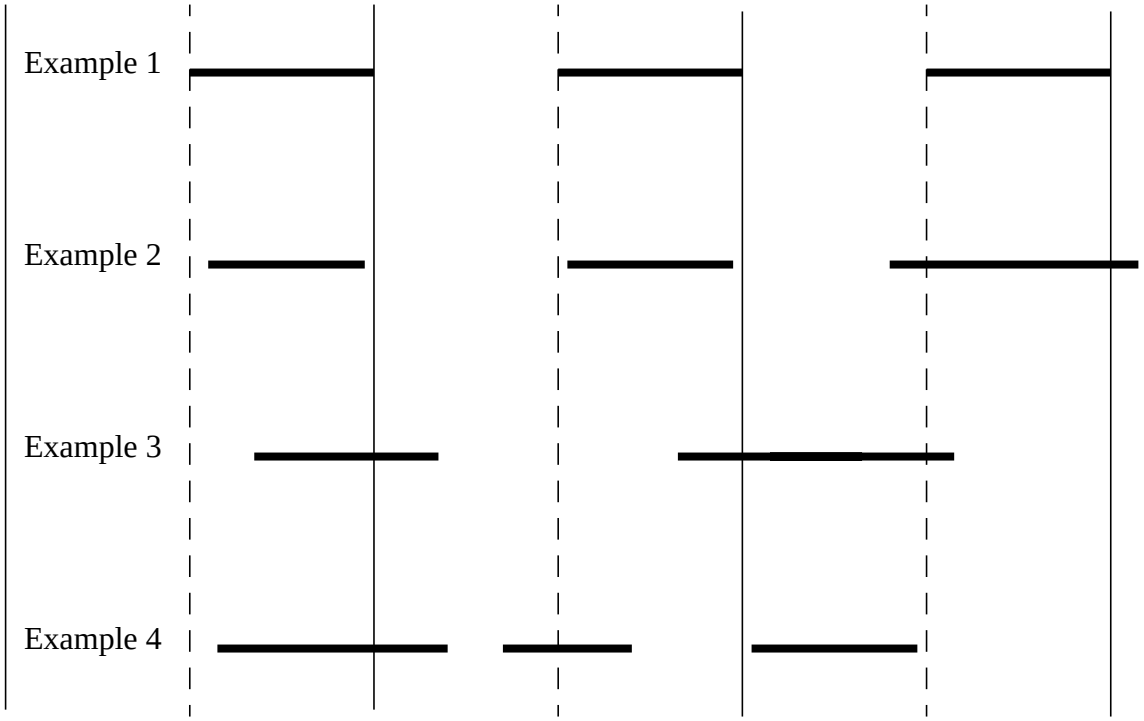


Figure A2. Terrain examples

B Complex controllers

This appendix presents, without explanation, those vehicle level controllers from Section 5.4 that were identified as being complex.

Controller from Section 5.4.1, Run 1

(+ (+ {(/ (* s3 (* {(+ {(/ (/ (- (/ 82 (* s3 (+ (+ 61 (* 21 252)) (* 52 (* {(+ (* s1 (* s2 185)) 113) < 0 (+ (* s3 (* 189 s3)) (* (* 61 84) 2)) s3 } (* 9 (/ (- 80 217) 239)))))))) s1) (+ 67 240)) 123) > 0 s1 s2 } (+ (* s3 (- (+ 45 (* 81 223)) {(* 35 34) > 0 (- (+ 154 126) 187) {{s2 > 0 157 (* 134 244) } = 0 {(* (* 127 s3) (- s1 163)) < 0 (/ (* (* 95 212) {s2 > 0 (+ 254 178) (/ (* 105 82) s3) }) s1) {(* 94 {(- 43 (* (* 108 220) 2)) < 0 s2 s3 }) = 0 s1 254 } } (/ 209 (* 108 (+ (+ 57 (* 31 34)) (* 21 (* {(+ (* s1 (+ (+ 126 (* 188 217)) 161)) 177) < 0 (+ 147 (* (* 248 212) 154)) s3 } (+ 196 (* 133 s1))))))) } })) 151)) < 0 s2 s3 } (+ (+ 12 (* 64 (- (/ 200 (* s3 (+ (* 115 150) (* 82 (/ s3 (+ 210 (* 209 s2)))))))) (* (* 202 238) 52)))) {(* 179 32) > 0 (+ 251 127) {{s2 > 0 96 (* 134 244) } = 0 {(* (* s2 255) (- s1 131)) < 0 (/ (* (* 31 209) {s1 > 0 219 (/ (* 105 19) s3) }) s1) {{(/ s1 (+ 90 (* 92 245))) < 0 210 s1 } = 0 s1 {s1 > 0 (+ 226 72) (/ (* 56 154) s3) } } } (/ 51 (- (/ 201 (* s3 (+ (+ 76 (* s3 98)) (* 53 s3)))) (/ (* 179 192) 196))) } }))) 115) < 0 s2 s3 } (+ 21 (* s3 {(* 55 150) > 0 (+ (+ 255 87) (+ s2 100)) {{s2 > 0 78 (* 105 174) } = 0 (* (* s2 206) (/ (+ 115 (* 92 151)) 179)) 22 } }))) 141)

Controller from Section 5.4.3, Run 3

(+ (+ (- s3 214) {s2 = 0 (+ s1 {(+ s2 (* 64 111)) = 0 {(* 162 {(* s3 140) > 0 s2 {s1 > 0 s1 141 } }) > 0 {{{(* s3 (- (- 13 194) (- s2 (* 193 25)))) = 0 (- (* (+ s1 24) s2) 35) {(* {(* (* (- s2 19) 199) s3) = 0 s3 s1 } (+ s2 103)) = 0 (/ 35 s2) s1 } } > 0 12 0 } > 0 s3 {{(/ s3 (- (* s1 s3) s3)) = 0 (- s3 164) (/ s1 (+ s2 {s2 > 0 s2 111 })) } > 0 (* (- {s1 = 0 254 233 } {(/ (- s2 84) 39) = 0 {(- 45 197) > 0 s3 37 } s1 }) s3) 1 } } 161 } (- (* 113 210) (+ s2 (/ 182 {s1 > 0 (- 70 143) 0 }))) }) (+ (- (* {(+ 235 s3) < 0 47 80 } s3) 19) s1) }) s2)

Controller from Section 5.4.6, Run 1

(- (* (+ (* 250 210) (/ {(/ (* s1 120) 55) < 0 s1 (/ (/ {s3 = 0 (/ s1 (* s1 (/ s2 (/ (+ s3 (* (+ 247 s2) (/ (* s3 (/ (- {{s3 > 0 s2 {(* 169 153) = 0 133 232 } } < 0 (- s1 (* 95 154)) (* (* (* 244 62) 131) s1) } (* (- s1 (- (+ (* 244 103) (+ (/ s2 (/ s3 (* s3 (- (+ s2 62) s2)))) (+ (- (- (+ (* 202 59) (- s3 {(- s3 s2) < 0 181 (+ 215 (* 173 96)) })) 17) 15) {s3 = 0 157 s3 }))) s3)) (+ (- s3 216) (* s3 (/ (- {{s3 > 0 s2 {(+ 173 217) = 0 133 232 } } < 0 s3 (* (* (* 215 229) 114) s1) } (* (- s1 (- (+ (* 54 67) (+ (/ s2 (/ s3 (* s3 (- 190 s2))))) (+ (- (- (+ (* 234 187) (- s3 {(- s3 s2) < 0 155 (+ 87 (* 173 96)) })) 19) 15) {s3 = 0 157 (- (* (+ (* 209 67) s2) s1) (- (/ s3 (- s3 (/ s2 (/ (/ s3 (* (* 202 s2) 3)) s3)))) s3)) }))) s3)) (+ (- s3 88) s2))) (* 161 {(- s3 (/ s2 (- s2 252))) = 0 (/ (* (* 170 173) 172) (- 198 (* s3 89))) (+ s3 (/ {s2 = 0 s1 7 } 160)) })))))) (* 177 {(- s3 (/ s2 (- s2 222))) = 0 (/ (* (* 202 37) 172) (- 206 (* s3 126))) (+ s3 (/ {s2 = 0 s1 71 } 160)) }))) (+ (+ (/ (+ 196 102) 151) {(* 21 104) > 0 s2 109 }) s3)))) s3)))) s1 } (* {s3 = 0 s3 (+ 197 98) } 77)) (+ {s1 = 0 s2 (- 158 s1) } (- (* (* 81 s1) (+ (/ s2 (/ s3 (* s3 232)))) 170)) (* (+ 98 s1) (- s1 207)))))) } (- {s2 > 0 {s3 > 0 93 (* (* 246 45) 100) } {(- 130 223) > 0 173 96 } } s1))) s1) (- (/ (/ s2 (/ (/ s3 (* (* s1 s2) (* s2 (/ (* 213 (+ (- s2 253) s2)) (* 193 (- 198 216))))))) s3)) (+ (- s2 234) s2)) s3))

C Acronyms used

This appendix presents an alphabetical listing of all acronyms used in this thesis. Each acronym is listed, along with what it stands for and the page reference where it is first used.

acronym	what it stands for	page used
GD	Genetic Design	1
GA	Genetic Alorithms	4
GP	Genetic Programming	4
NASA	National Aeronautics and Space Administration	5
JPL	Jet Propulaion Laboratory	5
MMC	Martin Marietta Corporation	5
FMC	Farm Machinery Corporation	5
MRSR	Mars Rover Sample and Return mission	5
ASV	Adaptive Suspension Vehicle	6
CFG	Context Free Grammar	11
CSG	Context Sensitive Grammar	11
STGP	Stongly Typed Genetic Programming	13
TBD	Task Based Design	16
BNF	Backus-Naur Form	22
SFM	Single Figure of Merit	48
MTBF	Mean Time Between Failures	51
CPM	Control Parameter Multiplier	59
VLC	Vehicle Level Controler	83
ADF	Automatically Defined Functions	148

References

[1] Ia. S. Ageikin. *Off-the-Road Mobility of Automobiles*, volume 56 of *Russian Translations Series*. A. A. Balkema, Rotterdam, 1987.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley Publishing Company, 1986.

[3] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–64, January 1994.

[4] D. Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. Ph.D. thesis, Technical Report 92-1275, Computer Science Department, Cornell University, 1992.

[5] J. Bares. *Orthogonal Walkers for Autonomous Exploration of Severe Terrain*. Ph.D. thesis, Department of Civil Engineering, Carnegie Mellon University, May 1991.

[6] M. Bekker. *Theory of Land Locomotion*. University of Michigan Press, 1956.

[7] M. Bekker. *Introduction to Terrain-Vehicle Systems*. University of Michigan Press, 1969.

[8] Richard K. Belew and Lashon B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1991.

[9] R. Beyers and S. Desa. An optimization-based framework for simultaneous plant-controller redesign. *Journal of Machine Design*, 116(2):396-404, 1994.

[10] Roger Bostelman et al. A stewart platform lunar rover. In *Robotics for Challenging Environments*, Albuquerque, NM, 1994.

[11] Braitenberg, Valentino. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1989.

[12] Don R. Brown and Kuo-Yen Hwang. Solving fixed configuration problems with genetic search. *Research in Engineering Design*, 5:80–87, 1993.

[13] M. Buckley. Multicriteria evaluation: measures, manipulation, and meaning. *Environment and Planning B*, 15(1):55–64, 1988.

[14] Jonathan Cagan and Alice M. Agogino. Innovative design of mechanical structures from first principles. *AI EDAM*, 1(3):169–189, 1987.

[15] Guanxiong Cha, Ravi S. Sastry, and Shin-Min Song. A comparative study of leg mechanisms for walking machine design. In *1st National Conference on Applied Mechanisms and Robotics*, Cincinnati, Ohio, November 1989.

[16] Jeffrey E. Chottiner. Simulation of a six wheeled martian rover called the rocker bogie. Master's thesis, Ohio State University, 1992.

[17] J.J. Craig. *Introduction to Robotics, Mechanics and Control*. Addison-Wesley Publishing Company, Menlo Park, California, 1986.

[18] J. Denavit and R.S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, pages 215–221, June 1955.

[19] Gan Dongying. Kinematics of six-legged vehicles on irregular terrain. In *ICAR*, pages 389–396, 1985.

[20] C. L. Dym. Representation and problem-solving: the foundations of engineering design. *Environment and Planning B*, 19(1):97–105, 1992.

[21] C.F. Earl and J. Rooney. Some Kinematic Structures for Robot Manipulator Designs. *Journal of Mechanisms, Transmissions, and Automation in Design*. 105:15-22, March 1983.

[22] H. Eschanauer, J. Koski, and A. Osyczka, editors. *Multicriteria Design Optimization*. Springer-Verlag, 1990.

[23] FMC Inc. Mars rover sample return (MRSR): Studies of rover mobility and surface rendezvous, August 1988. Work performed under JPL contract 958074.

[24] Susan Finger and John R. Dixon. A review of research in mechanical engineering design. part I: Descriptive, prescriptive and computer-based models of design processes. *Research in Engineering Design*, 1:51–67, 1989.

[25] Susan Finger and John R. Dixon. A review of research in mechanical engineering design. part II: Representations, analysis and design for the life cycle. *Research in Engineering Design*, 1:121–137, 1989.

[26] S. Finger and J. R. Rinderle. A transformational approach to mechanical design using a bond graph grammar. In *Design Theory and Methodology*, 1989.

[27] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, 1993.

[28] Stephanie Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1993.

[29] D. R. Freitag. History of wheels for off-road transportation. *Journal of Terramechanics*, 16(2):49–68, 1979.

[30] F. Freudenstein and E. R. Maki. The creation of mechanisms according to kinematic structure and function. *Environment and Planning B*. 6:375-391, 1979.

[31] Alan Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[32] Thomas D. Gillespie. *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, 1992.

[33] J. Gips and G. Stiny. Production Systems and Grammars. *Environment and Planning B*. 7:399-408, 1980.

[34] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[35] David E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, George Mason University, June 1989.

[36] David E. Goldberg, Bradley Kord and Kalyanmoy Deb. Messy Genetic Algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5): 493-530, 1989.

[37] Kurt Hain. *Applied Kinematics*. McGraw Hill, 1967.

[38] Richard S. Hartenberg and Jacques Denavit. *Kinematic Synthesis of Linkages*. McGraw-Hill, 1964.

[39] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1975.

[40] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison Wesley Publishing Company, 1969.

[41] L. M. Howard and D. J. D'Angelo. A fusion of genetic algorithm and genetic programming techniques for symbolic regression. To be published - obtained through private communique with author.

[42] M. Huang and K. Waldron. Relationship between payload and speed in legged locomotion systems. *IEEE Trans. Robotics and Automation*, 6(5):570–578, October 1990.

[43] Iwasaki, Mineo et. al. Development on aquatic walking robot for underwater inspection. *Report of the Port and Harbour Research Institute*, 26(5):393–422, December 1987.

[44] De Jong, K.A. and Spears, W.M. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5(1):1-26, April 1992.

[45] Thomas R. Kane and David A. Levinson. *Dynamics: Theory and Application*. McGraw Hill, New York, 1985.

[46] Makoto Kaneko, Susumu Tachi, Kazuo Tanie, and Abe Minoru. Basic study on similarity in walking machine from a point of energy efficiency. *IEEE Journal of Robotics and Automation*, 26(5):19–30, December 1987.

[47] Jin-Oh Kim. *Task based kinematic design of robot manipulators*. Ph.D. thesis, Carnegie Mellon University, August 1992.

[48] Kenneth E. Kinnear. *Advances in Genetic Programming*. The MIT Press, 1994.

[49] P. R. Klarer and J. W. Purvis. A highly agile mobility chassis design for a robotic all-terrain lunar exploration robot. In *American Nuclear Society's Fifth Topical Meeting on Robotics and Remote Systems*, Knoxville, TN, 1993.

[50] C. Klein, K. Olson, and D. Pugh. Use of force and attitude sensors for locomotion of a legged vehicle over irregular terrain. *International Journal of Robotics Research*, 2(2):3–13, 1983.

[51] Stephen J. Kline. *Similitude and Approximation Theory*. Springer-Verlag, Berlin, 1986.

[52] John. R. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[53] John. R. Koza. *Genetic Programming II*. MIT Press, Cambridge, MA, 1994.

[54] J. C. Larminie. Standards for the mobility requirements of military vehicles. *Journal of Terramechanics*, 253:171–189, 1988.

[55] Levy, Steven. *Artificial Life: The quest for new creation*. Pantheon Books, 1992.

[56] Lohmann, R. Structure evolution and incomplete induction. *Biological Cybernetics*, 69:319-326, 1993.

[57] Lohmann, Reinhard. Applications of structure evolution. *Japan-USA Symposium on Flexible Automation*, 1293-1300, Kobe, Japan, Jully 1994.

[58] Maher, M.L. and Kundu, S. Adaptive design using a genetic algorithm. *IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design*, 211-228, Tallinn, Estonia, June 1993.

[59] Benoit Mandlebrot. *The Fractal Geometry of Nature*. Freeman, San Francisco, California, 1982.

[60] J. C. Mankins, editor. *Technology Planning Workshop for the Mars Rover*, Pasadena, California, 1987. Jet Propulsion Laboratory.

[61] D. Manko. *A General Model of Legged Locomotion on Natural Terrain*. Ph.D. thesis, Dept. of Civil Engineering, Carnegie Mellon University, April 1990.

[62] Martin Marietta Inc. Mars rover/sample return (MRSR): Mobility and surface rendezvous studies, October 1988. Work performed under JPL contract 958073.

[63] R. McGhee. Vehicular legged locomotion. In G. N. Saridis, editor, *Advances in Automation and Robotics*. Jai Press, Greenwich, Connecticut, 1985.

[64] Thomas A. McMahon and John Tyler Bonner. *On Size and Life*. Scientific American Books, Inc., New York, NY, 1983.

[65] David J. Montana. Strongly typed genetic programming. Technical Report 7866, Bolt Beranek and Newman, Inc., Cambridge, MA, March 1994.

[66] H.J. Moore, R.J. Pike and G.E. Ulrich. *Lunar Terrain and Traverse Data for Lunar Roving Vehicle Design Study*. US Geological Survey, March 1969.

[67] Scott Mullins and James R. Rinderle. Grammatical approaches to engineering design, part 1: An introduction and commentary. *Research in Engineering Design*, 2:121–135, 1991.

[68] Peter V. Nagy. *An Investigation of Walker/Terrain Interaction*. Ph.D. thesis, Carnegie Mellon University, August 1991.

[69] Tatsua Nakamura. 3d dynamic simulator for walking robots. *Japan-USA FA Symposium*, 1990.

[70] Thang Nyugen and Thomas Huang. Evolvable 3d modeling for model-based object recognition systems. In *Advances in Genetic Programming*, chapter 22, pages 459–475. MIT Press, 1994.

[71] Jim R. Oliver. Discovering individual decision rules: an application of genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, 1993.

[72] Takeo Oomichi and Ibe Tomoyosi. Development of vehicles with legs and wheels. *Advanced Robotics*, 1(4):343–356, 1986.

[73] G. Pahl and W. Beitz. *Engineering Design*. Springer-Verlag, Berlin, 1984.

[74] Y. J. Pedley. *Scale Effects on Animal Locomotion*. Academic Press, London, 1977.

[75] Heinz-Otto Pietgen and Dietmar Saupe, eds. *The Science of Fractal Images*. Springer-Verlag, 1988.

[76] D.T. Pham and Y. Yang. A genetic algorithm based preliminary design system. *Proceedings of the Mechanical Engineering Institute, Part D (Journal of Automobile Engineering)*, 207:127–133, 1993.

[77] S. Price, W. Chun, M. Hammond, and A Hubert. Wheeled planetary rover testbed. In *SPIE Mobile Robots V*, volume 1388, 1990.

[78] M. Raibert. Introduction to special issue on legged locomotion. *Intl. Journal of Robotics Research*, 9(2), April 1990.

[79] J. R. Randolph, editor. *Mars Rover 1996 Mission Concept*, Pasadena, California, 1986. Jet Propulsion Laboratory. Available as Technical Report D-3922.

[80] Gregory J. E. Rawlins, editor. *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, 1991.

[81] Giridhar Reddy and Jonathan Cagan. An improved shape annealing method for truss topology generation. Accepted in *ASME Journal of Mechanical Design*, 1994.

[82] Craig W. Reynolds. Evolution of Corridor Following Behavior in a Noisy World. To appear in *Simulation of Adaptive Behavior*, 1994.

[83] James R. Rinderle. Grammatical approaches to engineering design, part II: Melding configuration and parametric design using attribute grammars. *Research in Engineering Design*, 2:137–146, 1991.

[84] Rojas-Gusmán, Carlos. GALGO: A Genetic ALGOrithm Decision Support Tools for Complex Uncertain Systems Modeled with Bayesian Relief Networks. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, 368-375, Washington, DC, July 1993.

[85] Gerald P. Roston and Kevin Dowling. *Daedalus: A Walking Robot for Autonomous Planetary Exploration*. In High Frontiers XI, Princeton, NJ, May 1993.

[86] A. D. Ryan and K. H. Hunt. Adjustable straight line linkages - possible leg-vehicle applications. *Journal of Mechanisms, Transmissions and Automation in Design*, 107:256–261, June 1985.

[87] George N. Sandor and Arthur G. Erdman. *Advanced Mechanism Design: Analysis and Synthesis, Volume 2*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[88] J. David Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1993.

[89] Hagen Schempf. Houdini: In-tank mobile cleanup robot. In *Robotics for Challenging Environments*, Albuquerque, NM, 1994.

[90] L.C. Schmidt and J. Cagan. Recursive annealing: A computational model for machine design. Accepted in *Research in Engineering Design*, 1994.

[91] Joseph E. Shigley. *Mechanical Engineering Design*. McGraw Hill, 1977.

[92] Joseph E. Shigley and John J. Uicker. *Theory of Machines and Mechanisms*. McGraw Hill Book Company, 1980.

[93] Shin-Min Song and Kenneth J. Waldron. Geometric design of a walking machine for optimal mobility. *Journal of Mechanisms, Transmissions, and Automation in Design*, 109:21–28, March 1987.

[94] Sims, Karl. Evolving Virtual Creatures. In *SIGGRAPH '94 Proceedings*, 15-22, July 1994.

[95] Sims, Karl. Evolving 3D Morphology and Behavior by Competition. In *Artificial Life IV Proceedings*, 28-39, MIT Press, 1994.

[96] S. Song and K. Waldron. *Machines that Walk: The Adaptive Suspension Vehicle*. MIT Press, Cambridge, Massachusetts, 1989.

[97] A. Spiessbach. Issues and options for a mars rover. In *SPIE Vol. 852 Mobile Robots II*, pages 164–171, Cambridge, Massachusetts, November 1987. Society of Photo-Optical Instrumentation Engineers.

[98] G. Stiny. Introduction to Shape and Shape Grammars. *Environment and Planning B*. 7:343-351, 1980.

[99] R. H. Sturges, K. O'Shaughnessy and R. G. Reed. A systematic approach to conceptual design based on function logic. International Journal of Concurrent Engineering, 1(2):93-106, 1992.

[100] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical Report CMU-RI-TR-93-20, The Robotics Institute, Carnegie Mellon University, 1993.

[101] D. Todd. *Walking Machines: An Introduction to Legged Robots*. Chapman and Hall, New York, 1985.

[102] David G. Ullman, Thomas G. Dietterich, and Larry A. Stauffer. A model of the mechanical design process based on empirical data. *AI EDAM*, 2(1):33–52, 1988.

[103] H. Voogd. Multicriteria evaluation: measures, manipulation, and meaning - a reply. *Environment and Planning B*, 15(1):65–72, 1988.

[104] James R. Wertz and Wiley J. Larson, editors. *Space Mission Design and Analysis*. Space Technology Library. Kluwer Academic Publishers, 1991.

[105] D. Wettergreen and C. Thorpe. Gait Generation for Legged Robots. *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, July 1992.

[106] L. Darrell Whitley, editor. *Foundations of Genetic Algorithms 2*. Morgan Kaufmann Publishers, 1993.

[107] D. D. Wright and R. E. Watson. Comparison of mobility system concepts for a mars rover. In *SPIE Vol. 852 Mobile Robots II*, pages 180–187, Cambridge, Massachusetts, November 1987. Society of Photo-Optical Instrumentation Engineers.

[108] R. N. Yong. Some further problems in the design of wheels and tracks. *Journal of Terramechanics*, 13(2):63–73, 1976.