Evolutionary Techniques Applied to Mask-layout Synthesis in Micro-Mechanical-Electronic Systems (MEMS)

Thesis by
Hui Li

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1999

(Defended May 17, 1999)

Acknowledgements

First and foremost, I would like to thank my advisor Erik. His guidance led me to initiate this thesis work. His grateful patience, trust and encouragement sustained my confidence throughout this thesis work.

A special thanks to Min for her love and support on every piece of my life, and for her inspirations to me. Undoubtedly, without her companion, this work would not be completed.

To my mother, father and sister.

Evolutionary Techniques Applied to Mask-layout Synthesis in Micro-Mechanical-Electronic Systems (MEMS)

by

Hui Li

In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Abstract

This thesis reports an automatic method for synthesizing MEMS mask-layouts. This method incorporates a forward simulation of fabrication into a general evolutionary algorithm loop. An initial random population of mask-layouts is generated. The fabrication of each layout is simulated through a digital process simulator to produce a 3D fabricated shape, which is compared to a user-specified desired shape. Each evolutionary loop governs the stochastic searching behavior such that the mask-layouts whose simulated shapes are closer to the desired shape are more likely to survive. More importantly, the "better" masks are more likely to be evolved among those survived mask-layouts for the next loop. Through such evolutionary iterations, a near global "optimum" mask-layout is likely to be found. By using this evolutionary approach, we are able to take use of existing simulations of fabrication processes to achieve those mask-layout synthesis where reversing a fabrication process simulation (so that a 2D mask-layout might be produced) appears not to be possible. The general evolutionary loop mainly consists of a mask genetics module, an evolutionary technique module and a MEMS simulation module. The mask genetics module provides heuristic genetic operations on mask-layouts, which includes mask coding scheme, random mask generation, random crossovers and mutations. The evolutionary technique module contains stochastic selection schemes and genetic operation schemes to control the searching convergence. The MEMS simulation module is the user input module, which requires a MEMS fabrication simulation and user-specified desired shape. A test loop is constructed

for the bulk wet etching mask synthesis by incorporating a 3D wet etching simulation. The obtained results demonstrate the feasibility of this approach to mask-layout synthesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 MEMS CAD

In this thesis, a MEMS (Micro Electro Mechanical System) mask-layout synthesis methodology and its implementation will be presented and examined.

MEMS technology emerged decades ago to create small feature size (typically micron-scaled) mechanical devices integrated with micro-electronics on substrates (typically silicon wafers) [62]. As MEMS technology has grown, more and more complex devices have been made that provide unique functional features most of which were unimaginable with conventional macro device techniques [48, 62]. However, the difficulty of developing a new micro device remains high. This difficulty is mainly caused by the strong dependency on human experience in MEMS design and fabrication, and such experience is usually obtained through many repeated trials and errors.

In MEMS, the formation of device shapes is primarily controlled by the initial pattern layout through lithography followed by material removal techniques such as wet or dry etching. Because many of the fabrication processes produce shapes that are distortions of the mask-layout pattern, a pre-distortion of the mask-layout is required to obtain a desired shape. The amount of pre-distortion is highly process-dependent, with some processes producing relatively little distortion, and others (such as anisotropic wet etching) producing significant and geometrically complex distortions. Additionally, process variations, both globally (*e.g.*, variations in etch rates due to ageing of the chemicals, and from chemical lot-to-lot), and locally (*e.g.*, non-uniform stirring, etch depletion in concave corners), add

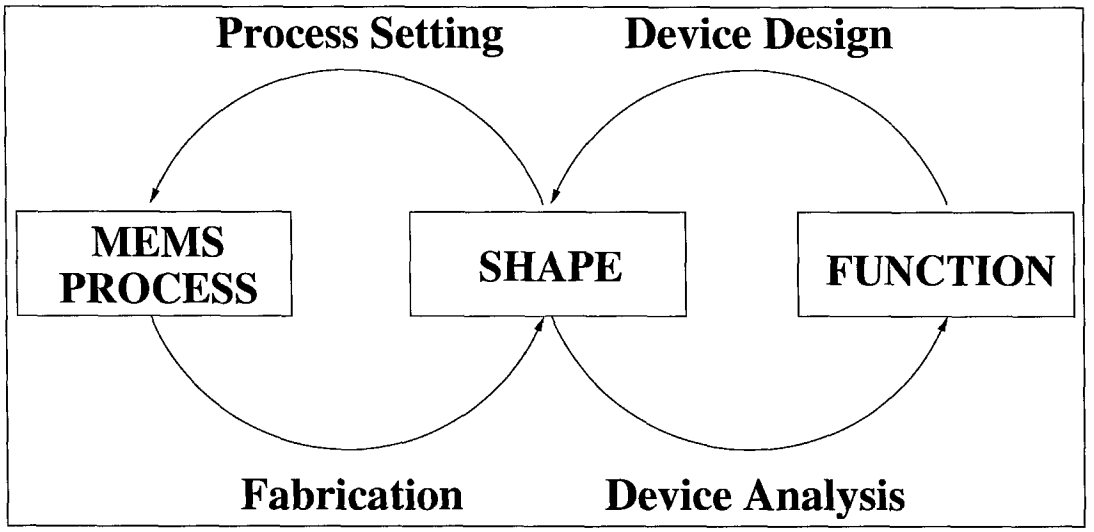


Figure 1.1: MEMS Design Process

additional complexity to developing an appropriately pre-distorted mask-layout pattern to produce a desired shape. Additionally, many current devices consist of several components which further increase the complexity level. To ensure their desired functional performance, high accuracy of each component as well as on the assembly precision between multiple components are required, which further requires mask-layout patterns that are robust to process variations. In this challenging design environment, a considerable level of experience, coupled with many repeated experiments is required. Due to the dependence on trial and error, the design of new micro-devices and systems is commonly slow and expensive.

Figure 1.1 illustrates the same problem in a more formal and systematic way, and gives a clearer picture of how it can be attacked. The figure shows the full spectrum of MEMS design activities. Designers start with the desired functional features. The device design step is taken to create a particular MEMS device. The designed outcome has to be verified through the performance testing based on the prototyped device. Such device verification process is referred to as the device-prototyping process. Iterations of the device design and the device-prototyping process usually move towards an optimum design solution. The entire device-prototyping process starts with the selection of a fabrication process and the settings of process inputs. The typical process inputs include materials, mask-layout and process parameters such as temperature and process duration. Then the actual fabrication

proceeds to produce the device prototype. Any fabrication failure will lead to the iteration between the steps of process parameter setting and device fabrication. The iteration involves the readjustment of the mask-layout pattern along with process parameters. The success of fabrication here is considered whenever the geometry mismatch between the fabricated device and the designed device meets the required tolerance. For successfully produced devices, the device analysis step is needed to further check the device performance against the desired functionalities. In case of functional failure, the device structural analysis during device operation is carried out to determine the failure cause. Depending on the analysis result, designers may have to either restart the device-prototyping process to readjust another settings of the fabrication process or even go back to another design process with a refined design solution. In either case, the iteration will be carried until a successful prototyped device realizes the desired functional features. So the whole device design is an iterative process between device design step and device-prototyping process which further is another iteration among the steps of process parameter setting, fabrication and device analysis.

Unfortunately, such iterations rely on the experience of the designer, and are often repeated in a device design process. This leads to the fabrictation of many device-prototypes, as illustrated above. There are two obvious strategies to attack this problem. One is to reduce and even eliminate the iterations through the continuing development of MEMS fabrication technology (requiring smaller and less geometrically complex pre-distortion of the mask-layout pattern); the other is to use structured design methods to automate (or semi-automate) the steps of the MEMS design process. Both strategies have driven the development of MEMS technology. This thesis only addresses structured design methods for MEMS, and focuses on fabrication by wet-etching, because it requires the largest degree of mask-layout pattern pre-distortion, and because it introduces the most geometrically complex distortions.

Figure 1.2 shows several different areas where computational methods can assist the design of MEMS. One method is the use of fabrication process simulation tools to permit the creation of virtual prototypes. Finite element analysis can be utilized to automate the step of stress and deformation analysis, and can include analysis of electrical forces and

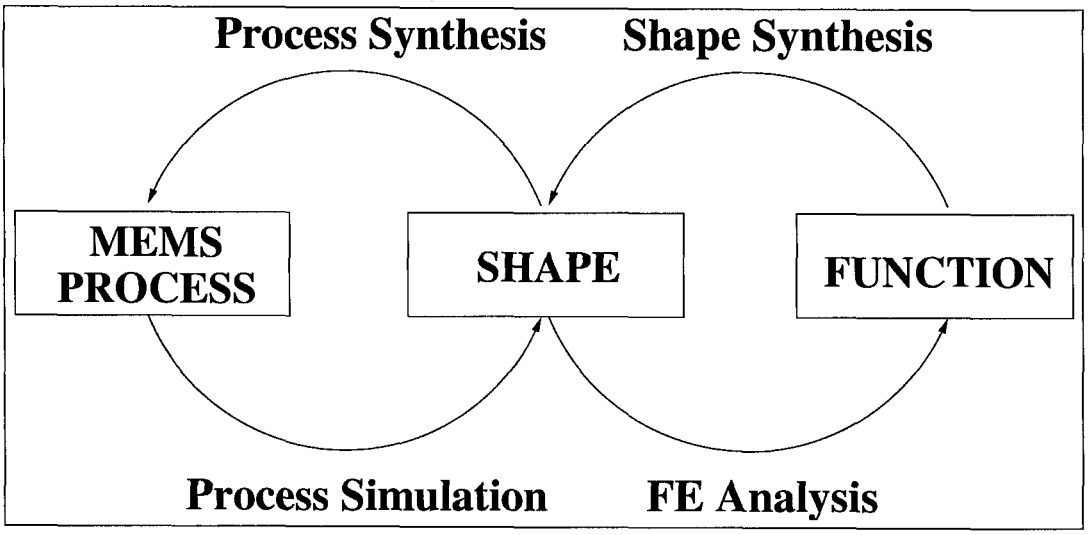


Figure 1.2: Computational Methods in MEMS Design

charge distribution as well as fluid flow and pressures. Combining the two, the cost and time of device-prototyping can be greatly decreased through the use of simulation on computers instead of physical device prototyping, with its many trials and errors. Naturally, a final device prototype is required, to evaluate the full (rather than the modelled) fabrication and performance of the device. The goal, of course, is to avoid wasteful and slow experiments by carrying out much cheaper computer work to get the fabrication "right" the first time.

Recently, various process synthesis tools have been developed to automatically synthesize the fabrication processes for a given device design [29]. By integrating such process synthesis tools with process simulation and device analysis tools, the entire device-prototyping process can be automated. In this way, the burdens of MEMS prototyping are greatly reduced, allowing designers to fully focus on the device functional design activities.

Even though currently there are no existing software tools directly targeting it, desires have been raised to develop future shape synthesis tools which will automatically synthesize the optimum design solution for the desired functional features. Such a design solution will mainly include the shape of functional elements and the interconnections among the elements. Utilizing this approach, including the automation of both the device design step and device prototyping process, designers can be completely released from the iterations

of design process, and only need to focus on the right device specifications within design guidelines.

In summary, Figure 1.2 essentially illustrates the road-map for the development of structured methods for MEMS design. More details on the process simulation tools and device analysis tools will be outlined below. The work on the synthesis tools is recently motivated by the success of VLSI CAD tools which will be elaborated more in the next section. The expectation on the shape synthesis tools will be recapped as the future work in this thesis.

The development of process simulation tools, as can be expected, is directly influenced by the technology of MEMS fabrication processes. Currently, there are mainly two types of MEMS fabrication techniques: surface micro-machining and bulk micro-machining. Surface micro-machining is used to fabricate single or multiple thin layers on the wafer. Each layer is patterned with high-resolution 2D geometry. With the use of one or more sacrificial layers, layer contact may not be statically bonded. Most of the techniques can be directly borrowed from integrated circuit processes which have been well-developed through the past decades. Therefore, most process simulation tools of surface micro-machining are obtained through the slight modification of existing VLSI simulators. These simulators cover standard semiconductor processes such as oxidation, diffusion, thin-film deposition, and plasma etching. An example is SAMPLE for simulation of projection lithography deposition, and etching [58, 59]. Bulk micro-machining, on the other hand, as a unique MEMS machining process, is used to fabricate 3D mechanical structures on the silicon substrate with the use of wet (chemical) or dry (plasma) etching technique combined with masking films or etch-resistant layers. Major effort has been devoted to developing simulations for bulk micro-machining, especially the wet etching process due to its capability of producing high aspect ratio structures. Up to now, several three-dimensional simulation programs for anisotropic etching profiles have been created such as ASEP by Buser and Rooij [11], the Slowness method by Foote and Sequin [69], the stochastic CA (cellular automata) algorithms by Hubbard and Antonsson [39], Than and Buttgenbach [75], SEGS by Hubbard and Antonsson [38], and recently, ACES by Zhu and Liu [81]. All these programs predict the 3D geometry change of the underlying etching structure for different etching time steps based on the input of a mask-layout and etch rate data. Different approaches were actually

used to model the entire etching process. The models of ASEP and the Slowness method are purely based on geometric analysis while Hubbard and Than's stochastic CA programs use cellular automata model. The geometric analysis is fast and accurate but only limited to basic shapes. The cellular automata approach is slow but is able to deal with complex shapes to certain accuracy. Recently, the cellular automata approach is improved by ACES with faster and more accurate results. SEGS uses edge segmentation approach to compromise the overall performance from the geometric analysis and cellular automata. In short, facing the currently available modeling techniques on the wet etching process, there are still existing gaps for the simulators to achieve the true automation. The gaps comes from the challenges of how to model the nonlinear factors in the etching process such as the change of etch rates caused by the depletion of chemical species, the bondage between the mask and the substrate to show the accurate undercut process and the etching stop condition, etc.

The framework of all the device analysis tools is essentially borrowed from the techniques of conventional mechanical CAD. Starting with a process-simulated 3D solid model along with specified boundary conditions, the discretization process is followed to mesh the solid model. Finally, a PDE solver is used to generate the performance evaluation results. Usually, the results are the strict data analysis on some function-critical properties such as stress and deformation distributions, thermal effects and exterior forces due to fields or fluids. In addition, most of these steps can be implemented through the existing commercial mechanical CAD tools such as ABAQUS for structural and thermal finite-element simulation and FASTCAP for electrostatic boundary-element simulation. The whole process, however, is rather computational intensive. Therefore, major effort is being devoted to developing fast and accurate algorithms to improve the efficiency of the process. Senturia *et al.*, addressed three computational challenges which are surface exterior force computation, coupled-domain simulation on interactive performance between devices, and nonlinear macro-modeling to rapidly map selected dynamical variables to the overall performance of a system-embedded device [67]. Even though the improvement is gradually made to effectively attack these challenges (e.g., Ljung *et al.*, shows a faster computation by using a boundary element method discretization and solver instead of traditional finite element analysis [46]), the gap still exists to achieve truly automated interactive device analysis.

Several MEMS CAD programs have been under development to integrate both currently available process simulation tools and device analysis tools. The examples are MEMCAD by MIT [68], IntelliCAD by IntelliSense [50] and CAEMEMS by the University of Michigan [14]. These programs typically also have a geometric solid modeler at front for users to construct 3D structures and also maintain material and process databases for users to specify the materials and fabrication processes. With the use of such multi-functional tools, designers can have much more effective way to carry "what if" experiments during the device prototyping process. However, as mentioned above, even under these integrated design environment, designers still need to bear many "what if" tries in order to identify the optimum design solution and the appropriate process settings. Therefore, the development of synthesis tools to achieve a higher level of computer-aided releasing becomes rather important. Especially, through years of industrial proof, it recently becomes well recognized that VLSI CAD synthesis tools have given great contribution to the success of VLSI technology, which provides a realistic image of the potential impact that MEMS synthesis tools can give to the future development of MEMS technology.

1.2 VLSI CAD

Prior to Mead and Conway's effort towards the computer-aided automation of the integrated VLSI system designs [51], the designers of VLSI systems needed detailed processing specifications of particular mask and fabrication firms in order to ensure the validity and performance of the design outcome. Therefore, the large-scaled design work essentially remained in the domain of highly trained and experienced specialists. Through the use of structured design methodology, Mead and Conway successfully enforced the clean separation between the maskmaking-to-fabrication sequence and the steps of function, design and layout. In this methodology, the circuit design is abstracted with hierarchic levels of functional design, physical design and fabrication. The key element to achieve such abstraction is a set of design synthesis rules automatically used to guide and synthesize the valid layouts according to interactive geometry input from the designers. Under such design tools, designers can start with some informal design sketches and notes with some impor-

tant cells and end with computer-generated pattern generator files according to the verified mask-layouts. Therefore, designers are fully released from the fabrication knowledge.

Recently, advances in the development of surface micro-machining has largely benefitted from the VLSI fabrication technology, which stimulated the open question of whether and how we can achieve the similar benefits from the success of VLSI CAD technology for the development of MEMS synthesis tools [2]. As far as surface micro-machining concerned, the structure of process flow is very similar to that of VLSI. Such similarity creates opportunities for MEMS structured design tools to borrow much of the VLSI framework, including the separated abstraction levels and the hierarchical system modeling based on primitive functional elements. The recent development of schematic synthesis design tools for surface micro-machining by Fedder *et al.*, has demonstrated such benefits [56]. Nevertheless, MEMS systems, in general, do not have the same modularity and topological simplicity as electronic circuits, and hence, pose a new challenge to both MEMS design and process synthesis. MEMS systems typically consist of a range of unstructured non-modular elements whose interconnections are complex especially with the inclusion of kinematics. In addition, a function no longer has a simple correlation with any physical form, but rather critically depends on the concrete geometries, material properties of one or multiple interacting elements. All these aspects cause great difficulty to define a set of design synthesis rules for the system structures and behaviors. The process synthesis is not simple either. Especially for bulk micro-machining, there is no simple mapping between mask layouts and processed structures. Moreover, the correct functional behaviors of MEMS devices relies on constraint-satisfaction on both the geometrical dimensions and the physical properties of materials. Such demands lead to tight tolerances which must be conservatively incorporated into the process design rules. In summary, even though there is some common fabrication nature between MEMS and VLSI, and thus the chance for the sharing of CAD technology, the difference in the functional and structural domains demand much more effort to realize a VLSI-like structured design methodology.

1.3 Thesis Contributions and Chapter Overview

1.3.1 Contributions

My research area has been focused on the development and implementation of a methodology for MEMS mask-layout synthesis. The contributions of my research work can be summarized in the following aspects: formalized a general methodology of using the techniques of evolutionary algorithm to achieve the MEMS mask-layout synthesis; developed an algorithmic architecture for the implementation of the synthesis methodology which is open to any forward fabrication process simulator; obtained automated mask-layout synthesis results for a wet-etching simulator.

The development of MEMS synthesis tools is an emerging area. Several methodologies have been proposed and are still under development whose details will be covered in the next Chapter. All these methodologies are based upon specific features of either surface micro-machining or bulk micro-machining. Therefore, they can only be applied to a particular set of design domain. A synthesis methodology which can be applied to MEMS design in general is badly needed. Facing the variety of MEMS functional design, the methodology should rather focus on the design of high-level synthesis framework such that once any automated low-level design activity is embedded into the framework, the corresponding synthesis is automatically provided. The key element here is that the synthesis framework has to be applicable to a broad set of MEMS design processes. With such an approach, one candidate methodology is proposed in this thesis to use an evolutionary algorithm to realize mask-layout synthesis applicable to any forward process simulation. Evolutionary algorithms are a global stochastic optimization technique. It is non-problem specific which means it can be applied to virtually any problem if only its objective function measurements are available. In particular, due to its unique method of searching solution space, evolutionary algorithms have a strong ability to optimize information flow in complex domains. These two features produce a unique advantage of using evolutionary algorithms as the driving engine to optimize some complex solutions for process synthesis such as mask-layouts. Furthermore, it is later shown that such approach can be easily extended to cover more general synthesis steps such as the direct synthesis from function to process, illustrated in Figure 1.2.

An object-oriented software architecture has been created to implement such an evolu-

tionary algorithm based framework. With the use of object abstraction, a set of base level objects is constructed to represent all the critical components of the evolutionary algorithm. Each base level object has a set of abstract interfaces corresponding to each implementation component of the evolutionary algorithm. In this way, the framework of evolutionary algorithms can be established independent of any application-specific modules. In addition, the development of various evolutionary techniques on top of the established framework becomes completely separated from the effort towards the embedding of various application components into the framework. Therefore, a high level of software modularity is obtained.

A bulk micro-machining wet etching simulator SEGS [38] has been plugged into the mask-layout synthesis application to demonstrate a synthesis outcome using the evolutionary algorithm. The high complexity level of wet etching mask-layout synthesis comes from the fact that there is no simple geometrical correlation between masks and the evolved 3D structures. Successful results have been obtained for several challenging tests. Of course, the full verification of such synthesis methodology has to be based on further tests on various other applications. Nevertheless, the demonstrated success over such complexity level of synthesis gives enough justifications to continue the development of such methodology.

1.3.2 Overview

The rest of the thesis consists of three parts. The first part is background materials which spans Chapter 2, 3, 4 and 5. Chapter 2 will overview the current development on the MEMS synthesis which covers both surface micro-machining and bulk micro-machining. Chapter 3 introduces the evolutionary optimization techniques. A full spectrum of the stochastic searching techniques will be examined with the main focus on genetic algorithms. Since the constructed mask-layout synthesis algorithm is tested through a bulk wet etching simulator, a brief background on bulk wet etching process is provided in Chapter 4 with focus on the chosen simulator (SEGS). Chapter 5 will cover the topics of shape matching algorithms mainly including the matching between two polygon shapes. This technique will be mainly used to test whether the shape produced under any simulated process close enough to a specified desired shape, which is a critical component to achieve mask-layout synthesis.

Chapter 6 examines the design and implementation of the mask-layout synthesis method-

ology based on evolutionary techniques. Chapter 7 provides the test of the mask-layout synthesis for a bulk wet etching simulator. The presented test results are focused on evolving the mask-layouts with compensation features.

Chapter 8 is a conclusion with a summary and a discussion on possible future work.

Chapter 2

MEMS Synthesis

2.1 Overview

The goal of MEMS synthesis s to automatically generate optimum solutions for both device design and fabrication process. Especially, as mentioned previously, process synthesis is the key element to achieve a rapid prototyping environment for MEMS design, which allows designers to fully focus on the design of the function of the device and thus effectively help increasing the complexity level of MEMS devices. With the effort towards the future VLSI-like standardization of MEMS fabrication processes, it is very promising to eventually achieve the goal of process synthesis. Due to the involvement of mechanical systems whose modularity and topology are rather complex, the synthesis of device design or the shape synthesis, as illustrated in Figure 1.2, is essentially limited to specific domains. Nevertheless, any effort towards such synthesis will increase the overall reuse level of design knowledge.

In general, a formal method for MEMS synthesis that is robust and accurate to the whole MEMS design domain is still under investigation. Several methodologies based on the specific features of surface micromachining or bulk micromachining have been proposed. The synthesis of surface micromachining has been a main focus due to the similarity between the surface micromachining and VLSI technology which creates opportunities for directly borrowing the structured design methodology from VLSI.

2.2 Surface Micromachining Synthesis

There are essentially two major research efforts involved with the synthesis of MEMS surface micromachining. One is the process synthesizer MISTIC developed at the University of Michigan; the other is the device design synthesis tool based on VLSI-like schematic representation of MEMS systems which is being created at Carnegie Mellon University.

Gogoi *et al.* [29], introduced a method which automatically generates fabrication sequences for surface micromachined structures starting from a two-dimensional geometrical description. This method essentially translates the device geometry into layers and a mathematical representation of layer order. By using topological sorting techniques, all possible process sequences can be extracted from the layer order in terms of fundamental processing steps like deposition, lithography and etching. In general, since the fabrication sequence is not unique, an optimal sequence is determined from the candidate set by using a cost function based upon a database of materials and processes. The synthesis program called MISTIC has been implemented which includes both the sequencing algorithm and optimization. A complete optimal fabrication sequence is synthesized with the output in a human-readable format.

Mukherjee and Fedder proposed a hierarchically structured design approach to assist an efficient design of complex surface MEMS devices [56]. The method is compatible with standard IC design. A graphical-based schematic and structural view are used to represent MEMS as a set of interconnected lumped-parameter elements. Once the complex device is decomposed into components, the simulation at the component level can be carried through a simulation interface provided by the schematic representation. Furthermore, MEMS components can be synthesized with common topologies so as to provide the system designer with rapid, optimized component layout and associated macro-models. As an example of the application, a synthesis module is developed for the popular folded-flexure micromechanical resonator topology. The algorithm minimizes a combination of total layout area and voltage applied to the electro-mechanical actuators. The effectiveness has been shown through the synthesis effort towards the design limits of behavioral parameters such as resonant frequency for a fixed process technology.

2.3 Bulk-micromachining Synthesis

Synthesis in bulk-micromachining has been focused on the generation of mask-layouts used in bulk wet etching processes. Wet etching is a widely used fabrication technique in micromachining. The key process input is the mask-layout, which is the major element in determining the final etched shape. Unfortunately, due to the geometric complexity of the etching, there is no simple mapping between a mask-layout and the corresponding etched shape. Thus, there is a critical need to automate the design of the mask-layout for any desired etched shape. Long *et al.*, [47] proposed a method to synthesize the mask-layout geometry through a direct inverse of a forward etch simulation. An inverse mechanism was developed earlier based on a forward simulator called slowness method [69]. During the inverse process, the desired etched shape is decomposed into planes and vertices. The etch rate of the planes is determined by the etch rate diagram. Each vertex is categorized according to its convexity and whether or not there new planes will appear during the etch. From there, a local mask shape that will produce the vertex corner is obtained. The overall mask-layout is formed after considering the global interactions between all the local mask shapes.

This thesis presents a new approach to realize the mask-layout synthesis by incorporating a forward simulation of fabrication into a general iterative loop as shown in Figure 2.1. In this approach, the user can specify a desired 3D shape. The loop starts with initial candidate mask-layouts. The fabrication of each layout is simulated through a digital process simulator to produce a 3D fabricated shape, which is compared to the desired shape. If the best comparison result is not close enough, the initial mask-layouts are refined and the process is repeated. The iteration stops either when the simulated shape is close enough to the desired shape, or the specified maximum searching effort (such as the total number of iterations) has been reached. Evolutionary techniques have been developed as part of this research to refine the, tried mask-layouts during each iteration. An evolutionary algorithm has been constructed and tested to use these evolutionary techniques to synthesize mask-layouts for a bulk wet etching simulator. The results obtained demonstrate the effectiveness of this approach. In this way, one is able to make use of existing simulations of fabrication processes to achieve mask-layout synthesis where reversing a fabrication process simula-

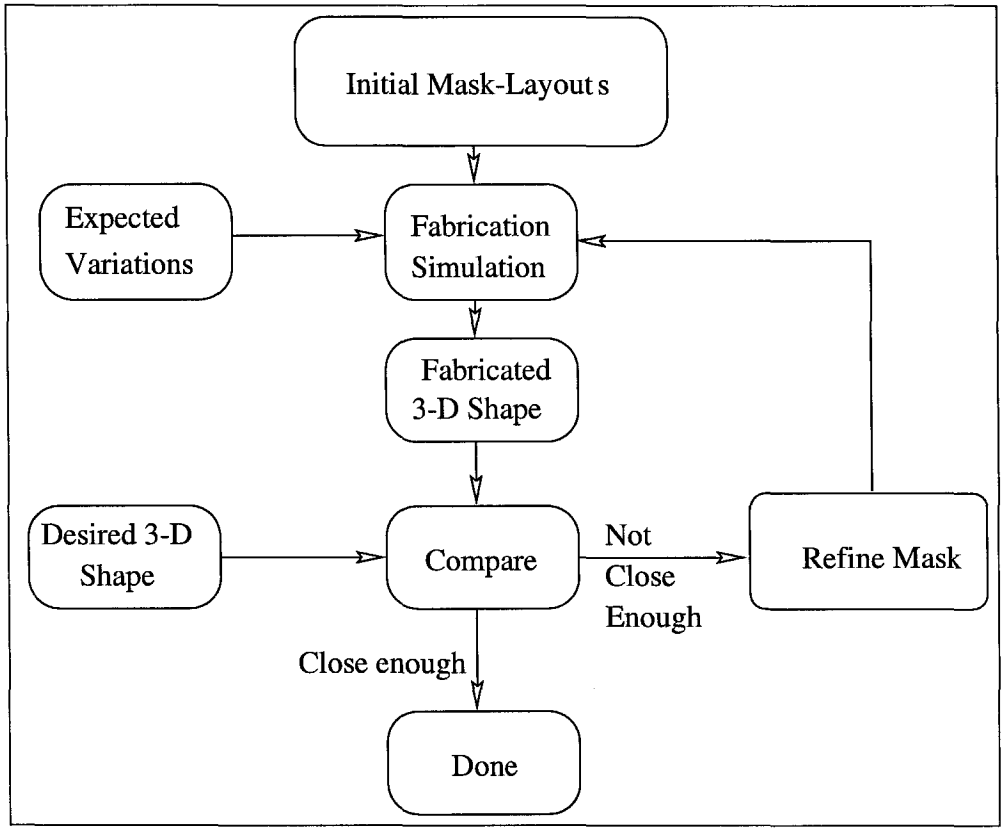


Figure 2.1: A General Approach to Realize Mask-layout Synthesis Through An Iterative Loop.

tion (so that a 2D mask-layout might be produced) appears not to be possible.

Chapter 3

Evolutionary Techniques

3.1 Introduction

An evolutionary algorithm is a global stochastic optimization technique based on the adaptive mechanics of natural genetics [35]. Figure 3.1 illustrates the linkage between the natural genetics and evolutionary algorithms through the comparison of the used terminologies between the two. The *strings* of artificial evolutionary systems are analogous to *chromosomes* in biological systems. In natural systems, one or more chromosomes combine to form a genetic package called a *genotype*. In artificial evolutionary systems, the total package of strings is called a structure. In natural systems, the organism formed by the interaction of the genotype with its environment is called *phenotype*. In artificial evolutionary systems, the structures decode to form a particular *solution form*. In natural systems, chromosomes are composed of *genes*, which may take on some number of values called *alleles*. In artificial evolutionary systems, strings are composed of *features*, which take on different *values*.

An evolutionary algorithm maintains a population of candidate solutions known as individuals. Typically, each individual is encoded into a string of characters or digits, through which the original solution space can be converted into an encoded space, in order to search for the global optimum. By analogy with genetics, the individuals in the original search space are referred as phenotypes; those in the encoded space are referred to genotypes. The initial population is generated randomly. During each iteration, within a population known as a generation, all individuals are evaluated to determine their performance values (called

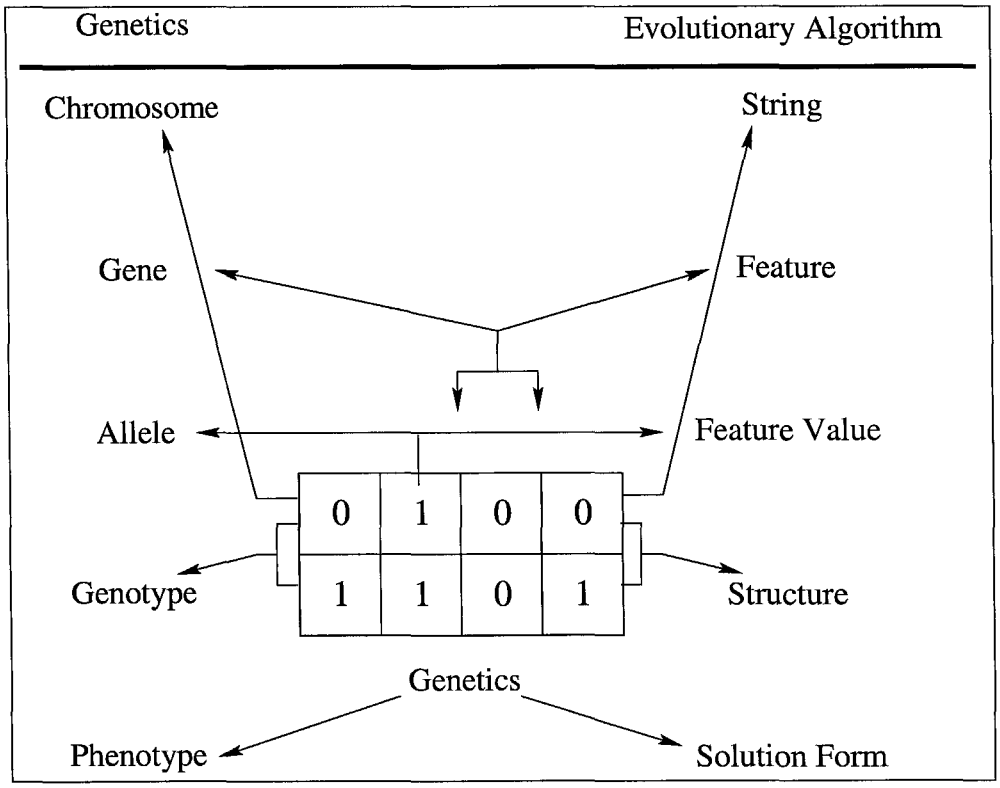


Figure 3.1: Comparison of Natural Genetics and Evolutionary Algorithm Terminology.

fitness values), and then go through genetic operations such as selection, crossover *etc.*, to form individuals of the next generation. The role of each genetic operation is rather different. Selection is used to choose individuals with better fitness performance. Crossover is applied to those selected individuals (parents) to evolve new ones (children) who are likely to preserve good features from their parents and more importantly grow new features that will outperform their parents. Mutation is applied to individuals to disrupt part of the existent features. The rationality of processing individuals in this way lies in the concept of survival of the fittest in the biological world. It is the fundamental belief that better-performing children are more likely to be evolved from well-performing parents because they have more chance to inherit, as well as combine, good gene features from their parents. Such iterative processing stops whenever the performance of an individual satisfies the desired goal, or the limit of the search effort is reached.

In general, evolutionary algorithms are a non-problem specific technique which can be

applied to virtually any problem if its objective function measurements are available. In particular, what makes evolutionary algorithms distinct among other stochastic optimization techniques is the involvement of a generation of individuals. Such individual pool participation provides efficient feature information used by genetic operations to maximize the chance to balance the search effort between exploration and exploitation, in the sense that every new exploration direction taken is based on the proper exploitation on currently preserved search regions. Such balance will further ensure a proper bias on the search region as well as increase the robustness to overcome the deception traps. Deceptions refer to any misleading the evolutionary process towards a local optimum. Evolutionary algorithms have been shown to successfully solve problems in various complex domains [30].

Finally, a clarification on the terminology should be made here. In this thesis, the major focus is on the evolutionary techniques. Any algorithm that uses those techniques will conduct an evolutionary search for a global optimum solution and therefore is treated as an evolutionary algorithm. Genetic algorithms are a major subset of evolutionary algorithms. Thus all the techniques described as genetic algorithms can be regarded as part of evolutionary techniques.

3.2 Coding Scheme

The task of any coding scheme is to encode a solution of the problem into a particular representational format which can be further manipulated by genetic operators. Using biological terminology, such representational format is referred to as a chromosome. Consider a problem with several parameters to be optimized, each chromosome will correspond to a vector of the parameters and it can be expressed using either binary bits, real numbers or integer numbers, which are the three widely used representation techniques. A chromosome with binary bits will encode each parameter as a bit string using either a standard binary coding or a Gray coding. The bit strings for the parameters are concatenated together to give a single bit string which represents the entire vector of parameters. In this case, each bit position corresponds to a gene of the chromosome and each bit value corresponds to an allele. Under standard binary coding, there is a direct mapping between a bit string and an integer

parameter, just as how the integers are stored in a computer. However, a value cutoff will be involved when a real parameter is mapped to a bit string. Let x_i denote the ith single parameter with its lower and upper bounds l_i and u_i respectively, then the binary coding of x_i using n bits will correspond to the binary code for the integer k for $0 \leq k < 2^n$ and k is such that x_i falls between $l_i + k\frac{u_i - l_i}{2^n}$ and $l_i + (k+1)\frac{u_i - l_i}{2^n}$. For example, if $l_i = 0$ and $u_i = 4$ and $n = 5$, then any real values between $\frac{3}{8}$ and $\frac{1}{2}$ would correspond to the same binary code 00011. Certainly such representation resolution can be increased with larger n. Gray coding is another way of coding parameters into binary bits which has the adjacency property that an increase of one step in the parameter value corresponds to a change of a single bit in the code. The conversion between binary coding and Gray coding is attached in Appendix A.

A chromosome with real numbers or integer numbers will simply encode each parameter with a real or integer number respectively which is then concatenated with others to represent the entire vector of parameters. In this case, a gene corresponds to a real or integer parameter and an allele corresponds to a real or integer value.

Holland's work [35] on the creation of genetic algorithm was based on binary strings. In addition, it was once believed that the principle of minimal alphabets should be used to guide the coding design in various problems, and obviously binary strings offer the possibly minimum number of alphabets [30]. A minor modification is the use of Gray code in the binary coding. Hollstien [36] investigated the use of genetic algorithm for optimizing functions of two variables, and claimed that a Gray code representation worked slightly better than the normal binary representation because of its adjacency property. Later on, as the canonical genetic algorithm is applied to more and more applications, it was realized that the bit string coding is not a natural coding scheme for some of the problems, especially the ones from industrial engineering world where real-valued parameters need to be optimized. So direct coding with real-value chromosomes raised some considerable interest [31, 79, 40]. There are essentially three primary motivations for using a real number coding scheme. First, real-coding of the genes eliminates the worry that there is adequate precision so that good values are representable in the search space. Whenever a parameter is binary coded, there is always the danger that one has not allowed enough precision

to represent parameter values that produce the best solution values. Second, the range of a parameter does not have to be a power of two. Third, genetic operations on real-coded genes have the ability to exploit the gradualness of functions of continuous variables. The gradualness here means that small changes in the variables correspond to small changes in the function [79]. The integer coding scheme is mainly used for sequencing problems where each solution is coded as a numerical sequence [26]. In case any of the above string-based representation pose difficult and sometimes unnatural answers to some optimization problems, some other coding techniques can be explored such as embedded lists for factory scheduling problems [53], variable element lists for semiconductor layout [53], matrix encodings for VLSI modules [13] and even LISP S-expressions [44].

The importance of choosing an appropriate coding scheme can not be overstated. A coding scheme determines the linkage between the solutions in phenotype space and the genetic structures in genotype space. An inappropriate coding scheme could lead to inefficient or even incorrect communication between the genetic information and the solution fitness of each individual and thus slow down or mislead the entire evolution. The ideal coding for a complex problem is a difficult choice. Often more than one coding strategy is possible for one problem. Thus some general rules are needed to guide such decision making. The literature indicates that codings should be chosen according to several properties among which the two most important ones are the following: the coding should embody the fundamental solution structures that are important for the problem type [30], and be amenable to a set of genetic operators that can propagate these solution structures from parent genotypes to the children genotypes [26]. The first rule requires an analysis of the solution features, followed by a choice of coding scheme such that the important solution features can be reflected through the chromosome in a simplest and most direct way. The second rule indicates that the chosen coding scheme should make it convenient for the genetic operators to recognize those important solution features and preserve them. Both these rules will be used to guide the later development of the coding scheme as part of the evolution techniques applied to mask-layout synthesis.

3.3 Initialization

As an initial start for an evolution process, a generation of individuals need to be randomly generated. During the initialization, the genotypes are usually the ones to be directly generated through a pseudorandom operator. Then, each genotype is decoded into phenotype which will give the fitness evaluation for each individual.

The ultimate goal of the initialization is to ensure a uniform distribution of individuals in phenotype space, or in other words, the sampled solution points uniformly cover the search space. To achieve this goal, all the genes in all the chromosome that belong to each genotype (one genotype can have multiple chromosomes) are randomly generated such that their allele values are uniformly distributed within each valid value regions. Under a proper coding scheme, the uniformity of genotype distribution should automatically lead to the uniformity of phenotype distribution. Here again, the coding scheme plays an important role. Especially, the coding scheme that has the property of one-to-one mapping between a genotype and a phenotype is strongly desirable. Any representational redundancy in genotypes can lead to inefficiency in obtaining a uniform phenotype distribution.

The generation mechanism on the allele values of all the genes depends on the coding scheme used. In a binary bit string coding, the bitstring-uniform procedure (BU) is traditionally used: it assigns value of 0 or 1 with probability 0.5 to every bit. This procedure is favored by its uniformity in the binary space. All points of n bit string have equal probability $1/2^n$ to be drawn. Moreover, the bit-wise diversity is maximal too: at every bit position, there will be in mean as many 0's than 1's. Recently, such procedure has been questioned by Kallel *et al.*, with different views of distribution uniformity [65]. As an improvement on these distributions, they proposed two new procedures to initialize the bit strings. One is the uniform covering procedure under which a density of 1's is uniformly selected in [0,1] for each bit string, and then choose each bit to be 1 with that specific probability; the other is the homogeneous block procedure under which a bit string is initialized to a default value (e.g., 0) and then gradually "add" homogeneous blocks of the other value (1).

Under a real coding scheme, the initialization can be rather different depending on the genotype structures under a particular application. In most applications, each gene represents an independent parameter and thus the initialization procedure can simply assign

a random value uniformly picked within its boundary range for each gene. Since each real-valued gene directly represents a solution value, the distribution uniformity should be assessed based on the sampled solutions in the phenotype space.

In an integer coding applied to a sequencing problem, usually the allele value of each gene represents an sequence order and the number of genes exactly matches the number of the interested orders. So it is the sequence of the allele values in each chromosome that needs a uniform distribution. One approach is to assign the genes with increasingly ordered values and then randomly shuffle those values for a specified number of times which is usually the number of genes in each chromosome [77].

In some situations, the domain specific application knowledge can be incorporated into the initialization so that a more effective initial sampling can be obtained. One example is to pre-seed the initial population with fit individuals, or lying in regions of the search space known to be of some interest [33]. A more general approach is that the initial population is built by taking the best of n randomly chosen individuals [49]. Another general approach is used by Eshelman [24] to re-initialize the population whenever a preconverged evolution is detected and during the re-initialization, a best currently evolved individual is maintained and the rest of individuals are introduced by randomly regenerate partial genes of the best one.

3.4 Crossover

3.4.1 Overview

In most applications, crossover is used as the vital force to evolve new individuals based on the ones in the current generation. Each time, a crossover is normally applied to a pair of parent individuals and produces one or two child individual(s). The goal of crossovers is to grow the genotype structures with better performing gene features which will ultimately lead to the global optimal solution. To do so, each crossover should be able to not only recognize and preserve the existing good-performing genes among the paired parents but also disrupt bad performing ones and replace them with other explored features so as to increase the chance to form even better overall features. Usually, the individuals being applied with

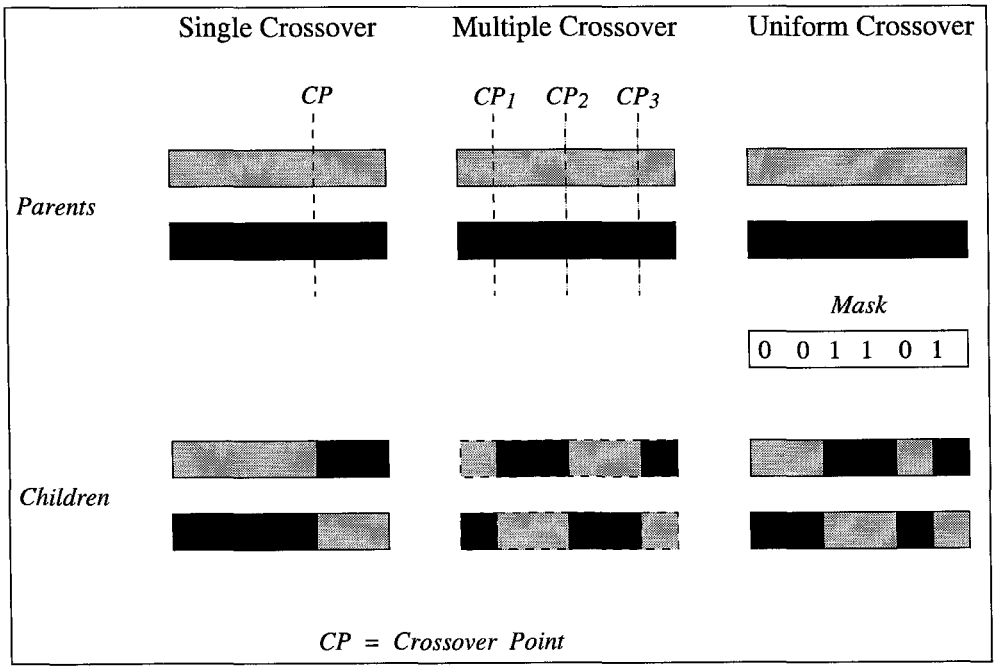


Figure 3.2: Several Binary Crossovers.

crossovers are relatively better performing ones selected from the fitness competition. The good gene features between any two paired parents is simply taken as their common or closely matched features. With a random processing, doing so will maximize the likelihood to recognize the well performing features, unless some deceptions occur which will be explained below. The rest of the dissimilar gene features are likely to experience more rigorous exploration with the replacement of rather different allele values. As will be seen, all the following crossovers have taken this approach to preserve the promising characteristics of the parents. However, the implementations vary to different coding schemes as well as applications.

3.4.2 Binary Crossover

Figure 3.2 shows several popular crossovers based on a bit string coding scheme. The first crossover is called single crossover or one-point crossover, during which a crossover point is randomly chosen among all the genes and then the bit values of the genes after the crossover point are exchanged between the two parents. As shown in the figure, CP is the crossover

point, all the bits right to the CP are exchanged between the parents to form the children. The second crossover is a generalized version of one-point crossover which is called multiple-point crossover. This crossover has more than one crossover points randomly chosen among all the genes. The bit values of the genes inbetween the regions formed by every other pairs of crossover points as well as the beginning and ending genes are exchanged.. As an example, a three-point crossover is shown in the figure with the crossover points CP_1, CP_2, CP_3. The bit values between CP_1 and CP_2 and those between CP_3 and the ending gene are exchanged. The third crossover is called uniform crossover which first generates a crossover mask with the same number of genes as the parents. For each gene in the crossover mask, if its allele value is 1, it means no allele value exchange for that gene between the parents; if its allele value is 0, it means the exchange. The figure shows a randomly generated crossover mask and the correspondent allele value exchange between the parents. There are also some other binary crossovers such as the reduce-surrogate crossover which guarantees that exactly half of the non-matching alleles are exchanged. In this way, the generated children will always have the maximum Hamming distance from their two parents, where the Hamming distance is defined as the number of genes having different allele values between any two binary strings. Note that all the crossovers are implemented through exchanging the bits between the parents. Therefore all the common bits in both parents will be automatically preserved in their offspring.

The preference of which crossover to use is debatable. Eshelman *et al.*, have conducted several experiments for various crossover operators [23]. A general comment is that each of these crossovers is particularly useful for some classes of problems and quite poor for others, except that one-point crossover is indicated as a "loser" experimentally. Although one-point crossover was initially inspired by biological processes and used in Holland's work [35], it has one major drawback that certain combinations of solution features can not be combined in some situations [53]. A multipoint crossover can be introduced to overcome this problem. As a result, the performance of generated offspring is greatly improved. Furthermore, DeJong [21] concluded that a two-point crossover seemed to be an optimal number for multipoint crossover. However later, this has been contradicted by Spears and DeJong [71] as two-point crossover could perform poorly in a situation where the population

has largely converged because of reduced crossover productivity. This low crossover productivity problem can be resolved by the proposal of reduce-surrogate crossover [9]. Since uniform crossover exchanges bits rather than segments, it can combine features regardless of their relative locations; however, it has strong power to disrupt some solution features. It is shown [73] that the former advantage may outweigh the later disadvantage, which makes uniform crossover a superior operator for some problems. A further example of this is the CHC, a nontraditional genetic algorithm, which uses uniform crossover combined with a conservative selection scheme to outperform traditional genetic algorithms [24].

3.4.3 Real Crossover

Under a real coding scheme, crossovers can be mainly classified into three groups: conventional operators, arithmetical operators and direction-based operators. The conventional operators are made by extending the operators for binary representation into the real coding case. The arithmetic operators are constructed by borrowing the concept of linear combination of vectors from the area of convex set theory. The direction-based operators are formed by introducing the approximate gradient direction or negative direction into genetic operators. Note that for real-valued genes, the common characteristics of parents lie in the inclusive value regions defined by all paired genes. All the three operators are constructed such that there is an intrinsic bias towards exploring those inclusive features and thus preserve the common parent features.

There are mainly two kinds of crossovers that belong to conventional operators. One is simple crossovers which include one-point crossover, multi-point crossover and uniform crossover. The implementations of all the three crossovers are exactly the same as those binary crossovers except now the exchanging materials are real element values instead of binary bit values. For details, see Spears and DeJong [72] and Syswerda [73]. The other kind is random crossovers which create children within a hyper-rectangle defined by the parent points. The basic one is given by Radcliffe [64], called flat crossover, which produces an offspring by uniformly picking a value for each gene from the range formed by the values of two corresponding parent genes. Eshelman and Schaffer presented a generalized crossover of Radcliffe's work [25], called blend crossover and denoted as BLX-α.

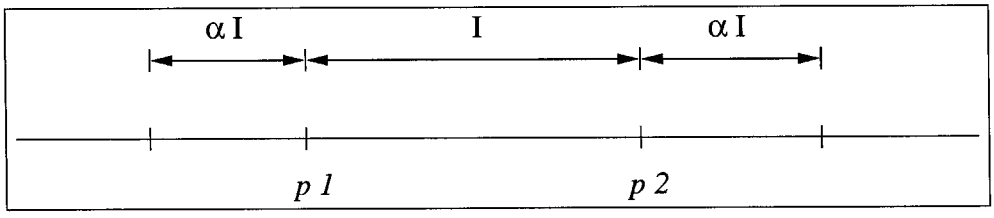


Figure 3.3: $BLX - \alpha$.

It uniformly picks values that lie between two points that contain the two parents, but may extend equally on either side determined by a user specified parameter α. As shown in Figure 3.3, BLX-α picks parameter values from points that lie on an interval that extends αI on either side of the interval I between the parents. Even though both blend crossover and float crossover exploit the parameter intervals determined by the parents, it is shown that blend crossover has more advantages. Because of the extended intervals, the crossover will have the chance to explore the values outside the parent intervals which means it can disrupt the inclusive features formed by the parents. Such disruptive power can be useful to break deceptive features carried by parents if needed and thus increase the robustness of the crossover. In particular, since the range of the extended intervals is linearly proportional to the parent interval, the provided disruptive power can be automatically synchronized with the convergence of the parent features as needed. To make it more clearly, at the initial stage of a typical evolution, the individuals are expected to be rather different which leads to large parent intervals and thus high disruptive power from the crossover. Meanwhile, any inclusive features formed by parents at this stage are rather suspicious even though both parents perform well. So these features are likely need to be disrupted in time to prevent any premature convergence. As the evolution goes on, individuals become converged which leads to small parent intervals and thus low disruptive power from the crossover. At this stage, any inclusive features formed between parents are more likely to be good features and need to be preserved for future exploitation. Through various tests, Eshelman and Schaffer [25] further demonstrated the best α value to be 0.5, and explained that such α value will lead to the probability that an offspring will lie outside its parents equal to the probability that it will lie between its parents and thus the convergent and divergent tendencies can be naturally balanced if without any selection pressure.

Arithmetical operators are defined as the combination of two vectors (chromosomes) as follows:

$$\begin{aligned} c_1 &= \lambda_1 p_1 + \lambda_2 p_2 \\ c_2 &= \lambda_1 p_2 + \lambda_2 p_1. \end{aligned} \tag{3.1}$$

where

$$\begin{aligned} c_1 &= \text{first child} \\ c_2 &= \text{second child} \\ p_1 &= \text{first parent} \\ p_2 &= \text{second parent} \\ \lambda_1 &= \text{first real multiplier} \\ \lambda_2 &= \text{second real multiplier.} \end{aligned}$$

According to the restriction on multipliers, three kinds of crossovers can be defined. When:

$$\lambda_1 + \lambda_2 = 1, \quad \lambda_1 > 0, \quad \lambda_2 > 0,$$

the weighted form (3.1) defines the convex crossover; if nonnegativity condition on the multipliers is dropped, it leads to the affine crossover; if the multipliers are simply required to be in real space, it yields the linear crossover. The convex crossover is the most commonly used one [53]. When restricting that $\lambda_1 = \lambda_2 = 0.5$, it yields a special case, which is usually called average crossover by Davis [20], or intermediate crossover by Schwefel [66]. The affine crossover was first proposed by Wright [79]. Let p_1 and p_2 be any two paired parents. During each crossover, three new points are generated, namely $\frac{1}{2}p_1 + \frac{1}{2}p_2$, $\frac{3}{2}p_1 - \frac{1}{2}p_2$ and $-\frac{1}{2}p_1 + \frac{3}{2}p_2$. The first point is the midpoint of p_1 and p_2, while the second and the third points lie on the line determined by p_1 and p_2. Then, the best two of the three points are selected as the offspring. The linear crossover was first tried by Cheng and Gen [12]. They restricted the multipliers as: $\lambda_1 + \lambda_2 \leq 2, \lambda_1 > 0, \lambda_2 > 0$. A geometric interpretation on all the arithmetic operators was given by Gen and Cheng [28]. Consider any two parents

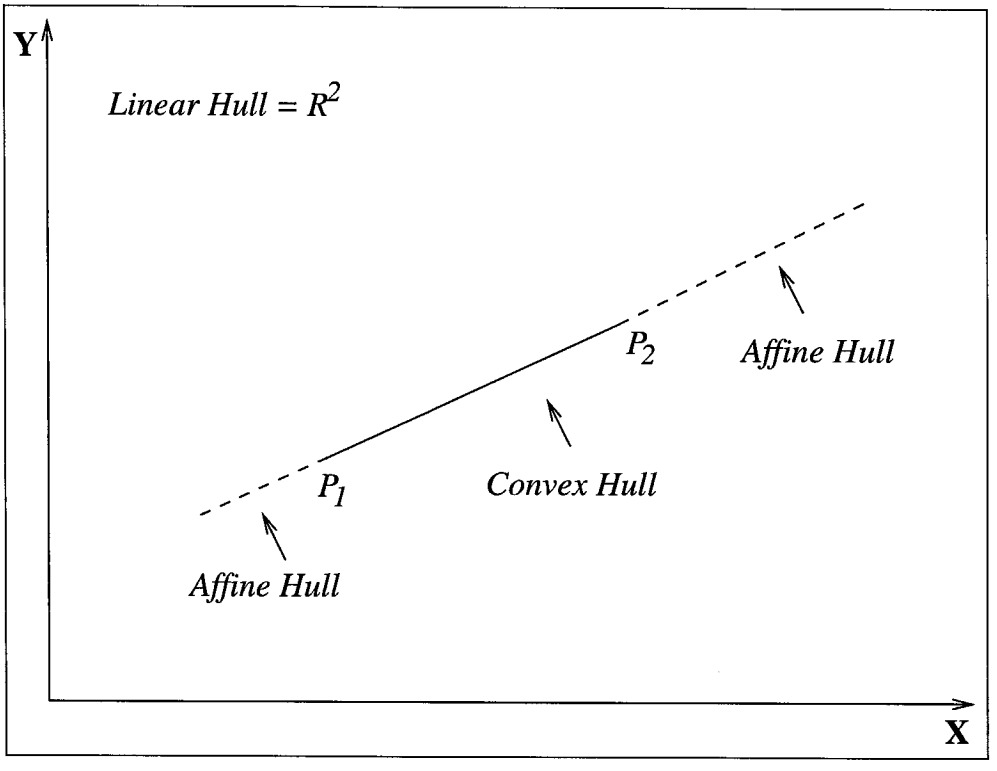


Figure 3.4: Illustration on Convex Hull, Affine Hull And Linear Hull Embedded in Two-Dimensional Space.

p_1 and p_2. The offspring generated with the convex crossover will lie in the space called convex hull. Similarly affine hull refers to the space containing the offspring generated by the affine crossover and linear hull is the space produced by the linear crossover. Obviously linear hull contains affine hull which further contains convex hull. Figure 3.4 gives an illustration on the three spaces embedded in two dimensional space.

The direction-based operators use problem-specific knowledge to produce improved offspring. As an example, Michalewicz *et al.* [52], constructed a direction-based crossover which uses the values of objective function in determining the exploration direction. The operator generates a single offspring c from two parents p_1 and p_2 according to the following rule:

$$c = r(p_2 - p_1) + p_2,$$

where

$$r \quad = \quad \text{a random number between 0 and 1.}$$

The above form assumes that the performance of parent p_2 is not worse than that of p_1. This crossover is essentially an affine crossover which always generates the offspring points lying in one side of the affine hull that is close to the better performing parent.

Among the above crossovers, the choice of which one to use always depends on individual application. Recently, BLX-α has become popular because of its superior performance with easy implementation [16]. In addition to the previously mentioned superior characteristics of BLX-α crossover, borrowing the concept shown in Figure 3.4, the advantage of BLX-α crossover can also be seen from its ability to explore the offspring uniformly over a restricted linear hull space surrounding the two parents. Such restricted hull includes the entire convex hull and two sides of affine hulls near to the parents. On the other hand, all the arithmetical crossovers only assign offspring with discrete points which could be further limited within either convex hull or affine hulls. While for the above direction-based crossover, the strong bias towards the better performing parents may aggravate the premature convergence during the early stages.

3.4.4 Integer Crossover

Since an integer coding scheme is mostly used for sequencing problems, each chromosome represents a particular sequence and each gene within a chromosome represents an order within a seqence. So the integer value of each gene are different from each other. The particular concern for integer crossovers is to make sure that all the order numbers of a sequence are assigned to all the genes of each generated chromosome. The particular focus of each crossover is how to reorder the sequences defined by the parents. There are quite a few operators developed to crossover two parent sequences under different applications. In particular, four major operators are widely used which are order crossover (OX) [18], cycle crossover (CX) [60], partially mapped crossover (PMX) [30] and edge recombination (ER) [77]. Among all these operators, the common features between any two parents are

considered as the same relative orders existing in both parent sequences. Effort has been made to preserve those common features to the offspring during the construction of various crossovers. The details of OX and ER will be introduced below to show how the common relative orders of allele values existed in both parents are preserved through the crossovers.

OX creates children which preserve the order and position of symbols in a subsequence of one parent while preserving the relative order of the remaining symbols from the other parent. First, two cut points are randomly selected among the paired parent genes. Then, the allele values between the cutpoints are copied from the first parent into the same genes of the first child. Next, copy allele values starting from the gene right after the second cutpoint of the second parent one by one into the genes of the first child with the same starting position. If current allele value is a duplicated one, the copy of that allele is skipped. If the end gene is reached for either the second parent or the first child, the copy is continued with the first gene. The process is finished until the copies of all the allele values of the second parent have been tried. The second child can be constructed by switching the roles of the parent sequences. An example is illustrated below:

$$
\begin{aligned}
Parent1: &\ A\ B\ C\ D \mid E\ F\ G \mid H\ I\ J\ K \\
Parent2: &\ K\ A\ G\ F \mid B\ D\ H \mid I\ J\ C\ E \\
Child1: &\ A\ B\ D\ H \mid E\ F\ G \mid I\ J\ C\ K \\
Child2: &\ C\ E\ F\ G \mid B\ D\ H \mid I\ J\ K\ A.
\end{aligned}
$$

So after each OX operation, for both parents, the relative orders between the two cutpoints are directly preserved by their children and the rest of the orders are also kept as much as possible without any duplications.

ER creates children which preserve edges, or immediate predecessor/successor relationships found in the parent sequences. It is implemented by constructing an edge map that lists the neighbors of each gene in the sequence. The allele value of the first gene to be placed into the child is chosen at random. The next one is chosen from the list of the first gene's neighbors. This process is continued until either the allele values of all genes are chosen or there are no neighbors to choose from for the left allele values. In the later case, any of the remaining allele values is chosen at random. In the following example, A

is chosen first, followed by B, after that the choices are arbitrary. The child produced in this way preserves the entire edges from its parents.

$$
\begin{aligned}
Parent1: &\quad A\ B\ C\ D\ E\ F \\
Parent2: &\quad D\ F\ B\ A\ C\ E \\
EdgeMap: &\quad A: \quad B, C \\
&\quad\ \ B: \quad A, C, F \\
&\quad\ \ C: \quad A, B, D, E \\
&\quad\ \ D: \quad C, E, F \\
&\quad\ \ E: \quad C, D, F \\
&\quad\ \ F: \quad A, B, D, E \\
Child: &\quad A\ B\ F\ D\ C\ E.
\end{aligned}
$$

Inspired by the above operators, Fox and McMahon later introduced intersection and union operators to directly capture the common characteristics of two parents by using the precedence matrix which stores the common predecessor/successor relationships between two sequences [26]. Another operator also often used is the subtour-chunking operator introduced by Grenfenstette [33], which works by alternately selecting segments (chuncks) from the two parent chromosomes and incorporating those into the offspring. Some other operators are designed to solve the specific problems which are examined by Gen and Cheng [28]. There is essentially no unified view on which one is better.

3.5 Mutation

Mutation has been traditionally considered as a "background" operator whose main contribution is to assure that no information is permanently lost from the maintained population. So mutation in general takes a subordinated role to amend the weakness of crossover. Usually, once the parent features are all closely matches or identical to each other, crossover will no longer produce much new information and thus lose its evolution power. This often happens as an evolution process gets converged. Under these circumstances, mutation can be used to bring diversity and introduce new features to the population. Thus, the design

of mutation should focus on its disruptive power to alternate existent features of individuals. Mutation is usually applied to a single parent each time and generate a single child. Just like crossover, the implemenation of mutation varies to different coding schemes and applications.

Mutation on a binary bit string is simply to flip several randomly chosen bits independently. Usually, during each mutation, all the bits in the parent chromosome is visited and for each bit, whether flip it or not is determined according to a specified probability called mutation rate. Usually the mutation rate is fixed as 0.5 so that on average, half of the parent bits are flipped.

Under real coding scheme, mutation can be rather different. In general, a gene is mutated within a certain value range with defined lower and upper boundary values. The basic mutation is called uniform mutation, which simply replaces a randomly picked gene with a randomly selected value within its value range. A mutation called boundary mutation goes to the extreme case which always replaces the randomly picked gene with either its lower or upper boundary value [52]. Janilow and Michalewicz introduced a mutation called nonuniform mutation which is designed for fine-tuning capabilities aimed at achieving high precision [40]. Under such mutation, a gene to be mutated is randomly determined among a given parent. Its allele value can be mutated in one of the two ways, which is randomly determined. The two ways of mutation are:

$$c_k = p_k + \Delta(t, p_k^U - p_k)$$

or

$$c_k = p_k - \Delta(t, p_k - p_k^L)$$

where

t = iteration number of an evolution process

k = k^{th} parent gene to be mutated

p_k = allele value of k^{th} parent gene

c_k = allele value of k^{th} child gene

p_k^L = lower allele bound of k^{th} parent gene

p_k^U = upper allele bound of k^{th} parent gene.

The function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the function value approaches 0 as t increases. This property causes the operator to explore the space uniformly during the early evolution stages while later focus on narrow portion of the space. The function $\Delta(t, y)$ is defined as:

$$\Delta(t, y) = y \cdot r \cdot (1 - \frac{t}{T})^b$$

where

r = random number from $[0, 1]$

T = maximum iteration number of an evolution process

b = nonuniformity parameter.

Gen *et al.*, gave another mutation based on the idea of Taylor expansion of a continuous differentiable function [27]. The mutation formula is:

$$c_k = p_k + r \cdot d$$

where

k = k^{th} parent gene to be mutated

p_k = allele value of k^{th} parent gene

c_k = allele value of k^{th} child gene

r = a nonnegative random number.

And d is the gradient value determined by:

$$d = \frac{f(p_1, ..., p_k + \Delta p_k, ..., p_n) - f(p_1, ..., p_k, ..., p_n)}{\Delta p_k}$$

where

$$\begin{aligned} f &= \text{fitness function} \\ n &= \text{length of each chromosome} \\ \Delta p_k &= \text{a small real number.} \end{aligned}$$

Here d is approximated as the gradient direction of the fitness function, by updating the gene along such direction, the mutation essentially increases the chance to replace the gene with better performing feature. The vein of such approach is quite similar to the direction-based crossover.

Most used mutation for integer coding scheme is to randomly select two genes of each to-be-mutated chromosome and their allele values are exchanged. Such mutation is called swapping mutation.

Again, there is no golden rule of choosing the right mutation for any applications. Recently, heuristic mutations have been often constructed to take advantage of particular knowledge in solution space so that they become more effective to either disrupt some predicted bad features or insert certain expected good features [28].

3.6 Schemata Theorem

In Holland's work, the dynamics and the power of genetic algorithms were explained by using the concept of schema, which is based on the strings whose string elements are drawn from a finite alphabet set.

Definition 1 Schema

A schema is a similarity template describing a subset of strings with similarities at certain string positions.

The above definition can be applied to the schemata (schema*s*) of binary strings under the ternary alphabet $\{0, 1, *\}$ with $*$ denoted as *don't care* symbol. As an example, consider the strings and schemata of length 5. The schema *111* describes a subset with four members $\{01110, 01111, 11110, 11111\}$. In another example, for a given subset such as {00001, 01001, 00011, 01011}, all the members have common *0*s in the first and third bits and common *1*s in the fifth bit and therefore the set can be represented as 0*0*1. So the defined bits of a schema represents the common bits shared by all the members of the underlying string set. In other words, each schema carries the information of similarities among a set of strings. On the other hand, consider any single binary string of length 5: 11111, for example. This string is a member of 2^5 schemata because each position may take on its actual value or a *don't care* symbol. In general, a particular string with length l contains 2^l schemata. For a population of size n maintained under a genetic algorithm, there are somewhere between 2^l and $n \cdot 2^l$ schemata existing in each generation. Therefore, the process of selecting individual strings and evolving them through genetic operators (mainly crossovers) to produce new different strings can be effectively viewed as the process of selecting a correspondent set of schemata and evolve it into a new set during which some old schemata are disrupted, some old ones are preserved and some new ones are generated. In this way, although on the surface level, a genetic algorithm directly evolve a moderate size of individual strings, a large amount of schemata flow which represent the important feature similarities are effectively processed on the background. It is the evolution of such huge amount of information flow that brings the true power to the genetic algorithms. In particular, such evolution is achieved completely in parallel without any special bookkeeping or extra memory and thus is named as *Implicit Parallelism* by Holland [35].

It is further shown that not all the schemata are processed equally even without selection bias. Depending on the properties of each schemata, some are easier to be disrupted while some are easier to be preserved and generated. Two important properties of a schema are its order and defining length defined below:

Definition 2 Schema Order

The order of a schema H, denoted by $o(H)$, is simply the number of fixed positions (the number of 1s and 0s for binary strings) present in the template.

Definition 3 Schema Defining Length

The defining length of a schema H, denoted by $\delta(H)$, is the distance (the number of string positions) between the first and last fixed position.

As an illustration, the order and defining length of the previously presented schema $0*0*1$ are 3 and 4 respectively. Based on the intuition, a schema with low order and small defining length is more likely to survive during crossover and mutation as compared to the one with high order and large defining length. A strict relationship has be given by Holland with the schema theorem [35]:

Theorem 1 (Schema Theorem) *The expected number of copies of a particular schema H from one generation to the next under selection, crossover and mutation is given by the following relationship:*

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H,t)}{\bar{f(t)}}[1 - p_c(t)\frac{\delta(H)}{l-1} - o(H)p_m(t)], \tag{3.2}$$

where

$$\begin{aligned} H &= \text{schema to be examined} \\ t &= \text{iteration number} \\ m(H,t) &= \text{expected number of copies of schema } H \text{ at iteration } t \\ f(H,t) &= \text{observed fitness of schema } H \text{ at iteration } t \\ \bar{f(t)} &= \text{observed average fitness of entire population at iteration } t \\ p_c(t) &= \text{crossover rate at iteration } t \\ p_m(t) &= \text{mutation rate at iteration } t \end{aligned}$$

$\delta(H)$ = defining length of schema H

$o(H)$ = order of schema H.

The fitness (performance level) of a schema is measured through the average fitness value of all the maintained individuals that instance the schema. The schema theorem shows that short, low order, above-average schemata receive exponentially increasing trials while those below-average schemata receive exponentially decreasing trials in subsequent generations. Even though it is only directly applicable to a single generational cycle, one can get an intuitive feeling for the dynamics of genetic algorithms by considering what happens to a schema that consistently has an observed fitness that is higher than the population average.

The schema theorem also tells that certain kind of schemata play an important role in the action of genetic algorithms and they are defined as building blocks:

Definition 4 Building Blocks

The building blocks are defined as those highly fit schemata with short defining length and low order.

With the introduction of building blocks, the overall dynamics of genetic algorithms is often expressed as "Building Block Hypothesis":

Theorem 2 (Building Block Hypothesis) *A genetic algorithm seeks near optimal performance through the juxtaposition of short, low-order, high performance schemata, or building blocks.*

The building block hypothesis essentially indicates that an evolution search is the process of evolving the building blocks with increasing orders. At any iteration, the maintained building blocks represent all the common features of highly performing individuals and define the promising search regions for the next iteration. During the initial stage of an evolution process, individual features are rather different and thus the individual performance levels are rather scattered. The survived building blocks remains many with low orders. Due to the selection pressure, more individuals are sampled within the region defined by the

relatively better performing building blocks because they will give larger chance to obtain individuals with higher fitnesses. As the process moves on, such competition will evolve more and more individuals within fewer and fewer regions defined by the decreasing number of survived building blocks. This will lead to closer features between individuals and increase the chance to grow building blocks with higher orders. Again, the higher order building blocks will further direct the search towards a narrower region. Therefore, such iterative process will eventually lead to a searched optimum point within a high-fitness narrow region defined by an optimal high order building block.

3.7 Deception Analysis

As no surprise, genetic algorithms could fail. The above section indicates that a genetic algorithm depends upon the recombination of building blocks to seek the best points. However, if the dominant building blocks are misleading during an evolution, the outcome may only stay at some local optimal point. The discussion of deceptions here is to show what are the main factors that lead to the deceptions and how the dynamic behavior of a genetic algorithm is affected and finally how to make the best try.

Davidor summarized three elements contribute to deceptions [17]: the structure of the solution space, the representation of the solution space and the sampling error as a result of finite and often small generation sizes. These three elements are not necessarily linked and the effect of each of them is not fixed. Davidor introduced the concept of epistasis which is borrowed from genetics and further discussed the connection between epistasis and the three influencing elements and showed that the level of deception can be quantitatively measured through the level of epistasis. As a start, the definition of epistasis is given below:

Definition 5 Epistasis

Epistasis refers to the situation where one gene masks or modifies the expression of another gene or in other words, the fitness contribution from one gene depends on another gene of the same chromosome.

The motivation of applying an epistasis analysis is the following. If a representation contains very little or no epistasis, no individual string element is affected by the values of other elements, and therefore optimization can be simply achieved by a greedy algorithm with a bit-wise maximization. At the other end when a representation is highly epistatic, too many elements are dependent on each other. Unless a complete set of unique element values is found simultaneously, no substantial fitness improvements can be noticed. This well indicates that there is too little structure in the solution space and a genetic algorithm will most likely drift and settle on a local optimum. Therefore, the level of epistasis contains the information about the amount of nonlinearity in terms of gene interaction embedded in a given representation, and thus gives an indication of the exposed structure of the solution space. A further investigation has been conducted to show that the level of epistasis is considerably sensitive to the sampling error. As the generation size diverts from a grand large size to smaller finite ones, the genetic algorithm suffers more and more nonlinearity. The calculations have indicated that the epistasis consists of two elements: the base epistasis resulted from the representation and an extra epistasis resulted from sampling noise. Even though the calculations on the level of epistasis conducted by Davidor were based on the binary strings with simple fitness functions and may not be applicable to general genetic algorithms, the demonstrated connection between the epistasis and the factors leading to the deceptions provides a strategy to make an overall prediction on the hardness of a problem through some judgments on the epistasis of any constructed chromosome.

Grefenstette further exposed two aspects that challenge any genetic algorithm to properly process schemata and thus lead to evolution failure [32]. One is the collateral convergence and the other is the fitness variance within schemata. As an example to illustrate the

collateral convergence, consider the following two first-order schema competitions:

		$f(H)$
H_A :	$0\#...\#\#\#...$	9
H_B :	$1\#...\#\#\#...$	1
H_C :	$\#\#...\#0\#...$	6
H_D :	$\#\#...\#1\#...$	4,

$f(H)$ is the observed fitness of each schema based on the actual sampled individuals. Assume that initial population is selected uniformly and the generation size is large enough so that the observed fitness at time $t = 0$ well approximates the true average fitness of each schema. Then, the schema theorem predicts that the expected allocation of trials in the second generation is as follows:

		$Pop\%$ *in* H *at time* $t = 1$
H_A :	$0\#...\#\#\#...$	90
H_B :	$1\#...\#\#\#...$	10
H_C :	$\#\#...\#0\#...$	60
H_D :	$\#\#...\#1\#...$	40,

Because the relative fitness ratio is higher in the first competition between H_A and H_B than the second competition between H_C and H_D, the theorem predicts that the population will begin to converge more rapidly with respect to the H_A than H_C. Such phenomenon is referred as *collateral convergence*. Because of that, the expected sampling of each competing schema becomes biased and thus their observed fitnesses are no longer aligned with the true average fitnesses. For example, it is expected that 90% of the observed representatives of H_C would have a 0 in the first position because of the convergence effect from H_A. Therefore, the schema theorem can no longer predict the actual convergence behavior of the

next iteration. In other words, depending on the degree of collateral convergence, the actual evolution convergence trajectory in the end may be far away from what building block hypothesis predicts. The second aspect tells that if the fitness variance within schemata is high, due to a limited generation size, the sampling error in the initial random generation will produce large errors in the estimate of each schema's true average fitness which again could lead to improper schemata processing.

In general, the sampling error is unavoidable due to the limited sampled generation size and the used pseudorandom generators. What is crucial here is the schemata variances. If schemata have high variance, the convergence behavior will be very sensitive to sampling error as well as collateral convergence and thus more likely to be improperly processed and mislead the evolution direction. One way to measure the schemata variance under a specific representation is through the epistasis analysis. Obviously, one effective way to improve this is to choose a good representation so that the epistasis level and schemata variance are low. However, in reality, the choice of representation are quite limited. Another approach is to choose appropriate evolutionary strategies so that the evolution process can be converged more gently which can provide more chances to make the self-adjustment if anything goes wrong. Various selection schemes and genetic operations have been proposed to provide such active control which will be covered in the following sections.

3.8 Selection Scheme

3.8.1 Overview

The task of each selection is to select individuals to be evolved for the next iteration. Selection schemes are the central component to provide the evolution competition between individuals based on their fitness values and thus mimic Darwinian natural selection process. Every selection scheme tends to bias towards the better performing individuals which are believed to have better chance to carry good-featured genes for evolving children with even better performance. Selection pressure is used to refer to the degree of such bias or equivalently the level of fitness competition. Typically, low selection pressure is indicated at the start of an evolution process in favor of a wide exploration of the search space, while

high selection pressure is recommended towards the end in order to exploit the most promising search region. The selection controls the overall convergence of an evolutionary search. Many selection methods have been proposed. Some of them are presented here with the focus on illustrating how the selection pressure is provided. For a more complete coverage, refer to Gen and Cheng's summary [28]. Most of the selection schemes can be divided into two categories: stochastic and deterministic selections. The discussion on each of them are separated into two subsections.

3.8.2 Stochastic Selection

Under stochastic selection, all the selection schemes use a random sampling approach to determine the selection of individuals. Usually, each sampling approach consists of two stages starting with assigning a sampling rate or selection probability to each individual and ending with selecting individuals according to their sampling rates. The selection pressure is mainly determined at the stage where the sampling rates are assigned.

There are majorly two mechanisms proposed to determine the sampling rates. One is the proportionate assignment and the other is the rank assignment. The proportionate assignment determines the sampling rate for each individual to be proportional to its fitness as follows:

$$p_i = \frac{N f_i}{\sum_{1 \leq j \leq N} f_j},$$

where

$$\begin{aligned} i &= i^{th} \text{ individual in current generation} \\ N &= \text{current generation size} \\ p_i &= \text{sampling rate of individual } i \\ f_i &= \text{fitness of individual } i. \end{aligned}$$

It turns out that such assignment often requires the prescaling of the fitness values [30]. Without fitness scaling, the selection pressure reflected through the sampling rates will

mostly not be properly controlled. In most cases, at the early stage of an evolution, individual features and their performance levels vary a lot, so there is a tendency for a few individuals to have relatively rather high fitnesses. Since the sampling rates are determined by the relative fitnesses, these few individuals may dominate the selection process which will lead to premature convergence. On the other hand, during the late evolution stage, individual fitnesses are fairly close to each other. The sampling rates are almost the same for every individual and thus the needed selection pressure to ensure a proper exploitation can not be provided. With fitness scaling, however, all the raw fitnesses will be converted to scaled values so that the relative ratios can be adjusted to obtain the desired sampling rates. Several scaling techniques are summarized by Gen and Cheng [28]. As an example, linear scaling assigns the best individual with a fixed sampling rate and thus prevents it from reproducing too many at the early evolution stage. The method is shown as below:

$$f_i' = a \cdot f_i + b$$

where

$$\begin{aligned} i &= i^{th} \text{ individual in current generation} \\ f_i' &= \text{scaled fitness of individual } i \\ f_i &= \text{raw fitness of individual } i \\ a, b &= \text{real parameters.} \end{aligned}$$

The parameters a and b are normally selected so that average individuals receive one sample on average and the best individual receives the specified number of samples (usually two). This method, however, may produce negative scaled fitnesses because of largely different raw fitnesses especially during the early stage. Mostly, these negative values have to be assigned as zeros which appears too artificial. Another example is Boltzmann scaling [53]:

$$f_i' = e^{f_i/T} \tag{3.3}$$

where

$$
\begin{aligned}
i &= i^{th} \text{ individual in current generation} \\
f_i' &= \text{scaled fitness of individual } i \\
f_i &= \text{raw fitness of individual } i \\
T &= \text{control parameter serving as temperature.}
\end{aligned}
$$

Under this approach, the scaled fitnesses can be continuously tuned according to the change of a single temperature value T. The adjustment of T can be nicely explained by the analogy of the temperature control in a thermal process. Consider each individual as an atom and its fitness as the energy level of the atom. The evolutionary converging process is taken as the process of reducing the atom energy levels by decreasing the temperature. So T is initially set high and then is gradually decreased. Coincidently, in form (3.3), the initial high-valued T leads to small selection pressure at the initial stage and as T decreasing later, the selection pressure increases at the later stage, which is exactly needed. Also such scaling eliminates the occurrence of negative values. In general, the proportionate sampling assignment can be rather sensitive to the relative fitness distribution and thus the convergence behavior can be easily affected by sampling error. Even though prescaling methods can reduce or even eliminate such effect, they usually require the proper setting of parameters which are mostly determined through trial-and-errors. Rank assignment is introduced by Baker to avoid the direct dependency on fitness values [8]. It has been shown to help in the avoidance of premature convergence and to speed up the search when the population approaches the final stage [78]. A rank assignment starts by sorting the population from the best to the worst and assign the sampling rate of each individual according to the ranking. Two methods are commonly used: linear ranking and exponential ranking. The following formula gives the linear ranking:

$$p_i = q - (i-1) \times \frac{q - q_0}{N - 1}$$

where

$$\begin{aligned} i &= i^{th} \text{ individual in current generation} \\ N &= \text{current generation size} \\ p_i &= \text{sampling rate of individual } i \\ q &= \text{probability of the best individual} \\ q_0 &= \text{probability of the worst individual.} \end{aligned}$$

In this case, the sampling rates are linearly proportional to their ranks. The selection pressure can be adjusted through both q and q_0. An example of exponential ranking is proposed by Michalewicz as the follows [53]:

$$p_i = q(1-q)^{i-1}$$

where

$$\begin{aligned} i &= i^{th} \text{ individual in current generation} \\ p_i &= \text{sampling rate of individual } i \\ q &= \text{probability of the best individual.} \end{aligned}$$

A larger value of q implies stronger selective pressure. In general, there is no clear judgment on which ranking scheme is better. However, linear ranking appears to receive more attention because of its simplicity and easy control [78].

Once the sampling rates are assigned to individuals, the selection stage is followed which is implemented through a sampling algorithm. As an example, a famous one is called roulette wheel selection. The "roulette wheel" is composed of all the individual sampling rates. The "spinner" has one pointer. Each "spin" of the "wheel" will select one individual according to the pointer position. N independent "spins" are needed to select

N individuals. The "wheel" remains unchanged between "spins." Another popular sampling algorithm is called stochastic universal sampling introduced by Baker [7]. Figure 3.5 shows the pseudocode of the procedure. The same "roulette wheel" is used, however, with different "spinner." There are N equally spaced pointers on the "spinner." The pointers are exactly 1.0 apart. During the sampling, only one single "spin" is needed with all the selected individuals according to the positions of N pointers. Baker presented three measures of the performance of selection algorithms: bias, spread and efficiency [8]. Bias defines the absolute difference between actual and expected selection probabilities of individuals. Spread is the range in the possible number of trials that an individual may achieve. Efficiency is related to the overall time complexity of the algorithms. Roulette wheel selection tends to give zero bias, but potentially inclines to spread unlimitedly. It can generally be implemented with time complexity of the order of $N \log N$ where N is the population size. Stochastic universal sampling (SUS) is another single-phase sampling algorithm with minimum spread, zero bias and the time complexity of SUS is in the order of N.

3.8.3 Deterministic Selection

Under deterministic selection, the selection schemes select individuals according to their fitnesses in a deterministic way. The generational replacement which replaces the entire set of parents by their offspring is an example. However, under such approach, the evolved good gene features can be easily lost due to any of the sub-performing iterations and thus the evolution process may either suffer long waiting time or easily converge to a suboptimal point because of the loss of best searched features. To amend such problem, a steady-state selection is proposed by Syswerda [74], where only n ($n <$ generation size) children are produced during each iteration and n worst parents will be replaced by the children to form the next generation. By doing so, the best evolved gene features will always be preserved. However, the produced children may not always be better than the worst n parents which disobeys the replacing purpose. In addition, such approach will not best utilize all the information available at each generation to evolve the offspring which may lead to low productivity. Elitist selection was introduced by Eshelman to bring the cross-generational competition [24]. During each iteration, N (generation size) children are produced. The

Variables:

sum = *Accumulated Sampling rates*

ptr = *Positions of Pointers*

N = *Number of Individuals to Be Selected*

$rand()$ = *Returns A Random Value Uniformly Distributed Within* $[0, 1)$

StochasticUniversalSample

```
1   sum ← 0;
2   ptr ← rand();
3   for k ← 1 to N {
4     sum ← sum + sampling rate of individual k;
5     while (sum > ptr) {
6       select individual k;
7       ptr ← ptr + 1;
8     }
9   }
```

Figure 3.5: Pseudocode for Stochastic Universal Sampling.

children will compete with parents so that the best N individuals among both parents and children will be selected as the next generation. In this way, a child only replaces a member of parent population if it is better. Such approach is rather conservative in terms of preserving good gene features. Eshelman further showed that by combining such selection scheme with highly disruptive crossovers, an overall well balance between the exploration and exploitation can be obtained [24]. Later, Bäck and Hoffmeister introduced $(\mu + \lambda)$ and (μ, λ) selections [6] [5]. Under $(\mu + \lambda)$ selection, μ parents and λ offspring compete for survival and the μ best ones are selected as next generation. Under (μ, λ) selection, the best μ individuals are selected out of λ offspring ($\mu < \lambda$). $(\mu + \lambda)$ selection is essentially a modification of elitist selection. (μ, λ) selection is essentially a generational replacement approach with reduced risk of losing good genes by generating more children than needed and selecting the best ones.

3.9 Genetic Operation

The task of genetic operation is to determine how the selected individuals participate the offspring breeding through genetic operators which mainly include crossover and mutation. So far, most developed evolution algorithms do not have special control on this step. That is, during crossover, the pairing of parents are randomly picked from all the survived individuals and if applicable, a fixed crossover probability is used to determine whether each paired genes are under crossover or not, and during mutation, individuals are randomly picked and if applicable, a fixed mutation probability is used to determine whether each gene is mutated or not. Nevertheless, some special control has been provided which is mainly focused on two aspects. One is to adaptively adjust the operator probabilities as the evolution process goes on and the other is to pair individuals based on their features, both of which will be discussed in the following.

One try on adapting operator probabilities has been made by Davis [19]. The basic idea is to adapt the probability that a genetic operator will be applied in reproduction based on the performance of the operator's offspring. During an evolution process, there are several operators available to be used during each reproduction. Only one operator can be picked

at a time. The probability of being picked for each operator is adaptively updated. Two intuitions are used to underlie the adaptation principle. One is that the probability of applying an operator is in proportion to the observed performance of the individuals created by that operator in the course of a run. The second is that each time not only the operator that produces a good child will be rewarded but also the operator that set the stage for this production will be credited. Through the actual tests, Davis concluded that such adaptation mechanism allows rapid parameterization of operator probabilities across the range of potential genetic algorithms. As another example, Ng *et al.*, used adaptive mutation scheme in a genetic algorithm [43]. Such scheme varies the mutation rate proportional to the similarities between the paired parents. So that when individuals become converged, the mutation rate increases in order to effectively provide background diversity as needed. From the two examples, it can be seen that the advantage of using an adaptive operator probability is to avoid the prior experience in setting the otherwise fixed operator probability.

Eshelman introduced a special mechanism to avoid incest during the individual pairing [24]. During the genetic operation, two members of the parent population is randomly chosen without replacement and paired for mating. Before mating, however, the Hamming distance between the two is calculated and if half that distance does not exceed a difference threshold, they are not mated and are deleted from the child population. In addition, the difference threshold is dynamically decreased whenever the deletion occurrence is too much. It is shown that such mechanism not only avoids the random chances that the same parent are paired which wastes the entire crossover effort, but also effectively maintains the diversity of population so as to prevent premature convergence.

The above few examples show some effort towards developing a more robust genetic operation so that the applied genetic operator can function as needed. At this point, the roles of genetic operators should be viewed at a higher level. Mutation serves to create random diversity in the population, while crossover serves as an accelerator that promotes emergent behavior from individuals. What is critical here is the relative importance of diversity and construction which equivalently means the search balance between exploration and exploitation. It is true that such balance ought to be mainly controlled by the selection scheme through the selection pressure from the fitness competition. However, such control

power provided by simple duplication of better performing individuals may not be strong enough. Random paring gives weakest incentive to battle with sampling error. On the other hand, any proper extra strategy on genetic operation can provide the additional power to control the amount of exploration and exploitation and thus help selection scheme to direct the evolution process effectively.

Chapter 4

Bulk Wet-etching and SEGS Simulator

4.1 Bulk Wet-etching

There are mainly two types of techniques which have been developed to produce MEMS devices: surface micro-machining and bulk micro-machining, both of which have been briefly mentioned in Chapter 1. This Chapter will focus on the introduction of a particular type of bulk micro-machining technique: bulk wet-etching process and a simulator developed previously at Caltech called SEGS [38].

The bulk wet-etching process is a popular technique to produce high-aspect ratio 3D mechanical structures. The detail steps have been described by Hubbard [37]. A typical process starts with a wafer cut along some crystal orientation such as (100). Then the preparation of the wafer includes a cleaning process followed by a deposition of a thin mask layer and then a drying in a high temperature furnace. The commonly deposited mask layer is silicon dioxide or silicon nitride due to their high selectivity. Next a uniform layer of photoresist is spun onto the mask layer. Such photoresist layer is used to define the geometric pattern of the mask layer through the light exposure, developing and mask removing steps. Finally, the wafer with patterned mask-layout is placed in a container of etchant to experience the etching process. The popular etchants are KOH (potassium hydroxide), EDP (ethylene diamine pyrocatecol) and TMAH (tetramethyl ammonium hydroxide). Since the etching rates of most etchants are sensitive to environment temperature, the temperature control mechanism are needed to stabilize the entire process. After a set period of etching time, the wafer is then removed. The remaining masking layer can be stripped, leaving a

three-dimensional shape in the silicon wafer.

It is known that the physical properties of silicon, especially the atom densities, depend on its crystal planes. These crystal planes are characterized by their Miller indices (h,k,l) which are the inverses of the (x,y,z) intercepts for the planes. For example, the (100) plane intersects the x axis but not the y or z axes. Because of such crystalline anisotropic properties, when silicon is etched, different planes are etched at different rates. The more dense planes are etched at a slower rate; The etching rates are also affected by the etchant with particular concentration and temperature. In general, the (111) planes are the most dense planes and thus etched the slowest. It is also observed that etching rates under most etchants preserve the same symmetry as the crystal planes. For example, (100) wafers tend to produce shapes with four-fold symmetry.

4.2 SEGS Simulator

Various simulation methods have been briefly mentioned in Chapter 1. The SEGS method developed by Hubbard and Antonsson [38] uses edge segmentation approach to compromise the overall performance from the geometric analysis and cellular automata. In SEGS, the etched profile at each time step is determined through two types of interactions: local and global. Local interactions deal with the local intersections as planes appear and disappear at etched corners; while global interactions take care of the global intersections arising when two originally separated parts of a mask shape grow during etching and their etched shapes are later merged. The geometric analysis approach is used to efficiently solve the local interactions while the idea of cellular automata scheme is adopted to accurately detect any global object interactions. During the local interaction, the calculations involve the two nearest neighbors and check the relative validity of adjacent line segments. A segment is valid if it lies in the still unetched half planes of its two neighbors in terms of the local tangents and normal:

$$(n_i \cdot t_{i-1})(n_i \cdot \delta_{i,i-1}) \geq 0,$$

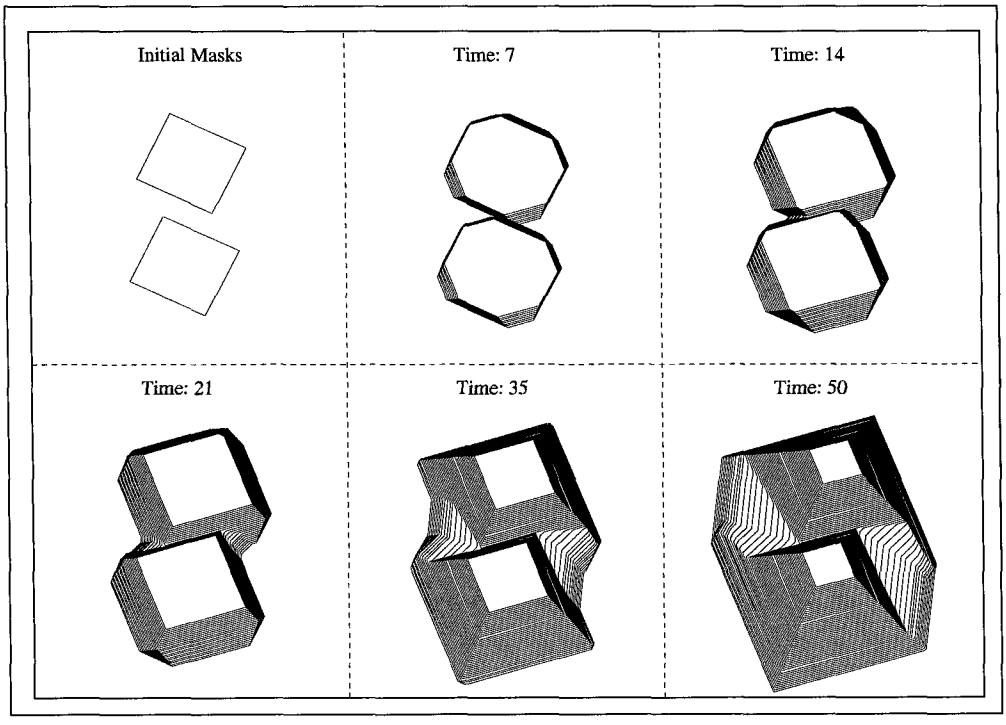


Figure 4.1: Etching Results Under Various Time Steps Simulated by SEGS for Two (Originally Square) Holes Merging.

where

$$
\begin{aligned}
i &= i^{th} \text{ segment} \\
n &= \text{segment normal} \\
t &= \text{segment tangent} \\
\delta &= \text{segment position delta vector.}
\end{aligned}
$$

The invalid segments are eliminated and the test is repeated for all the segments in each shape. The global cellular calculations involve an overlayed grid. Each line segment location is written to the grid. Global interactions are easily detected when a grid cell that has been previously written into is overwritten. It is shown that the two hybridized methods decouple local and global interaction calculations and permits each to be optimized individually.

Figure 4.1 shows a typical simulated result. Initially, the mask-layout consists of two

identical square masks with four sides aligned in <100> directions. The area inside the mask is to be etched. Each snapshot corresponds to an etched shape at a different time step. The output etched shapes are presented through a series of polygon layers.

Later in a mask-layout synthesis application, the SEGS simulator will be used with the later developed evolutionary algorithm to provide the simulated shape for each candidate mask-layout.

Chapter 5

Shape Matching

5.1 Overview

In this thesis, the goal is to search for a mask-layout that produces globally optimum etched shape. So each mask-layout produces a solution point. The shape mismatch between any two solutions can be treated as the feature difference or the distance between any two solution points. A 2D polygon shape matching algorithm is needed to obtain such shape mismatch. In addition, the later synthesis test through a bulk-etching simulator requires a measurement on the shape mismatch between each 3D etched shape and the user specified 3D target shape. Thus, a 3D shape matching algorithm is also needed.

There are many matching algorithms developed especially in the computer vision community to recognize a shape that is translational, rotational and size invariant to some known shapes. Such techniques are often referred as model-based matchings [42]. Most of these methods are 2D matching algorithms applied to image recognition. Among those techniques, one popular category is the polygonal approximation scheme suggested by Pavlidis [76]. Under such scheme, an arbitrary 2D shape is approximated by a polygon and then the shape closeness is measured through the matching between polygons. In this chapter, some of the polygon matching algorithms developed under this category will be examined. One particularly useful algorithm, called $L2$ distance polygon matching algorithm, will be introduced in more detail. In terms of 3D matching, the literature is not plentiful. Most methods searched are limited to some specific object features such as the work done by Boyse [41]. One general approach is proposed by Bribiesca [10], which

however requires a large amount of computation. Inspired by the layer representation of 3D etched shapes used in the SEGS simulator, 3D shape matching is decomposed into 2D shape matching.

5.2 2D Polygon Shape Matching

In general, the mismatch between any two polygons can be measured through a distance function which has the properties of a metric [57]. That is, let A and B denote any two polygons and $d(A, B)$ represent the distance function of A and B, then $d(A, B)$ satisfies the following conditions:

1. $d(A, B) \geq 0$ for all A and B.
2. $d(A, B) = 0$ if and only if $A = B$. A shape resembles itself.
3. $d(A, B) = d(B, A)$ for all A and B (Symmetry). The order of comparison does not matter.
4. $d(A, B) + d(B, C) \geq d(A, C)$ for all A, B, and C (Triangle Inequality). The triangle inequality fits the intuition. It implies that if A is very similar to B and B is very similar to C, then A and C should not be too dissimilar.

Several distance functions have been proposed. Atallah [54] introduced Hausdorff distance for any two convex polygons. Let p be any point and $d(p, A)$ denote the distance from p to polygon A. That is, $d(p, A)$ is zero if p is in the interior of A, otherwise it is the shortest distance from p to the boundary of A or equivalently the distance from p to the point of A that is nearest to p. Then the Hausdorff distance between polygon A and polygon B is defined as:

$$d_H(A, B) = d_H(B, A) = max\left\{\max_{p \in B} d(p, A), \max_{q \in A} d(q, B)\right\},$$

i.e., the maximum distance from any point of any polygon to the other polygon. The obvious the drawback of this distance is the limitation of convex polygons. Later, Cox *et al.* [61],

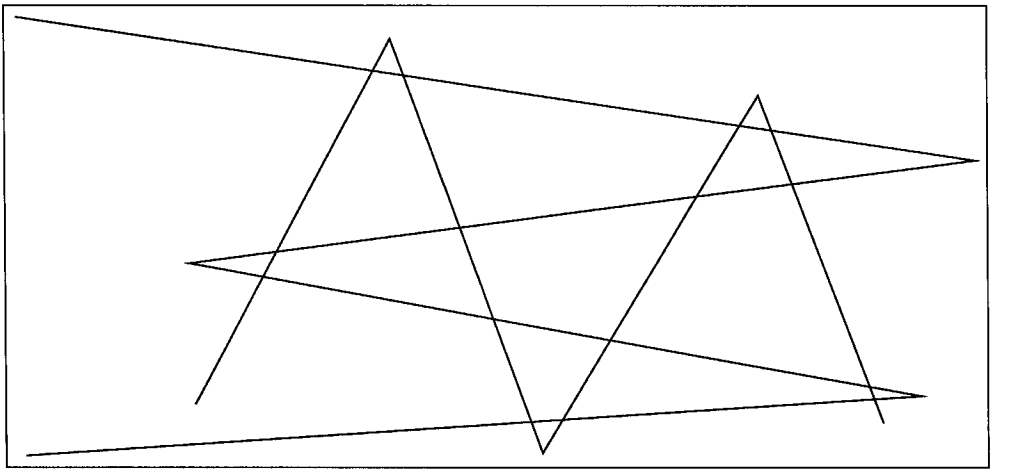


Figure 5.1: Failure Case for Hausdorff Distance Function.

revised the Hausdorff distance as follows:

$$d_{CMMR}(A,B) = d_{CMMR}(B,A) = \sum_{p \in V(B)} d^2(p,A) + \sum_{q \in V(A)} d^2(p,B),$$

where

$$\begin{aligned} V(A) &= \text{vertex set of polygon } A \\ V(B) &= \text{vertex set of polygon } B, \end{aligned}$$

i.e., the sum of the distance squares from each vertex of each polygon to the other polygon. Such distance function is still initially applied to convex polygons, however later, Atallah *et al.* [55], extended it to any polygons including concave ones. Nevertheless, Alt and Godau [34] argued that since the Hausdorff distance only takes into account the sets of points on the boundaries and does not reflect the course of the boundaries, it is not an appropriate measurement when the course of the boundaries are important. Figure 5.1 shows a failure case where the Hausdorff distance is small but the two boundaries do not resemble each other at all. So Alt and Godau further introduced a new distance function called FRÉCHET distance. The exact formulation of the distance function is given in reference [34]. A popular illustration is the following: suppose a man is walking his dog and he

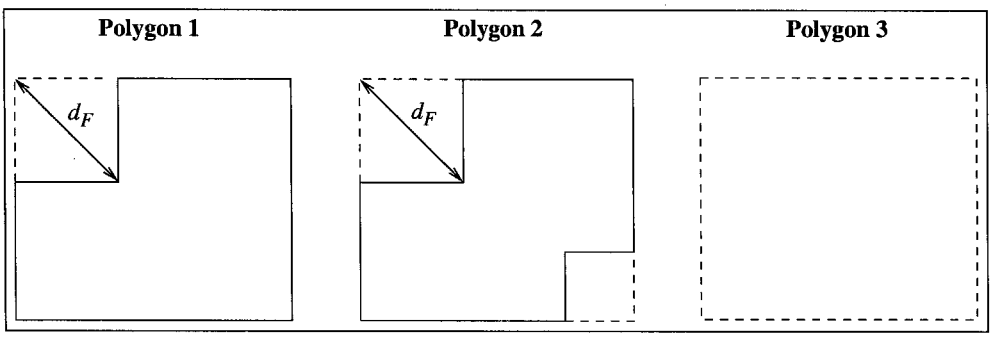


Figure 5.2: Failure case for Fréchet Distance Function.

is walking on the boundary of polygon A and his dog on the boundary of polygon B. Each time both man and dog can choose different starting points and speeds and then start walking until both of them finish one cycle and determine the required length of a leash. The task is to seek the optimal speeds and starting points for each of them so that the required length of a leash is minimum. Such minimum length is the FRÉCHET distance between polygon A and polygon B. In other words, this measurement treats each boundary mismatch as a single worst distance between any two boundary points on the two compared polygons. The special effort made is that it tries all possible matches so that the best of all the worst distances is picked as the final distance. It seems that such distance calculation requires a large amount of computation, in fact, Alt and Godau were able to show that the entire runtime for polygon matching can be $O(pqlog(pq))$, where p and q are the number of the edges of each compared polygons. The major drawback of such an approach, however, is that it still uses a single two-point distance to determine the mismatch of two entire polygon boundaries. Sometimes such a determination may not align with an intuitive notion of shape-resemblance. Figure 5.2 shows that two fairly different polygons are matched against the same third polygon and yield the same Fréchet distances. In the figure, both "Polygon 1" and "Polygon 2" are matched against "Polygon 3". Two copies of "Polygon 3" with dotted contours are aligned with the other two to show the measured Fréchet distances which are exactly the same under both matchings.

The final choice of the distance function used in this thesis is the slight modification based on the one developed by Arkin *et al.* [22], which is called $L2$ distance function and is applicable to any polygons. Arkin *et al.*, used turning function $\Theta_A(s)$ to represent the

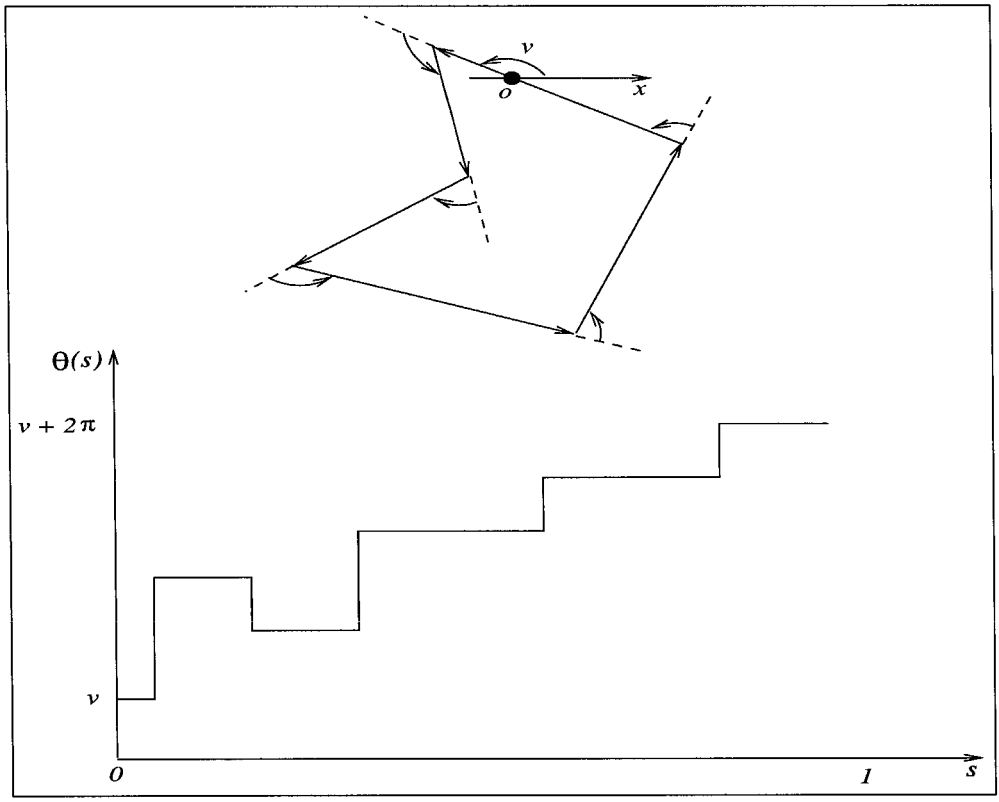


Figure 5.3: The Turning Angles and Turning Functions of A Simple Polygon.

boundary of each polygon A. As illustrated in Figure 5.3, the function $\Theta_A(s)$ measures the angle of the counterclockwise tangent as a function of the arc length s, measured from some reference point O on A's boundary. Thus $\Theta_A(0)$ is the tangent angle v at point O with respect to some reference orientation which is usually set as x-axis. $\Theta_A(s)$ keeps track of the angle turnings that take place at all the vertex corners, increasing with left-hand turns and decreasing with right-hand turns. Each polygon size is scaled to a unit size. The turning function for a polygon is a piecewise-constant function defined between 0 to 1 which makes further computations particularly easy and fast.

With the definition of turning angles, the distance function of the shape mismatch between polygon A and polygon B is:

$$d_p(A,B) = \left(\min_{\Theta \in R \; t \in [0,1]} \int_0^1 |\Theta_A(s+t) - \Theta_B(s) + \theta|^p ds \right)^{\frac{1}{p}}, \tag{5.1}$$

where

$$
\begin{aligned}
d_p(A,B) &= p\text{-norm distance value} \\
\Theta_A(s) &= \text{turning function of polygon } A = \text{turning function } A \\
\Theta_B(s) &= \text{turning function of polygon } B = \text{turning function } B \\
t &= \text{shifting of reference } O \\
\theta &= \text{rotating of polygon } A.
\end{aligned}
$$

To interpret the above form, first ignore t and θ and the *min* operator, then the above formula essentially calculates the difference between the turning function A and turning function B with the only extra feature that the difference is expressed in a p-norm. Then add t into the formula, which means the reference O is moved along the A's boundary by amount t to get a different starting point. $\Theta_A(s+t)$ represents a horizontally shifted turning function A. Then add θ in the form, which means A is rotated by angle θ to orient A in different angles. The turning function A is vertically shifted by θ. So far the updated distance calculates the difference between the shifted turning function A and the original turning function B according to a different starting point on A and a different orientation of A. Lastly, add the *min* operator to the form which means both starting point on A and orientation of A are varied continuously and at each step obtain the difference between the shifted turning function A and the original turning function B. The final distance is the minimum among all the obtained differences. As can be seen, the overall procedure is similar to the calculation of a Fréchet distance, both of which are trying to find a best boundary alignment and from there assign the difference of the boundary features as the mismatch between the two compared polygons. However, when calculating the difference of the boundary features, the $L2$ distance reflects all the boundary differences while Fréchet distance only reflect the worst boundary difference. Therefore, in some sense, $L2$ distance function is able to capture more feature differences than Fréchet distance function. Figure 5.4 shows the same polygon matchings illustrated in Figure 5.2. The shaded area represents the difference between the turning functions. The smaller shaded area for the matching between "Polygon 1" and "Polygon 3" indicates that "Polygon 1" is closer to "Polygon 3" than "Polygon 2"

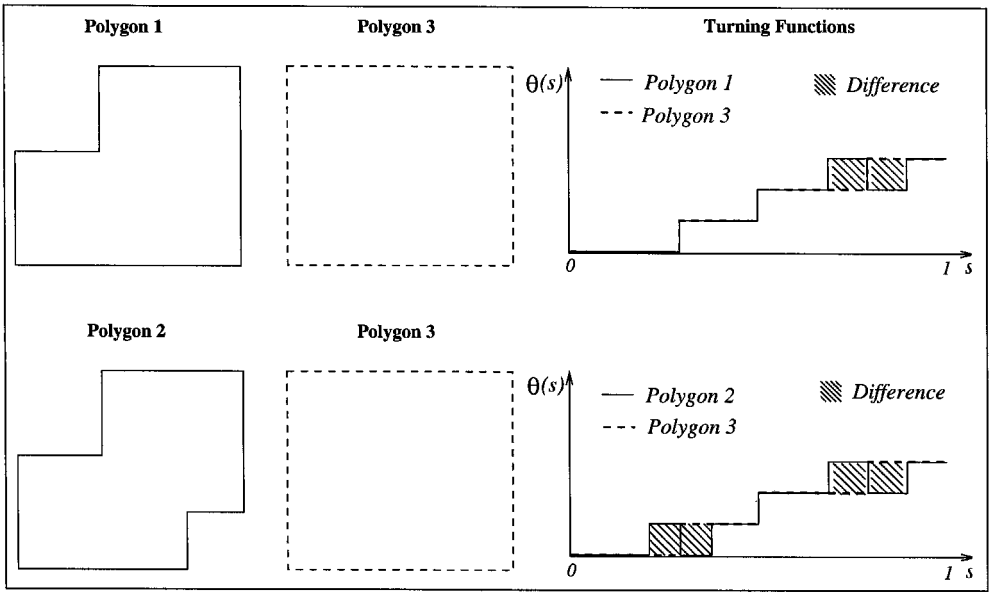


Figure 5.4: Polygon Matching Using L2 Distance Function.

is. It seems again that to vary all different values of t and θ requires a huge amount of computation. Arkin *et al.* [22], shows that assume A has m vertices and B has n vertices, the distance function always achieves its local minimum when t is at one of mn discrete vertices on $[0, 1]$. These are called critical events. In other words, during the horizontal shifting of turning function A, the distance values only need to be updated for the cases where any of the vertices of A meet any of the vertices of B. Also it is shown that the best θ value for a given t can be found in constant time. At this point, the runtime can be $O(mn(m+n))$. It is further shown that through some bookkeeping, except for the initial state, the distance value can be incrementally updated instead of entirely recalculated for every horizontal shifting of turning function A. Thus, the final runtime can be achieved as $O(mnlog(mn))$.

Finally consider the shape matching between any two mask-layouts. Since the absolute orientation of each mask edge affects the etching outcome, the edge orientations become part of the polygon features. Thus, the distance function does not need to be rotationally invariant. Furthermore, through some initial tests, it is observed that there is not much difference between the distance functions with different norms in our applications. So 1-norm distance function is used because of its simple calculation. With that, Equation (5.1)

can be simplified as follows:

$$d_1(A,B) = \min_{\Theta \in R\, t \in [0,1]} \int_0^1 |\Theta_A(s+t) - \Theta_B(s)| ds, \tag{5.2}$$

where

$$d_1(A,B) \quad = \quad \text{1-norm distance value.}$$

This shape matching function will later be used to measure the shape difference between any two mask-layouts, and also be used to measure the polygon layer matching between two 3D layer-represented shapes

Chapter 6

Mask-layout Synthesis and Implementation

6.1 Introduction

The goal of mask-layout synthesis is to generate the optimal mask layouts for a given MEMS fabrication process to produce a desired functional target shape. The approach taken here is to incorporate a forward fabrication simulator into a general evolutionary algorithm loop as shown in Figure 6.1 because reversing fabrication process simulators (so that a mask-layout might be produced) appears not to be possible, except in limited cases. An initial random population of mask-layouts is generated. The fabrication of each layout is simulated through a digital process simulator to produce a 3D fabricated shape, which is compared to a user-specified desired shape. Each evolutionary loop governs the stochastic searching behavior such that the mask-layouts whose simulated shapes are closer to the desired shape are more likely to survive. More importantly, the "better" masks are more likely to be evolved among those survived mask-layouts for the next iteration. Through such evolutionary iterations, a near global "optimum" mask-layout is likely to be found [35]. In general, such an approach involves relatively high computational cost due to the random learning process. It does, however, give a high level of modularity because the techniques of evolving mask-layouts can be separated from the details of any process simulation, in other words, evolutionary techniques can be combined with any existing simulation of fabrication processes to achieve mask-layout synthesis where reversing a fabrication process simulator (so that a mask-layout might be produced) appears not to be possible.

Even though other global searching algorithms can also fit into the above framework,

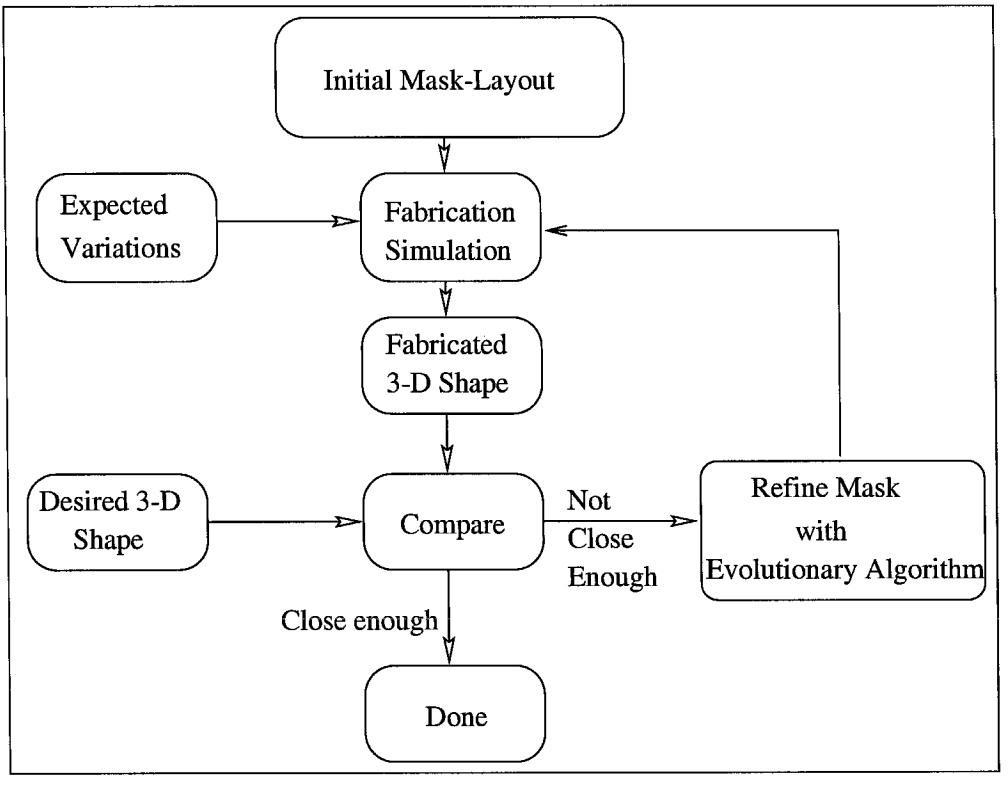


Figure 6.1: A General Approach to Realize Mask-layout Synthesis Using Evolutionary Algorithm.

as introduced in Chapter 3, evolutionary algorithms are particular good candidates to serve as the desired optimization technique here for two reasons. First, they are global stochastic optimization techniques with a high level of robustness. Based on the adaptive mechanics of natural genetics, an evolutionary algorithm searches global optimal region through the genetic evolution of a generation of individuals. According to "Implicit Parallelism" illustrated in section 3.6, such a pool of individuals brings a huge flow of solution information effectively processed along with the genetic operations on the individuals. It is such huge information flow that provides the algorithm with strong robust learning ability to overcome the deceptive traps and thus increase the chance to search the global optimum in various complex domains. Secondly, they are non-problem specific which means they can be virtually applied to any problem as long as its objective function measurements are available. So with the use of an evolutionary algorithm, the above-mentioned modularity can be practically achieved.

In our problem, mask-layouts are geometrically treated as 2D simple polygons (later referred as polygons), which form the underlying solution space. Therefore, in the view of an evolutionary algorithm, such solution space is the phenotype space. Each phenotype is a 2D polygon. The ultimate goal of finding an optimal mask-layout is thus directly translated into the seeking of an optimal 2D polygon in the entire 2D polygon space. It is assumed that the micro-fabrication outcome does not depend on where the input mask-layout is located on a substrate, e.g., a silicon wafer. This is mostly ensured by the homogeneity of the substrate properties and the mask-layout deposition environment. So the polygons to be explored here are translational invariant with no interest in the absolute locations of the polygon vertices. The interesting geometric features of each polygon can be separated into its size and shape information. The size is simply the value of the polygon length. The shape here is translational and size invariant but is critically dependent on rotational variant because each edge orientation affects the etching outcome. Each shape is reflected through a scaled polygon with each edge scaled by the original polygon size. Each scaled edge length is called edge distance ratio which directly indicates the distance percentage of the edge over the entire polygon size. Therefore, each polygon shape defines the edge boundaries of a polygon through the edge distance ratios and edge directional angles. The separation of the size and shape for any polygon further enables the entire searching for the optimal mask polygon to be split into two stages: first find the optimal polygon shape defined through a scaled polygon, and then find the optimal polygon size. The second stage can be simply carried through a greedy-based searching [1] in most applications, where the size of a fabricated shape is proportional to the size of the mask-layout. However, the search for an optimal polygon shape is rather challenging due to the richness of edge boundaries. Therefore, an evolutionary algorithm has been developed with the particular goal of effectively finding a global optimal polygon shape.

To complete an evolutionary algorithm, several components have to be provided. First, genetic operators including initializations, crossovers and mutations are needed to genetically evolve individuals. Second, a set of evolution strategies are used to control the entire evolution process. Third, the performance evaluation is necessary to give the performance for each individual. An algorithmic implementation (called OOGA) has been completed

with an object-oriented concept applied to all the above components. Each component is treated as an object and fully functions as an independent module. This chapter will cover the developed techniques in the first two modules, all of which are independent to specific applications and therefore can be reused.

6.2 Overall Architecture

An object-oriented software architecture has been created to implement such an evolutionary algorithm based framework. The framework mainly consists of three independent modules, namely: mask genetics module, evolutionary strategy module and MEMS simulation module. The mask genetics module provides heuristic genetic operators for mask-layouts, which include random mask generation, random crossovers and mutations. The evolutionary strategy module contains strategy routines to control the convergence of searching process such as stochastic selection schemes and genetic operations to balance the searching effort between exploration versus exploitation. The MEMS simulation module is the user input module which contains user specified MEMS fabrication simulations and the desired fabricated shape.

With the use of object abstraction, the development of the three modules becomes separated and thus achieves a high level of software modularity. Figure 6.2 shows the design of three layers of object inheritance. The base layer consists of two abstract object types called GENOTYPE and PHENOTYPE which mnemonically represent genotype and phenotype individuals respectively. GENOTYPE provides the interfaces of coding and genetic operators. PHENOTYPE provides the interface of performance evaluation. All these interfaces are independent to the underlying implementations and therefore, different sets of genetic operators associated with different coding schemes can be conveniently chosen. Different performance criteria can also be easily changed to be used for different applications with the same simulator. The second layer has the derived object types MASK GENOTYPE and MASK PHENOTYPE devoted to mask synthesis application. The geometry of mask-layouts is stored in this layer. The third layer has the final derived types such as REAL MASK GENOTYPE and SHAPE MATCH MASK PHENOTYPE. REAL

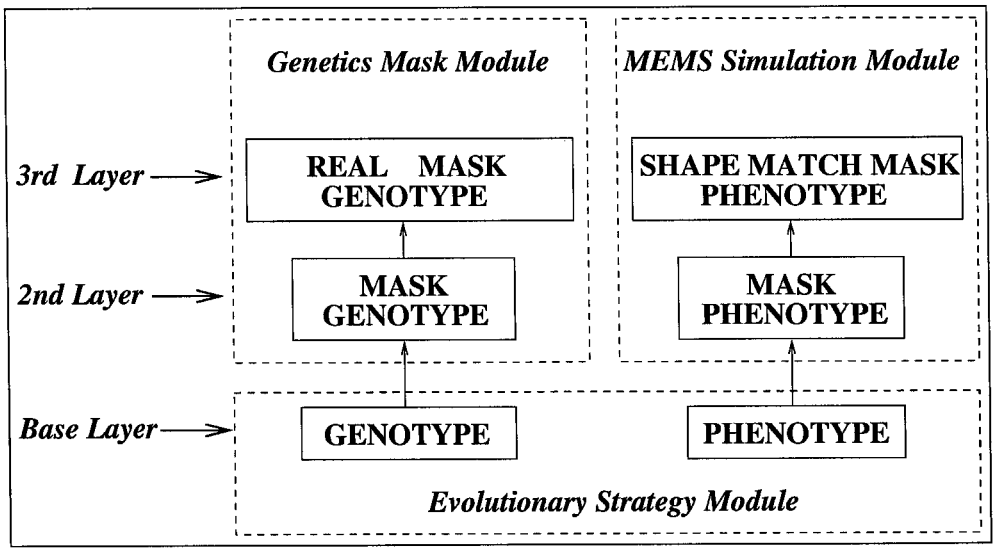


Figure 6.2: An Object-oriented Architecture On Evolutionary Algorithm.

MASK GENOTYPE refers to the MASK GENOTYPE with a real coding scheme of mask-layouts. It provides the implementations of the interfaces defined by GENOTYPE. SHAPE MATCH MASK PHENOTYPE fulfills the major task of evaluating the performance of the mask-layout through a specified process simulation and shape closeness measurements. Figure 6.2 also shows the relationship between the three modules and the three layers. The mask genetics module is built upon the base type MASK GENOTYPE and its derived types such as REAL MASK GENOTYPE; while the simulation module is developed based on MASK PHENOTYPE and its derived types such as SHAPE MATCH MASK PHENOTYPE. In this way, mask genetics module and simulation module are completely separated by the different inheritance chains. The evolutionary strategy module is entirely constructed from the base level objects GENOTYPE and PHENOTYPE and thus is insulated from the other two modules by the entire second inheritance layer.

6.3 Mask Genetics Module

6.3.1 Overview

The goal of this module is to design genetic operators that will provide the genetic manipulations on polygon shapes. The developed operators include initializations, crossovers and

mutations, all of which are the necessary components to implement a complete evolution. Particular knowledge from polygon shapes is used to provide important heuristics, which guide the constructions of these operators so that they can more efficiently and properly process individual features. As an example, one challenging constraint is from polygon simplicity which involves global feature detections. With the help of heuristics, such a constraint can effectively be overcome. Here, the individual features are the edge boundary features of each polygon shape. The design of each operator focuses on different ways to process such boundary features. The main purpose of the initializations is to create enough boundary features to serve as the major gene pool for future evolution. The challenge is how to generate uniformly distributed polygon shapes so as to minimize the initial bias towards any particular boundary features. Such fairness provides a "healthy" basis for crossovers to function properly. Crossovers provide the vital force to drive an evolution process towards the global optimal polygon shape. The entire search effort from crossovers is used to exploit existing popular boundary features and meanwhile explore new boundary features. The main challenge here is how to balance such exploitation and exploration so as to ensure a productive searching. Therefore, crossovers are responsible for properly growing boundary features. Mutations are designed to help crossover overcome deceptive traps that will induce improper exploitation towards a local optimal region. So the main goal of mutations is how to disrupt existent boundary features among the currently maintained individuals.

Before the design of any operators, a proper way to represent polygon shapes in genotype space is needed. Three popular coding schemes will be introduced. The comparison is made with the focus on which one gives the closest intuitive match with the solution features.

6.3.2 Mask Coding

The polygon shapes of mask-layouts are the solution points which serve as the phenotypes. To form the corresponding genotypes, we need an appropriate coding scheme which includes the structures of genetic strings and the associated semantics. In general, for each two dimensional polygon, two independent genetic strings are needed to specify two degrees of freedom. The two genetic strings will have the same size both being equal to the

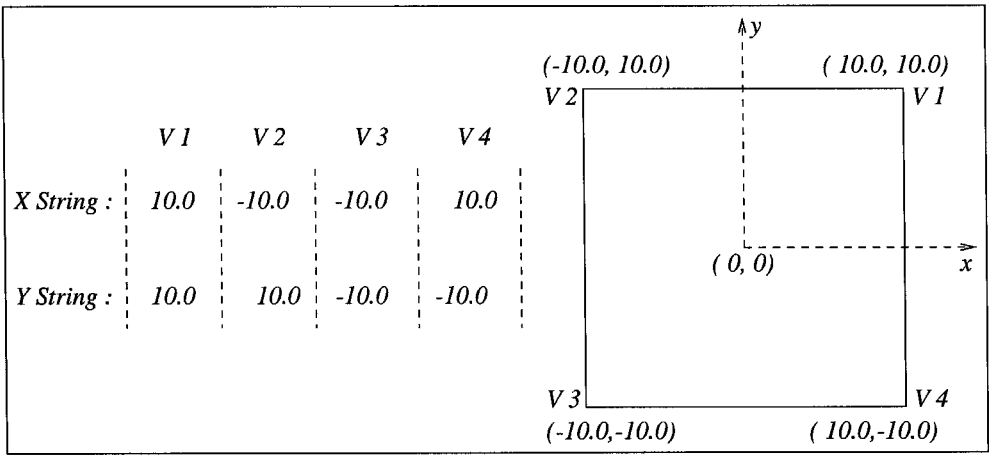


Figure 6.3: A Schematic Illustration Of XY Coding Scheme.

number of polygon sides or polygon vertices. A real coding scheme appears to be superior to a binary coding scheme because it provides adequate precision with a much shorter string length. The semantics of the coding scheme however, can be various due to the different ways to represent a polygon, some of which are described.

A polygon can be represented simply by specifying all its vertex locations in xy co-ordinates. The two genetic strings are used to store the x and y coordinate values of each vertex respectively. The string associated with the x coordinates is called "X String" and the other one is called "Y String." Two elements from each string with the same element position describe a polygon vertex. Such a coding scheme is called an xy-coding scheme, and is illustrated in Figure 6.3. Under such coding, searching for an optimal polygon shape is converted into searching of the optimal sequences of x and y coordinate values which indicate the locations of all the polygon vertices.

Another polygon representation can be constructed by specifying all vertex locations in polar coordinates. Each vertex has a corresponding polar ray from the origin to the vertex. The two genetic strings are used to store the length and directional angle of each polar ray respectively. The directional angles are measured from the positive x axis in counterclockwise direction and are always between 0 to 2π. The string associated with the ray length is called "RL String" and the one associated with the ray angle is called "RA String." Again, two elements from each string with the same element position describe a

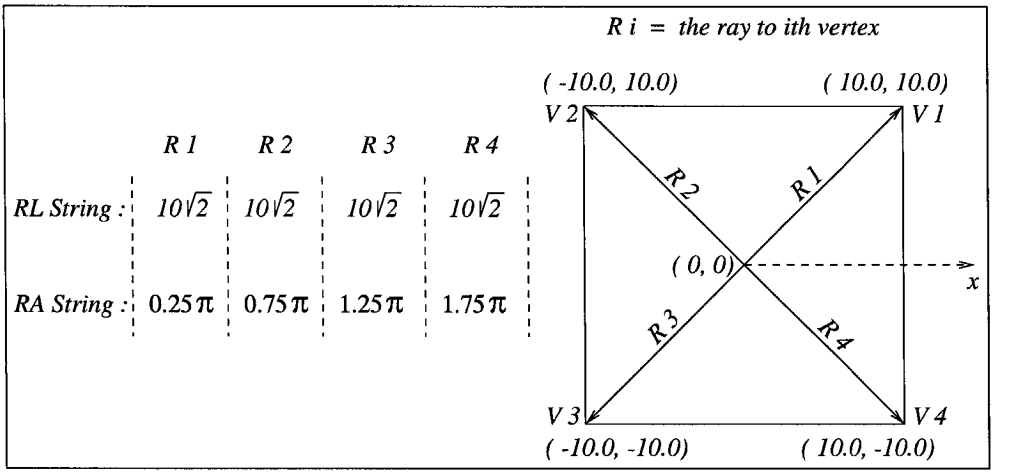


Figure 6.4: A Schematic Illustration Of Polar Coding Scheme.

polygon vertex. Such a coding scheme is called a polar-coding scheme, and is illustrated in Figure 6.4. Under such coding, searching for an optimal polygon shape is converted into searching of the optimal sequences of polar rays which also indicate the locations of all the polygon vertices.

A third way to represent a polygon is to capture the orientation and size information of each edge. The location of the first polygon vertex can be arbitrarily chosen since the vertex locations are not critical here. All the polygons must have a consistent winding direction, either clockwise or counterclockwise, so that the orientation of each edge can be determined. The two genetic strings are used to store the directional angles and the distance ratios of all the edges respectively. The directional angles are measured from the positive x axis in counterclockwise direction and are always between 0 to 2π. The corresponding storage string is called "Angle String." The distance ratios are obtained by scaling each edge length with a common factor, which is usually the polygon size. The associated storage string is called "Distance String." In this case, the two elements from each string with the same element position describe a polygon edge. Such a coding scheme is called an edge-coding scheme and is illustrated in Figure 6.5. The polygon shown is wound in a counterclockwise direction. Under such coding, searching for an optimal polygon shape is converted into searching of the optimal sequence of orientations and relative sizes of polygon edges.

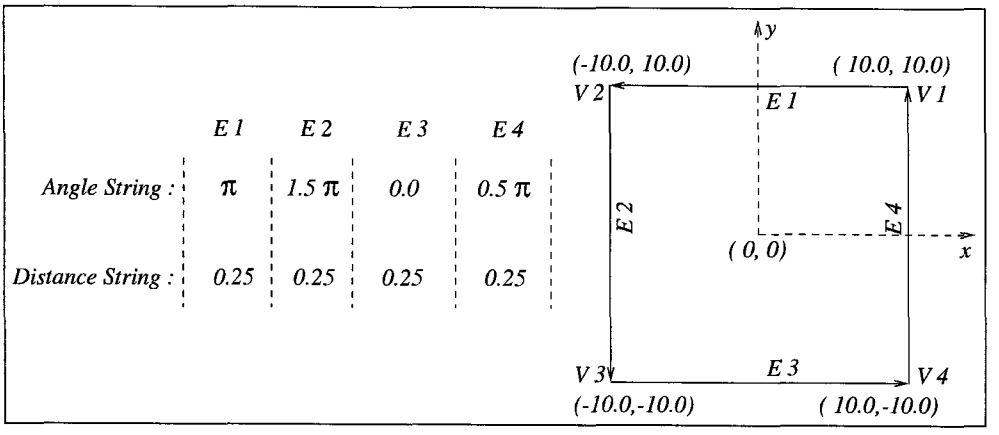


Figure 6.5: A Schematic Illustration Of Edge Coding Scheme.

The coding semantics illustrated above are common. The implementation of OOGA provides a convenient way for users to construct other semantics and attach them to the string structures. In general, as discussed in section 3.2, it is important to choose an appropriate coding scheme for the particular problem in two aspects: one is that the important solution features can be reflected through the coded strings in a rather simple and direct way, and the other is that the genetic operators can be conveniently designed to effectively propagate the solution features. The above three coding schemes can all serve as feasible ones. With the help of heuristics, to ensure the simplicity of the evolved polygons, the associated genetic operators such as crossover and mutation can be constructed easily. However, when considering how close each coding scheme is representationally related to the targeted solution, differences between the techniques become apparent. The solutions here are the shapes of polygons, which are captured through the distance ratios and the turning angles of polygon edges. Under an edge-coding scheme, the edge distance ratios are directly encoded in "Distance String"; the edge turning angles can also be directly obtained through the encoded directional angles in "Angle String" as shown in the following linear

relationship:

$$\begin{aligned} \Delta DA_i &= DA_i - DA_{i-1} \\ TA_i &= \begin{cases} -\Delta DA_i + 2\pi & -2\pi \leq \Delta DA_i < -\pi \\ \Delta DA_i & -\pi \leq \Delta DA_i < \pi \\ \Delta DA_i - 2\pi & \pi \leq \Delta DA_i < 2\pi \end{cases} \end{aligned} \tag{6.1}$$

where

$$\begin{aligned} DA_i &= \text{directional angle of edge i} \\ \Delta DA_i &= \text{directional angle difference of edge i and edge i-1} \\ TA_i &= \text{turning angle of edge i.} \end{aligned}$$

Therefore, under such coding scheme, there is a simple direct linkage between the evolution of the genetic strings in genotype space and the polygon shapes in phenotype space. Such a close correlation provides a convenient basis for the design of effective genetic operations as shown in the later sections. Under both an xy-coding scheme and a polar-coding scheme, however, the polygon shape is indirectly captured through polygon vertices, there is no simple relationship between the polygon vertices and the defined polygon shapes, Thus, how the searching in the space of polygon shapes is effectively carried out by the evolution of the polygon vertices is rather unclear. In fact, the later tests on the mask initialization shows the random distribution of polygon shape under an xy-coding scheme is relatively poor compared to those under an edge-coding scheme. Upon such considerations, all the currently developed implementations in OOGA are based on the edge-coding scheme.

6.3.3 Mask Initialization

The goal of the initialization process here is to randomly generate the first generation of polygon shapes in 2D polygon space. The uniform distribution of the polygon shapes is strongly desirable because it will bring the varieties of sample points that are needed for the future effective evolution. Based on the above-described coding structure, the whole ini-

tialization process starts from the random generation of all the individuals in the genotype space each of which consists of two genetic strings, and then, through the chosen coding semantics, each genotype is decoded into a phenotype which is a 2D polygon shape. Under the real coding scheme, all the string elements in the two genetic strings are real values within certain boundaries. Many random generators of real numbers are available to uniformly distribute each string element over its boundary. Therefore, the uniformity of the individual distribution in the genotype space can be practically obtained. Now the question is whether such uniformity can be ensured once each genotype is decoded into the corresponding phenotype which is a polygon shape. The answer will be affected by the coding semantics chosen as well as how to satisfy the polygon simplicity constraint. Especially due to the geometric constraint from polygon simplicity, the entire initialization process will be carried out in a heuristic manner. The outcomes of different initialization techniques will be further assessed through actual tests.

At this point, a method to measure the distribution of polygon shapes has to be constructed to assess the quality of any initialization technique. Since any polygon shape is defined through a combination of two independent variable sets which are edge distance ratios and edge directional angles, the distribution of polygon shapes can be fully reflected through the two separate distributions of those two variable sets. In each initialization process, the two variable distributions will be determined to assess the distribution of each polygon edge based on all the generated polygons. Several statistics variables are used to measure each distribution which includes average value, standard deviation and normalized χ^2-value through a χ^2-test. The χ^2-test essentially follows the steps described by Kreyszig [45]. Both valid ranges of edge directional angles and edge distance ratios are divided into intervals. For each polygon edge, among all the generated polygons, the observed frequency of each interval is recorded for directional angles which is called angle frequency and for distance ratios which is called distance frequency. The key formula used to calculate the normalized χ^2-value is the following:

$$\bar{\chi}^2 = \sum_{1 \le k \le I} \frac{(v_k - e_k)^2}{N * e_k}, \tag{6.2}$$

where

$$\begin{aligned} \bar{\chi}^2 &= \text{normalized value of } \chi^2 \\ I &= \text{total interval number} \\ N &= \text{total polygon number} \\ e_k &= \frac{N}{I}, \text{ expected frequency of interval } k \\ v_k &= \text{observed frequency of interval } k. \end{aligned}$$

Since the regular χ^2-value is sensitive to the sample size, the sample size needs to be fixed throughout the tests for consistent comparisons. To further simplify the results, the normalized χ^2-value is used which is the regular χ^2-value divided by the sample size as shown in (6.2). The difference between standard deviation and $\bar{\chi}^2$-value should be appreciated. Standard deviation measures the spread of values themselves while $\bar{\chi}^2$-value measures the spread of value occurrences. A large standard deviation may well indicate a wide spread of values, which is a good thing here. However only combined with a small $\bar{\chi}^2$-value, can the spread be expected to have good uniformity. With both large standard deviation and $\bar{\chi}^2$-value, the distribution is widely spread but skewed towards either the small value region or the large value region. So both quantities are needed to describe the whole picture of a distribution.

Consider the edge directional angles, the valid range is from 0 to 2π. The whole range is divided into 36 intervals. Since the directional angles of different edges are expected to be independent upon each other, the expected average value of directional angles is the average of valid value range which is π, and the expected angle frequency of each interval is the total generated polygon number (generation size) divided by 36. The distribution of the edge distance ratios is more complicated because the distance ratios of different edges are dependent upon each other. To simplify the measurement, all the generated polygons are scaled to have the same size equal to 100. Thus, each edge length directly indicates the percentage value of the edge distance ratio. The valid range is from 0 to 100. The upper bound can be further narrowed down to 50 because any edge length represents the

shortest distance between its two vertices and as part of a closed polygon, its length can never exceed the half of the polygon length. The whole valid range of edge distance ratios is divided into 20 intervals from 0 to 50. Since the sum of all the edge distance ratios is fixed at 100 and there is no bias on any particular edge, the expected average value of edge distance ratio shall be 100 divided by the number of edges. For each edge, however, the expected distribution is no longer uniform. Every occurrence of large distance ratio for one edge will lead to the small distance ratios for all other edges. So in general, the observed frequency will be higher in the intervals corresponding to smaller values or in other words, the distributions will bias towards relatively small value regions. In fact, to achieve wide spread of edge distance ratios is difficult and plays a crucial factor to ensure a rich set of the sampled polygon shapes.

It is rather challenging to satisfy the polygon simplicity constraint. In general, there are two strategies to deal with it. One is to ignore such constraint during the generation process and check the simplicity of all the decoded polygons and punish the non-simple (e.g., self intersecting) ones as invalid solutions. However, this strategy usually causes too many invalid solutions among the initial generation. Under the initial condition with only few valid individuals, the further evolution is almost impossible to be effectively carried on. Therefore, the second strategy becomes the feasible one here, which is to ensure the simplicity of each polygon during the process of random generation. This is essentially the problem of how to generate random uniform simple polygons. From a theoretical point of view, no polynomial-time solution is known to solve such a problem [15]. Several heuristic methods have been proposed, which can be categorized into two general schemes. One is to generate a polygon among a set of fixed vertices; the other is to generate a polygon with its vertices moving around. One available method under the first scheme and two new methods belonging to the second scheme are presented below and the performances of the initializations using those methods are compared through statistic tests.

Most currently available methods fall into the first scheme which are summarized by T. Auer and M. Held [3]. All the methods start by randomly generating a specified number of vertices with a uniform distribution in a specified region. Then a simple polygon is constructed through these vertices in a heuristic random manner. Ideally, pure uniformity

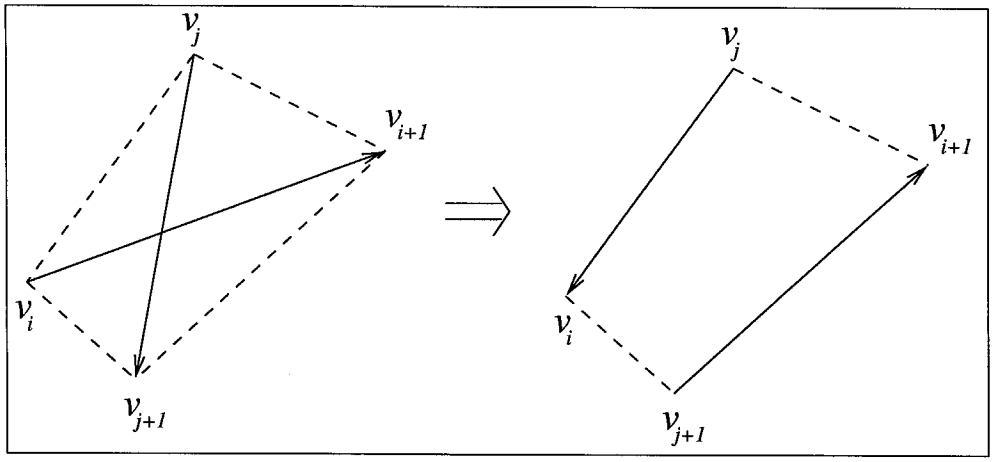


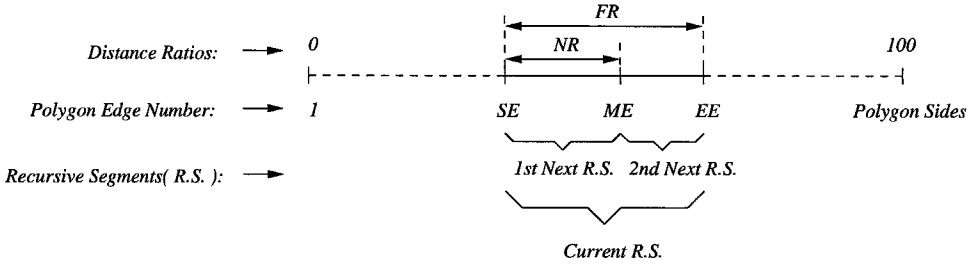
Figure 6.6: A Schematic Illustration Of A 2-opt Move.

can be achieved by generating all the possible simple polygons through these uniformly distributed vertices and randomly picking one. It is, however, still an open question to ask how many simple polygons there are with a given set of vertices. Nevertheless, the above approaches provide feasible ways to randomly generate simple polygons in an effective heuristic manner. As an example, one method constructed by Zhu *et al.* [15] is called "2-opt Moves". To generate an n-sided polygon, this method first generates n points uniformly distributed within a specified boundary region. The n points are then randomly permuted in an initial sequence to form the initial polygon P. Any self-intersections of P are removed by applying so-called 2-opt moves as illustrated in Figure 6.6. Each 2-opt move replaces a pair of intersecting edges (v_i, v_{i+1}), (v_j, v_{j+1}) with the edges (v_{j+1}, v_{i+1}) and (v_j, v_i). Through the actual testing, the "2-opt Moves" method is shown to be able to generate a rich set of simple polygons [3]. If the initialization process incorporates any of the above heuristic methods such as "2-opt Moves" method to overcome the self-intersection constraint, the xy-coding scheme will be the best fit because the uniform distributions of the string elements in "X String" and "Y String" can directly ensure the same uniformity of the correspondent vertex distributions. However, the only concern left is again that the connection between the vertex distribution and the polygon shape distribution is not quite clear.

Two new methods are constructed here to directly control the distribution of polygon

shapes during random generation. Both methods focus on the random generation of polygon edges instead of vertices. In the first method, a heuristic self-checking is used to incrementally obtain each valid edge, whose edge directional angle and edge length are repeatedly generated until no intersection exists between the edge and any previously generated ones. The random values of both edge directional angles and edge lengths are uniformly generated within their valid ranges. However, because of the polygon simplicity constraint, only the first edge is expected to have uniform distributions of both directional angles and lengths. As more each new edge is generated, the distribution of the current edge will be more affected by the restriction of avoiding edge intersections. Naturally, control over the last edge is completely lost due to closing the polygon. Thus, the distributions of different edges are expected to have uneven uniformities with bias towards the starting edges. To correct this bias, the first edge of each polygon is randomly selected among all the generated edges. The distance ratio of each edge will be the edge length divided by the size of the generated polygon. However, it is difficult for the edge distance ratios obtained in this way to have large values. In order for an edge to have a large distance ratio, its edge length must be large, and the remaining edge lengths must be small. This is almost impossible to occur since the generation of edge lengths are independent random processes.

To increase the chance of obtaining large edge distance ratios, a second method is constructed. The distance ratios of all the edges are directly generated first. Then, each edge is incrementally obtained by repeatedly generating the edge directional angle until no intersection exists between the edge and any previous ones. Figure 6.7 shows the pseudocode for the direct generation of edge distance ratios. At each recursive step, both a distance ratio range FR and a portion of the edges with edge number from SE to EE available. FR is randomly subdivided into two subranges NR and $FR - NR$; the whole portion of the edges is also randomly split into two subportions from SE to ME and from ME to EE. Then, NR is mapped to the portion of the edges from SE to ME, and $FR - NR$ is assigned to the remaining portion of the edges. Both of these portions will continue to be subdivided through subsequent recursive steps until one distance ratio range is mapped to each edge. The initial distance ratio is 100 and the edge number is from 1 to the number of polygon sides. The key here is the random mapping between any value of distance ratio



Variables:

SE = Starting Edge Number of Current Recursive Segment

EE = End Edge Number of Current Recursive Segment

ME = Middle Edge Number of Current Recursive Segment

FR = Distance Ratio Range of Current RecursiveSegment

NR = Distance Ratio Range of 1st Next Recursive Segment

RandomDistanceRatios(**int** SE, **int** EE, **float** FR, **float*&** DistanceRatios)

```
1  if ( EE − SE == 1 )
2      Distance Ratio of Edge SE ← FR;
3      return;
4  int ME; float NR;
5  ME ← A Random Integer Value Within (SE, EE);
6  NR ← A Random Float Value Within (0.0, FR);
7  RandomDistanceRatios(SE, ME, NR, DistanceRatios);
8  RandomDistanceRatios(ME, EE, FR − NR, DistanceRatios).
```

Figure 6.7: Pseudocode for Random Generation of Edge Distance Ratios under "Direct-ratio Edge Increment" Method.

range and any portion of the edges. So it is possible for one edge to have large distance ratio once a large distance ratio range is mapped to a small portion of the edges. The main difference between the above two methods is the way edge distance ratios are generated. The first method is called "Indirect-ratio Edge Increment" and the second one is called "Direct-ratio Edge Increment." Both of the two methods directly determine the directional angles and distance ratios of polygon edges through random generation.

If an initialization process uses either "Indirect-ratio Edge Increment" or "Direct-ratio Edge Increment" to generate polygon shapes, the edge-coding scheme will be the best fit because its "Angle String" and "Distance String" directly capture the directional angle and distance ratio of each edge.

Tests have been carried out to evaluate the performance of three different initialization techniques. The first one named as "XY-2opt" is based on the xy-coding scheme using "2-opt Moves" to generate simple polygons; the second one named as "Edge Indirect" is based on the edge-coding scheme with "Indirect-ratio Edge Increment" as the polygon generation method; the third one named as "Edge Direct" is also based on the edge-coding scheme but polygons are generated under "Direct-ratio Edge Increment" method. The package "rpg" is used to implement the "2-opt Moves" method [4]. All the test outcomes presented are based on the random generation of $10,000$ polygons with the side number of 5, 18 and 30. For each side number, five edges are chosen and for each edge, the general statistic measurements on the distributions of its directional angles and its distance ratios are presented. In addition, due to the particular challenge of obtaining a good distribution of edge distance ratios, the distributions of edge distance ratios are plotted for the three initialization techniques, which give schematic comparisons on the distribution uniformities. All the distribution plots are obtained through four steps: 1) divide the entire distance ratio range of 50 into 100 intervals; 2) measure the distance frequency of each interval for all the edges; 3) obtain the average distance frequency for each interval by averaging the corresponding distance frequencies of all the edges; 4) plot all the average distance frequencies against each interval. Therefore, each distribution curve exactly shows how the total 10,000 generations of a typical edge distance ratio are distributed into the 100 intervals.

The test results are presented according to the different polygon side number. Table 6.1

and Figure 6.8 are corresponding to 5-side polygons; Table 6.2 and Figure 6.9 are associated with 18-side polygons; Table 6.3 and Figure 6.10 belong to 30-side polygons. Among the results, some common observations can be drawn. First, all the observed average directional angles and distance ratios listed in the tables are fairly close to the expected values. Most average directional angles vary from 3.11 to 3.16 which are around the expected value of π. The observed average distance ratios for 5-side, 18-side, 30-side polygons are between the narrow variations from 19.7 to 20.2, from 3.3 to 3.4 and from 5.4 to 5.6 respectively, all of which are quite close to the expected values of 20, 5.5 and 3.3 correspondingly. Second, under each initialization technique, for each statistic measurement, the observed values in the tables are fairly close among the different chosen edges. For example, in Table 6.1, consider the values of standard deviations measured for "XY-2opt," the variation for the directional angles among the five edges is from 1.803 to 1.827 and that for the distance ratios is from 9.496 to 9.699, both of which are small range of differences. Such small statistic variations among different edges indicate that there is no intrinsic bias towards any particular edges during the polygon generation. Third, the directional angle distributions are much better than the distance ratio distributions. The observed $\bar{\chi}^2$-values of directional angle distributions in the tables are all much smaller compared to those of the distance ratio distributions, which indicates that the directional angle distributions have a much better uniformity. In addition, as the number of the polygon sides increases, neither standard deviations nor $\bar{\chi}^2$-values of directional angle distributions varies much except the $\bar{\chi}^2$-values under "XY-2opt" which will be explained later. For example, all the standard deviations of directional angles are almost fixed around 1.8 even as the polygon side number changes among 5, 18 and 30. Such small variations mean that equally good directional angle distributions can be practically obtained even for the polygons with large number of sides. On the other hand, both standard deviations and $\bar{\chi}^2$-values of distance ratio distributions degrade as the number of polygon sides goes up. For an example, under "XY-2opt", as the number of polygon sides increases from 5 to 30, the standard deviations decreases from around 9.5 to about 2.1 while the $\bar{\chi}^2$-values increases from around 0.4 to about 5.8, which means the distribution spread becomes narrower and has lower level of uniformity. Therefore, overall, the major concern is not directional angle distributions but rather the

distance ratio distributions. Fourth, the distance ratio distribution under "Edge Indirect" is consistently better than the one under "XY-2opt." Throughout the tables, as compared to "XY-2opt," "Edge Indirect" has higher values for all the standard deviations and lower values for all the $\bar{\chi}^2$-values, which means its distribution spread is wider and more uniform. The comparisons on the quality of the spread between the two are also illustrated in the three plots. The curve of "Edge Indirect" is flatter and wider than that of "XY-2opt" which is more obvious as the polygon side number increases to 18 and 30 shown in Figure 6.9 and Figure 6.10. Also, the "Edge Indirect" curve is consistently higher than the "XY-2opt" curve in both the small value and large value regions of distance ratios. The outperforming "Edge Indirect" indicates that by directly generating the edges instead of vertex locations, the overall shape distributions improve in terms of the better distributed edge distance ratios. Fifth, from all the three plots, the "Edge Direct" curve consistently outperforms the other two techniques in both the small value and large value regions of distance ratios, which means that using "Direct-ratio Edge Increment" method does increase the chance to generate extreme distance ratios especially the large value ones.

Among the different tables and figures, there are also some different comparison results. From Table 6.1, there is not much difference between the directional angle distributions among the three initialization techniques. The distance ratio distributions however does have clear difference. The distributions under "Edge Direct" is the best which has lowest $\bar{\chi}^2$-values and the highest standard deviations. Especially, the $\bar{\chi}^2$-values of "Edge Direct" is around 0.16 much lower than the other two whose values are around 0.4. Figure 6.8 further demonstrates the wide spread of the "Edge Direct" curve. Especially in the small value and large value regions of distance ratios, "Edge Direct" curve gives a big improvement. From both Table 6.2 and Table 6.3, as indicated previously, the directional angle distributions under "XY-2opt" degrade as the polygon side number increases from 5 to 18 and 30. The $\bar{\chi}^2$-values increases roughly from 0.006 to 0.027. This again shows the worse performance of "XY-2opt" which uses random vertex locations to control the distributions of polygon shapes. Additionally, it becomes hard to judge the overall quality of the distance ratio distribution for "Edge Direct" technique when the polygon side number increases to 18 and 30. In both tables, the standard deviations are still the highest which indicates a wide spread

of the distribution. But the $\bar{\chi}^2$-values are also the highest which means a poor uniformity. The distribution spreads are illustrated in both Figure 6.9 and Figure 6.10. Even though there are occurrences of large distance ratios for "Edge Direct" curve, a large number of occurrences have very small distance ratios and thus, in the average value region, the "Edge Direct" curve is worse than both "Edge Indirect" and "XY-2opt." Such uneven distribution demonstrates the poor uniformity. Again, this is because as the number of polygon sides increases, every single occurrence of large distance ratio for one edge directly causes more of the rest of edges to fall into the small value regions. Such self-squeezing effect simply can not be avoided.

The computation cost (execution time) for all the three initialization techniques to implement the initializations for 5-side, 18-side and 30-side polygons are listed in Table 6.4. "Edge Direct" is the most efficient among the three. This can be explained by the fact that most generated edge lengths are small values and it is relatively easy for the edges with small length to avoid self-intersections over a range of different directional angles. Therefore, with fewer occurrences of self-intersections, the construction of each random polygon is much faster. "Edge Indirect," however, is overall the worst even though it is faster than "XY-2opt" for the polygons with small number of sides. The bad efficiency for "Edge Indirect" is because each time when generating an edge, once self-intersection occurs, the current edge is randomly regenerated without learning, so the probability of causing self-intersection is always the same for each regeneration. To make it worse, such probability will increase as the number of edges generated increases. So the self-intersection occurs frequently during each construction for polygons with large number of sides. "XY-2opt" is better because once a permutation of vertices is chosen the internal number of self-intersections is fixed. The further construction can only decrease the self-intersections. This scheme is stable for large numbers of sides.

In summary, based on the above comparisons, it can be seen that, even though it seems easy for an initialization process to incorporate several available methods (such as "2-opt Moves") to randomly generate simple polygons based on randomly distributed vertices, the test results on the performance of "XY-2opt" shows that such an initialization technique does not give as good polygon shape distributions as "Edge Indirect," which directly obtains

Initialization Techniques	Edge No.	Directional Angle			Distance Ratio		
		Average Value	Standard Deviation	$\bar{\chi}^2$-value	Average Value	Standard Deviation	$\bar{\chi}^2$-value
XY-coding & 2-opt Moves	1	3.134	1.818	0.007	19.995	9.547	0.421
	2	3.160	1.818	0.006	19.964	9.551	0.418
	3	3.124	1.821	0.006	19.964	9.496	0.431
	4	3.174	1.803	0.005	20.047	9.699	0.399
	5	3.126	1.827	0.006	20.069	9.588	0.418
Edge-coding & Indirect-ratio Edge Increment	1	3.090	1.806	0.004	20.051	10.191	0.361
	2	3.118	1.810	0.005	20.057	10.067	0.368
	3	3.157	1.811	0.004	19.990	10.124	0.357
	4	3.147	1.827	0.004	20.048	10.178	0.370
	5	3.155	1.821	0.005	19.852	10.039	0.372
Edge-coding & Direct-ratio Edge Increment	1	3.143	1.827	0.004	20.250	12.431	0.160
	2	3.144	1.798	0.003	19.934	12.326	0.164
	3	3.157	1.818	0.002	19.829	12.485	0.157
	4	3.159	1.811	0.004	19.792	12.398	0.164
	5	3.164	1.823	0.003	20.192	12.425	0.157

Table 6.1: General Statistic Measurement on the Edge Distance Ratios and Edge Directional Angles of 10,000 5-Side Simple Polygons Generated under "XY-2opt", "Edge Indirect" and "Edge Direct."

Initialization Techniques	Edge No.	Directional Angle			Distance Ratio		
		Average Value	Standard Deviation	$\bar{\chi}^2$-value	Average Value	Standard Deviation	$\bar{\chi}^2$-value
XY-coding & 2-opt Moves	1	3.160	1.819	0.020	5.517	3.256	3.159
	6	3.148	1.821	0.025	5.576	3.304	3.135
	10	3.122	1.833	0.018	5.613	3.301	3.113
	14	3.154	1.826	0.019	5.548	3.290	3.129
	18	3.100	1.831	0.016	5.510	3.253	3.172
Edge-coding & Indirect-ratio Edge Increment	1	3.130	1.811	0.003	5.550	3.441	3.069
	6	3.152	1.828	0.002	5.554	3.441	3.063
	10	3.148	1.805	0.003	5.558	3.478	3.061
	14	3.130	1.817	0.002	5.556	3.483	3.060
	18	3.105	1.807	0.003	5.609	3.454	3.051
Edge-coding & Direct-ratio Edge Increment	1	3.137	1.816	0.003	5.570	9.231	4.309
	6	3.157	1.815	0.003	5.607	9.157	4.242
	10	3.139	1.824	0.003	5.607	9.122	4.229
	14	3.146	1.812	0.002	5.451	9.086	4.380
	18	3.124	1.817	0.002	5.428	8.966	4.343

Table 6.2: General Statistic Measurement on the Edge Distance Ratios and Edge Directional Angles of 10,000 18-Side Simple Polygons Generated under "XY-2opt," "Edge Indirect" and "Edge Direct."

Initialization Techniques	Edge No.	Directional Angle			Distance Ratio		
		Average Value	Standard Deviation	$\bar{\chi}^2$-value	Average Value	Standard Deviation	$\bar{\chi}^2$-value
XY-coding & 2-opt Moves	1	3.131	1.822	0.025	3.311	2.091	5.863
	8	3.157	1.822	0.029	3.333	2.101	5.819
	15	3.138	1.823	0.027	3.317	2.080	5.827
	22	3.154	1.817	0.027	3.309	2.099	5.880
	30	3.142	1.818	0.021	3.314	2.083	5.820
Edge-coding & Indirect-ratio Edge Increment	1	3.167	1.811	0.002	3.304	2.171	5.220
	8	3.142	1.822	0.003	3.361	2.171	5.151
	15	3.145	1.801	0.003	3.331	2.188	5.183
	22	3.154	1.819	0.002	3.341	2.171	5.217
	30	3.139	1.813	0.003	3.300	2.173	5.227
Edge-coding & Direct-ratio Edge Increment	1	3.164	1.808	0.004	3.325	7.127	5.959
	8	3.117	1.806	0.003	3.304	7.040	5.924
	15	3.113	1.816	0.003	3.431	7.254	5.836
	22	3.166	1.8111	0.004	3.352	7.204	5.963
	30	3.146	1.820	0.002	3.279	7.098	5.997

Table 6.3: General Statistic Measurement on the Edge Distance Ratios and Edge Directional Angles of 10,000 30-side Simple Polygons Generated under "XY-2opt," "Edge Indirect" and "Edge Direct."

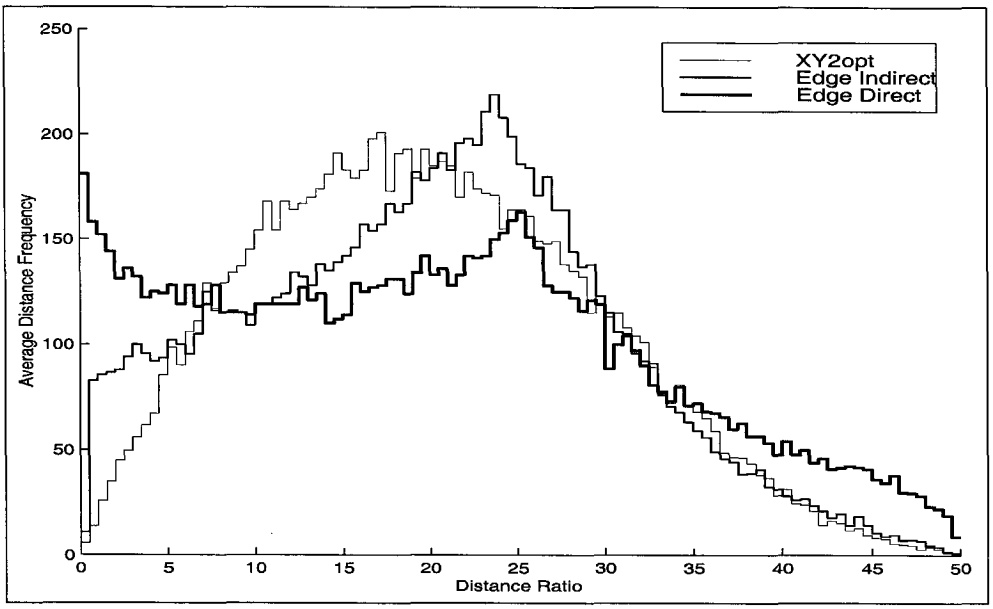


Figure 6.8: Edge Distance Distributions of 10,000 5-Side Simple Polygons Generated under "XY-2opt," "Edge Indirect" and "Edge Direct."

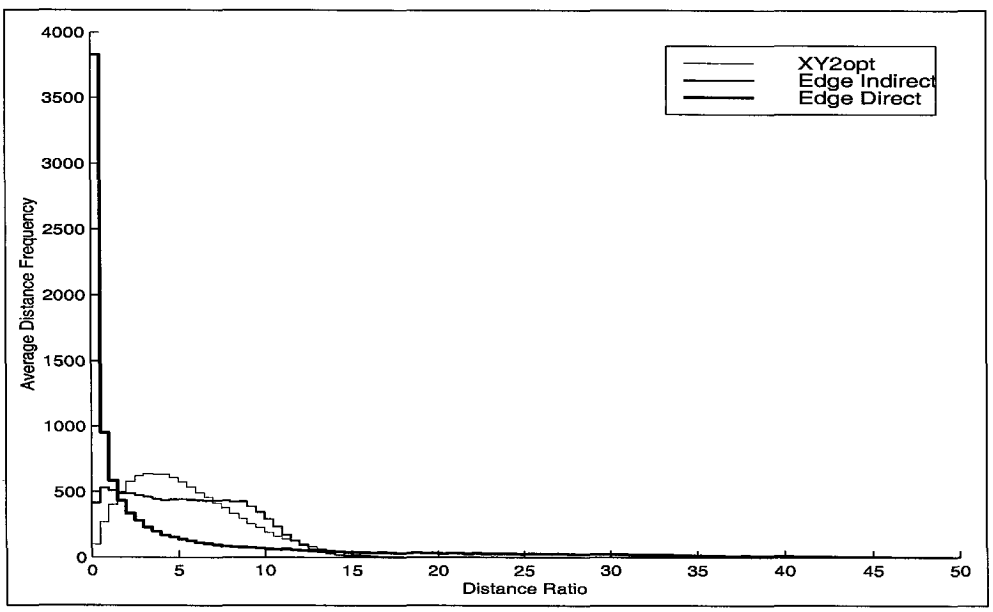


Figure 6.9: Edge Distance Distributions of 10,000 18-Side Simple Polygons Generated under "XY-2opt," "Edge Indirect" and "Edge Direct."

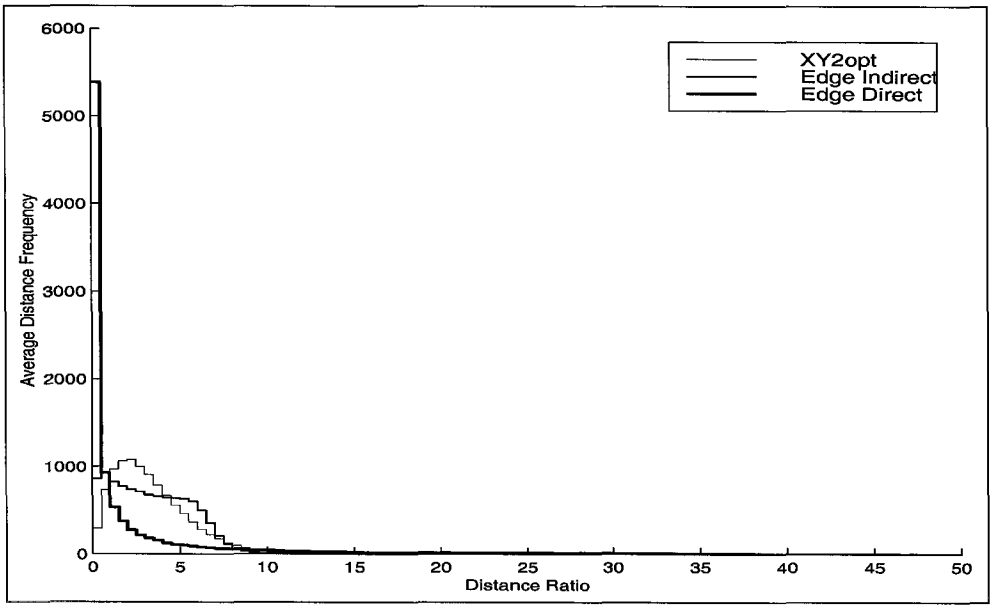


Figure 6.10: Edge Distance Distributions of 10,000 30-Side Simple Polygons Generated under "XY-2opt," "Edge Indirect" and "Edge Direct."

each polygon through the random generation of polygon edges. By using the "Direct-ratio Edge Increment" method, large edge distance ratios can be obtained with a greater chance. Especially for polygons with a small number of sides, "Edge Direct" initialization is the best choice both in terms of the expected quality of the distributions and the efficiency. However, poor uniformity of the distance ratio distributions for polygons with a large number of sides suggests that it should only be used when the optimal shapes are expected to have some dominant edges with very large edge lengths. "Edge Indirect" is the best choice among the three in most applications where the optimal shapes are expected to have non-dominant edge lengths. It provides the best overall spread of the distance ratio distributions around the average-valued region. Although the high computation cost is a concern, when considered over the entire evolution process, good initialized samples are much more important. In situations where the optimal shapes are completely unknown, the combinations of "Edge Indirect" and "Edge Direct" should be used to achieve a good distribution of polygon shapes.

Initialization Techniques	5 Sides (sec)	18 Sides (sec)	30 Sides (sec)
XY-2opt	9.27	116.46	361.02
Edge Indirect	5.05	224.7	663.6
Edge Direct	4.71	75.9	213.6

Table 6.4: CPU Time for Sun Ultra-1 to Randomly Generate 5-Sided, 18-Sided and 30-Sided 10,000 Simple Polygons under "XY-2opt," "Edge Indirect" and "Edge Direct."

6.3.4 Mask Crossover

As described in section 3.4, a crossover serves as the vital force for an evolutionary algorithm to search for an optimal solution. A crossover is directly applied to genotypes. With a proper coding scheme, genotypes are usually convenient for the design of the crossover. Even though a crossover is implemented in the genotype space, the ultimate goal is to give a balanced searching effort between exploration versus exploitation in the underlying phenotype or solution space. Exploitation refers to the effort towards narrowing the existent small differences, while the exploration refers to the effort towards aggressively trying alternative features. In our problem, the solution space consists of all 2D polygon shapes all of which are encoded into two genetic strings in the genotype space. Every crossover between the real-coded strings corresponds to an underlying crossover between polygon shapes. The particular goal of the crossover here is to effectively exploit and explore such solution space from a given generation of polygon shapes. The particular meaning of the balance between such exploitation and exploration is further illustrated through Figure 6.11. Comparing the edge boundaries of the two polygons, the first four edges marked with dark outlines are rather close to each other while edge $E5$ and edge $E6$ are quite different. Through the crossover, the generated children polygons should inherit the common edge boundary features of these two parent polygons. So the first four edges of all the children polygons should be very close to their parents as a family feature, while the individual distinct feature is carried through the variations in the last two edges. In general, between any

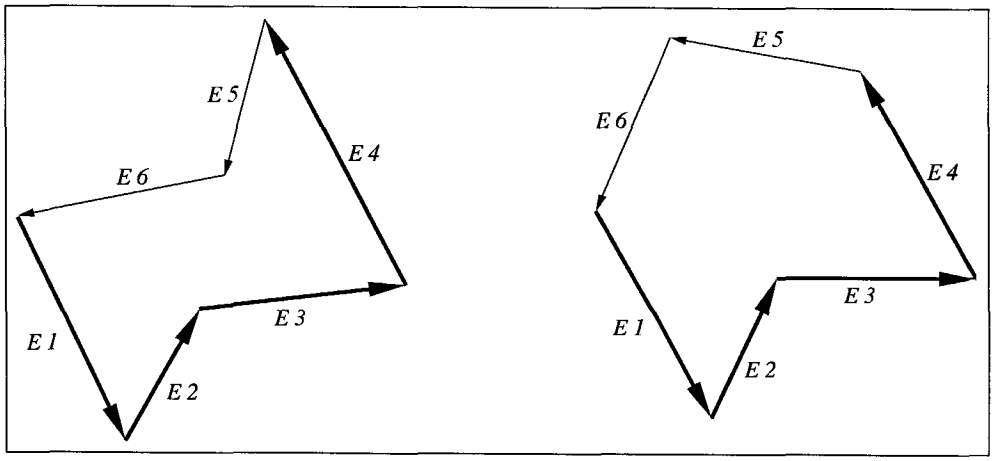


Figure 6.11: Boundary Comparison Between Two Polygons.

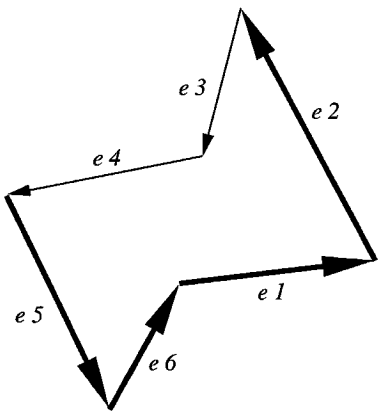
two polygons, some portions of their boundaries may have a closer match than others. The crossover should be able to differentiate them and treat them differently. To do that, the closely matched portions should be exploited while the largely mismatched portions need to be explored. The exploitation refers to the effort towards narrowing the existent small differences, while the exploration refers to the effort towards aggressively trying alternative features.

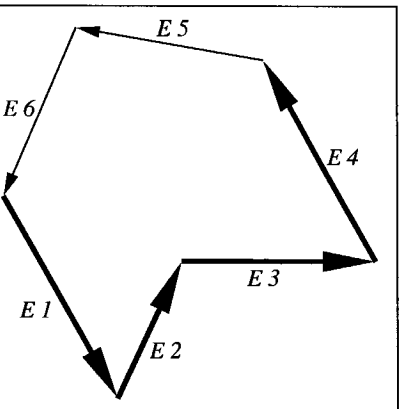
A coding scheme has to be chosen in order to start the design of the crossover. The edge-coding scheme illustrated in section 6.3.2 gives a convenient way for the design of a crossover to directly process the edge boundaries of polygons. Under the edge-coding scheme, each genotype consists of two real strings. As shown in Figure 6.5, the "Angle String" is used to represent the directional angles of edges and the "Distance String" is used to store the distance ratios of edges. Since each of the two strings physically covers one degree of freedom for the representation of a 2D polygon, they are independent of each other and need to be processed separately during each crossover. Each time, a crossover is applied to two parent genotypes, during which a pair of "Angle String"s and a pair of "Distance String"s are formed separately. Crossover is implemented onto the two string pairs independently. Within each string pair, the crossover operator is applied to paired real-valued elements which corresponds to the edge directional angles for "Angle String" pair and the edge distance ratios for "Distance String" pair. In this way, the entire crossover

on genotypes directly controls the crossover on the edge boundaries of polygons, which is further realized through two separate crossovers of the edge directional angles and the edge distance ratios.

Several available crossover operators for real-coded strings were introduced in section 3.4. The blend crossover BLX-α, with α value set as 0.5, is used here. As shown in Figure 3.3, this operator uniformly picks values between two points that contain the two parents $p1$ and $p2$, but may extend equally on either side, determined by a user specified parameter α. With the α value set as 0.5, the extended interval on either side is 50 percent of the interval between the two parent values. The key feature here is that the crossover search range is linearly proportional to the difference in value between the parents. Under the edge-coding scheme, BLX-0.5 is applied to all the paired elements in both the "Angle String" pair and the "Distance String" pair. And in addition, the crossovers on different paired elements are independent to each other. Since the parent element values directly represent the edge boundaries, the difference in the parent values indicates the gap between the edge boundaries. By using BLX-0.5, the crossover search range becomes narrower wherever the matched edge boundaries are closer, and becomes wider whenever the matched edge boundaries are apart. The narrow search range leads to the children values being close to the parent values, which coincides the exploitation, while the wide search range will accordingly lead to exploration. Therefore, an effective way to balance the exploitation and exploration from the crossover can be achieved through the use of BLX-0.5 under the edge-coding scheme.

Similar to initialization, the constraint of polygon simplicity has to be satisfied for all new polygons generated through crossover. Again, an effective way to meet such a constraint is to ensure the simplicity of each new polygon during each crossover. Similar to the "Indirect-ratio Edge Increment" method, each new edge is incrementally generated through crossover, and the intersections are checked against all the previous edges. The process is repeated for each edge until no self-intersection exists. With the choice of coding scheme and the crossover operator plus constraint satisfaction, the entire crossover process can be fully carried out. However, with the use of the following heuristic techniques, the overall crossover can be more properly and effectively implemented.





Aei -- Directional Angle of Edge ei

Dei -- Distance Raio of Edge ei

A_{Ei} -- *Directional Angle of Edge Ei*

D_{Ei} -- *Distance Ratio of Edge Ei*

Before Alignment

Angle String Pair:	*Ae1*	*Ae2*	*Ae3*	*Ae4*	*Ae5*	*Ae6*
	A_{E1}	A_{E2}	A_{E3}	A_{E4}	A_{E5}	A_{E6}
Distance String Pair:	*De1*	*De2*	*De3*	*De4*	*De5*	*De6*
	D_{E1}	D_{E2}	D_{E3}	D_{E4}	D_{E5}	D_{E6}

After Alignment

Angle String Pair:	*Ae5*	*Ae6*	*Ae1*	*Ae2*	*Ae3*	*Ae4*
	A_{E1}	A_{E2}	A_{E3}	A_{E4}	A_{E5}	A_{E6}
Distance String Pair:	*De5*	*De6*	*De1*	*De2*	*De3*	*De4*
	D_{E1}	D_{E2}	D_{E3}	D_{E4}	D_{E5}	D_{E6}

Figure 6.12: Effects from Edge Alignment on Crossover.

The first heuristic technique is to adjust the match sequence of the edge boundaries stored in the genetic strings according to the geometric alignment of the two corresponding polygon shapes. Such adjustment needs to be made before a crossover is applied. Figure 6.12 gives an illustration. Without such preadjustment, the crossover would take the initial storage sequence in both the "Angle String" pair and "Distance String" pair as the match sequence between the edge boundaries, i.e., edge $e1$ would cross with edge $E1$ and so on. Apparently it would not give proper exploitation on those closely aligned edge boundaries indicated with dark outlines. After the preadjustment, the correct match between the edge boundaries is found where edge $e1$ should be paired with edge $E3$ and so on. During such an adjustment, both "Angle String" and "Distance String" of one paired polygon need to be shifted according to the correct match. In Figure 6.12, the two strings of the left polygon are shifted until the edge boundaries are aligned. The edge boundaries are considered as aligned whenever the mismatch value of the boundary shapes corresponding to each shifting is the minimum. Such shape alignment procedure is exactly the same as the matching of two polygons using $L2$ distance function illustrated in Chapter 5, which is implemented using Equation (5.2).

Figure 6.13 schematically shows how the alignment is detected through the match between the turning angle functions of the two polygons in Figure 6.12. The thinner function curve corresponds to the left polygon which is shifted during the alignment search. The top plot shows the initial state and the bottom plot shows the aligned state. The total shaded area between two matched function curves indicates the mismatch value. The improvement on the edge boundary alignment from the initial state to the aligned state can be observed from the less shaded area of the bottom plot as compared to the top plot. Through such alignment, the further crossover always becomes favorable to exploit the maximum level of common geometric features shared by the two parent polygons.

The second heuristic is used to deal with the "last-edge problem," which refers to the fact that there always exists a last edge boundary unable to participate in the crossover operation due to closing of a polygon shape, and thus receives an unfair amount of search effort. One way to dilute the effect of such a problem can be to randomly pick the last edge before each crossover starts. Under the randomness of the evolution processes, the

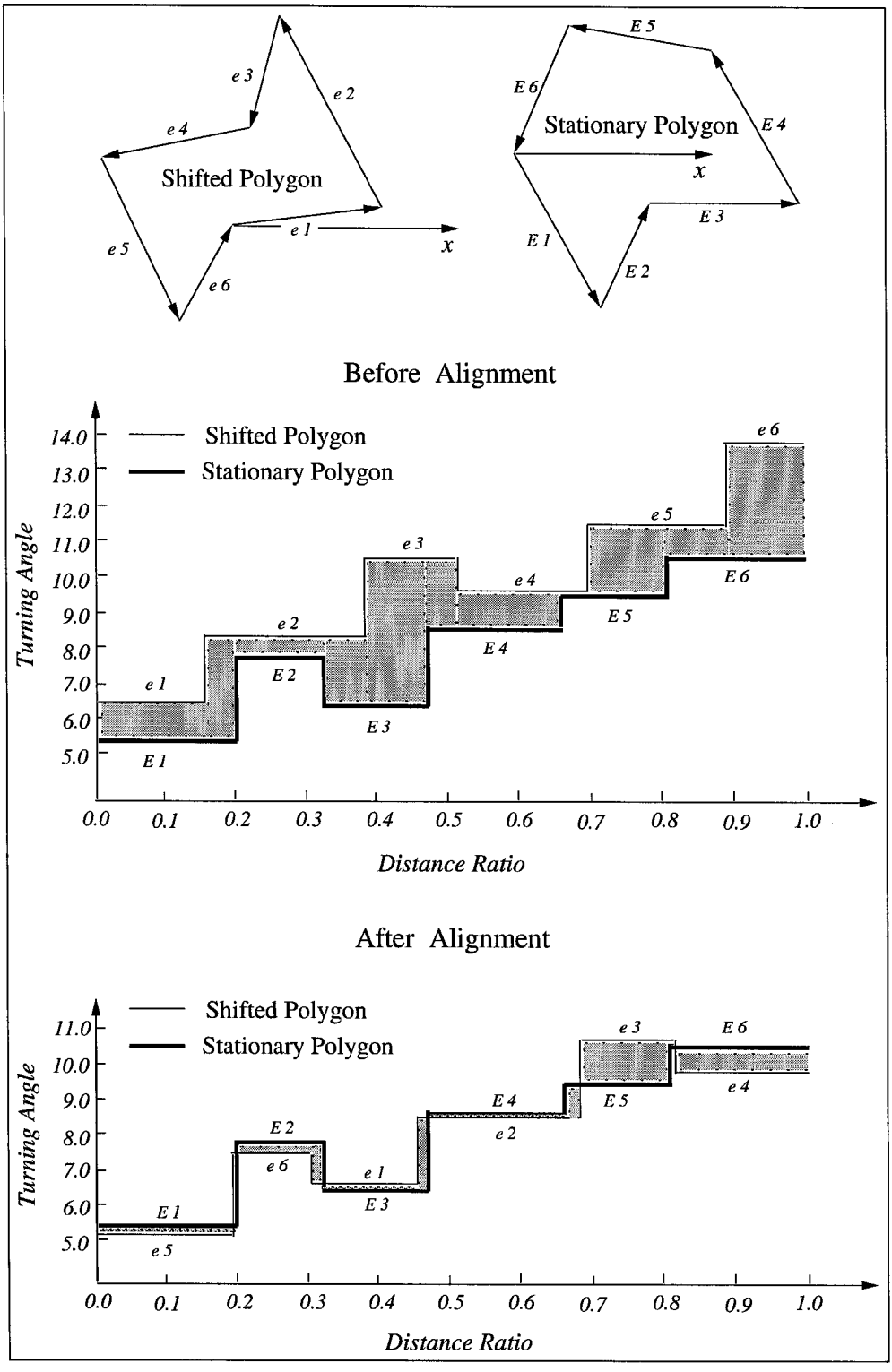
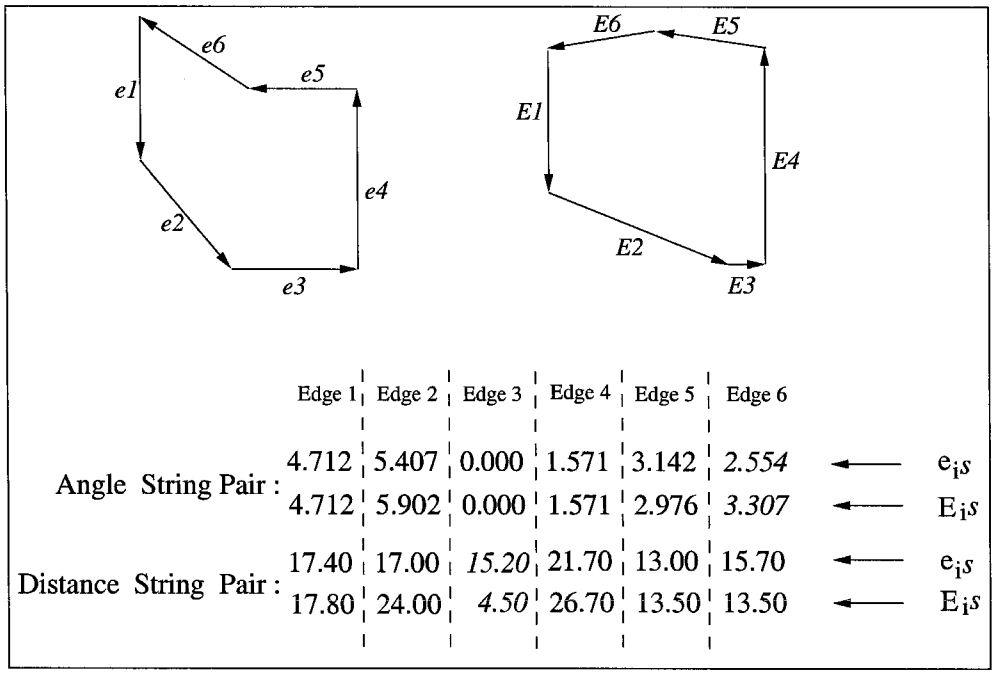


Figure 6.13: Edge Boundary Mismatch vs. Edge Alignment.



	Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	
Angle String Pair :	4.712	5.407	0.000	1.571	3.142	*2.554*	← $e_i s$
	4.712	5.902	0.000	1.571	2.976	*3.307*	← $E_i s$
Distance String Pair :	17.40	17.00	*15.20*	21.70	13.00	15.70	← $e_i s$
	17.80	24.00	*4.50*	26.70	13.50	13.50	← $E_i s$

Figure 6.14: Edge Boundary Match Between Two Simple Polygons.

overall edge boundaries may have a better chance to be get a fairly balanced search effort. However, it still does not solve the problem. An algorithm is developed below to solve the problem in the sense that the search effort from the crossover can still be made according to the expected exploitation or exploration level of all edge boundaries. Since the last edge is formed without direct control from the crossover operator, its formation can be effectively regarded as a random exploration. Therefore, if the last edge happens to be part of the boundaries expected to be aggressively explored through the crossover, there is no problem.

The problem comes when the last edge boundary belongs to the common features between the parent polygons, which needs exploitation instead of exploration. The loss of the control on such an edge could easily break those closely matched edge boundaries. So an effective approach to attack this problem is to avoid choosing the last edge as part of the closely matched edge boundaries, and in fact the last edge should always correspond to the worst matched edge boundaries of the two parent polygons. As a start, the worst matched edge boundaries between the two parent polygons have to be determined. The difficulty here is that the "Angle String" pair and the "Distance String" pair are processed separately.

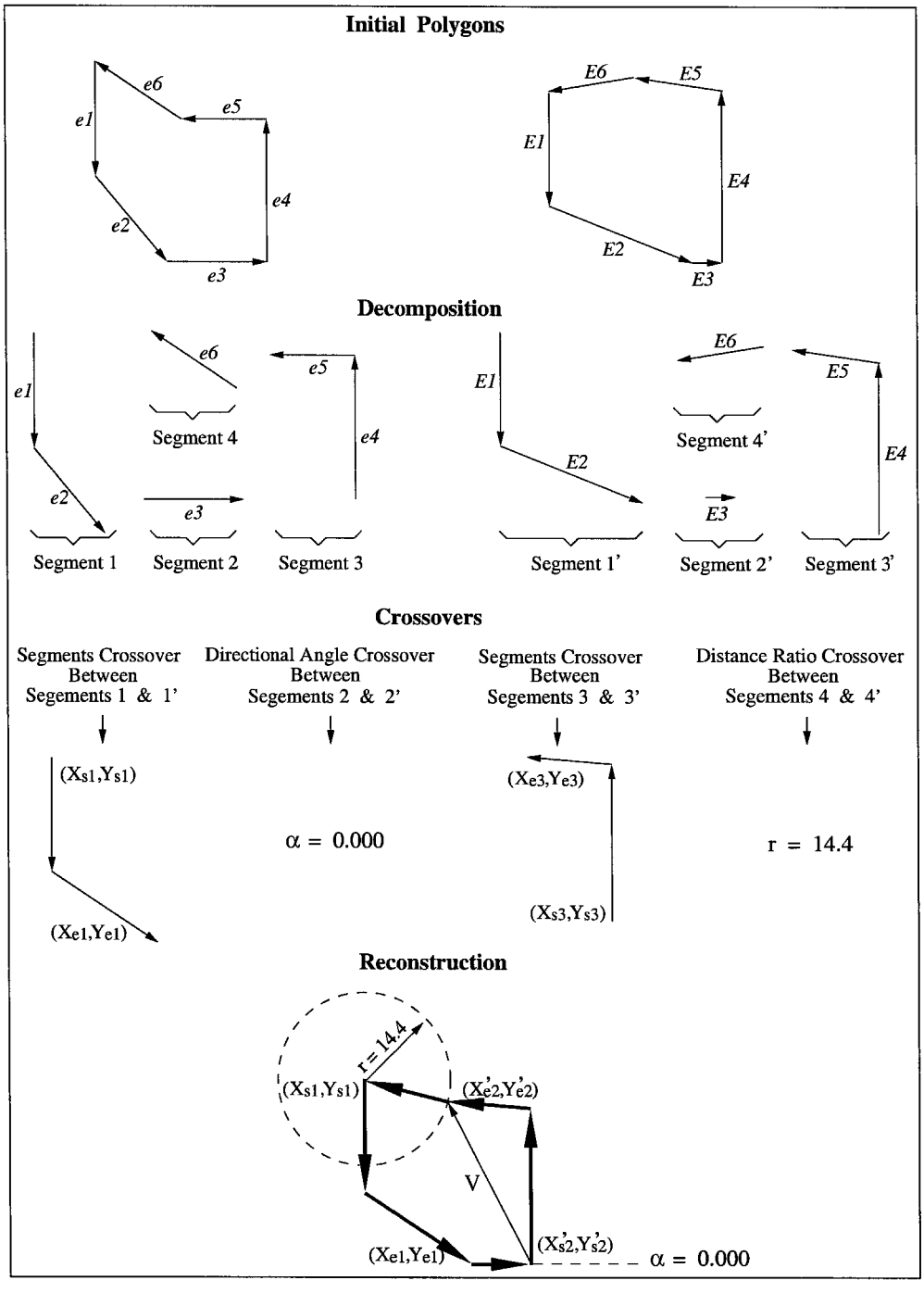


Figure 6.15: An Crossover Between Two Simple Polygons.

The worst match between the edge directional angles and between the edge distance ratios can be found by comparing all the elements in each string pair respectively. However, the two worst matches may not correspond to the same edge boundary. Figure 6.14 gives a schematic illustration. The match between the edge boundaries of the two polygons is shown through the "Angle String" pair *and* "Distance String" pair. The largest gap between the paired element values is highlighted with italics in each string pair. The worst match of edge directional angles occurs between Edge $e6$ and Edge $E6$, while the worst match of edge distance ratios occurs between Edge $e3$ and Edge $E3$. One way to overcome such a dilemma is illustrated in Figure 6.15. Both parent polygons are first decomposed into four segments. The second segment and the forth segment of each polygon are assigned to the edge boundaries with the worst distance ratio match and the worst directional angle match respectively. Consider the left polygon in the figure: Segment2 is Edge $e3$ and Segment4 is Edge $e6$. Then the first segment is the edge boundary portion from the forth segment to the second segment, and the third segment is from the second segment to the fourth segment. In the case of the left polygon, Segment1 includes Edge $e1$ and Edge $e2$ and Segment3 has Edge $e4$ and Edge $e5$. After the segment decompositions, the correspondent segments of the two polygons are paired respectively, i.e., Segment1 and Segment1$'$ form the first segment pair and so on. The first segment pair and the third segments then both have the BLX-0.5 crossover operation applied to generate two evolved segments for a child polygon. As shown in the figure, the segment between vertex (X_{s_1}, Y_{s_1}) and vertex (X_{e_1}, Y_{e_1}) is the outcome of the crossover between Segment1 and Segment1$'$; the evolved segment between vertex (X_{s_3}, Y_{s_3}) and vertex (X_{e_3}, Y_{e_3}) is evolved by the crossover between Segment3 and Segment3$'$. The BLX-0.5 crossover between the second segment pair is only applied to its directional angles. Because the distance ratios of the second segment pair is the worst match in the "Distance String" pair, the child distance ratio will not be directly evolved from its parents, and instead it will be determined during the next reconstruction stage from the geometric constraints. Accordingly, the BLX-0.5 crossover between the forth segment pair is only applied to its distance ratios because of the worst directional angle match in this case. Therefore, in the figure, only the crossover between the directional angles of Segment2 and Segment2$'$ is applied

and yields the zero-valued directional angle of the third child edge. Correspondently, only the crossover between the distance ratios of Segment4 and Segment4′ is implemented, and the resulting distance ratio of the sixth child edge is 14.4. Lastly, the reconstruction is needed to construct the whole child polygon based on the evolved first and third segments plus the directional angle of the second segment and the distance ratio of the forth segment.

All the quantities are shown in the reconstruction section of Figure 6.15, which are:

$$
\begin{aligned}
(X_{s_1}, Y_{s_1}) &= \textit{the starting vertex of the evolved first child segment} \\
(X_{e_1}, Y_{e_1}) &= \textit{the end vertex of the evolved first child segment} \\
(X_{s_3}, Y_{s_3}) &= \textit{the starting vertex of the evolved third child segment} \\
(X_{e_3}, Y_{e_3}) &= \textit{the end vertex of the evolved third child segment} \\
(X'_{s_3}, Y'_{s_3}) &= \textit{the starting vertex of the shifted third child segment} \\
(X'_{e_3}, Y'_{e_3}) &= \textit{the end vertex of the shifted third child segment} \\
r &= \textit{the distance ratio of the forth child segment,} \\
\alpha &= \textit{the directional angle of the second child segment,} \\
V &= \textit{the vector from } (X_{s_3}, Y_{s_3}) \textit{ to } (X_{e_3}, Y_{e_3}).
\end{aligned}
$$

During the reconstruction process, the first evolved segment is fixed while shifting the third evolved segment so that two geometric constraints are satisfied: 1) the distance between (X_{s_1}, Y_{s_1}) and (X'_{e_3}, Y'_{e_3}) is equal to r; 2) the directional angle of the edge from (X_{e_1}, Y_{e_1}) to (X'_{s_3}, Y'_{s_3}) is α. Note that the vertex coordinates of (X_{s_1}, Y_{s_1}), (X_{e_1}, Y_{e_1}), (X_{s_3}, Y_{s_3}), (X_{e_3}, Y_{e_3}) and the values of r and α are known. The goal is to calculate the coordinates of the starting vertex (X'_{s_3}, Y'_{s_3}) of the shifted third segment. The following geometric relationship is shown in the calculation:

Constraint (1) gives:

$$(X'_{e_3} - X_{s_1})^2 + (Y'_{e_3} - Y_{s_1})^2 = r^2 \tag{6.3}$$

Constraint (2) gives:

$$\frac{Y'_{s_3} - Y_{e_1}}{X'_{s_3} - X_{e_1}} = \tan\alpha \tag{6.4}$$

Definition of V gives:

$$\begin{aligned} V &= (X_{e_3}, Y_{e_3}) - (X_{s_3}, Y_{s_3}) \\ &= (X'_{e_3}, Y'_{e_3}) - (X'_{s_3}, Y'_{s_3}) \\ V_x &= X_{e_3} - X_{s_3} = X'_{e_3} - X'_{s_3} \\ V_y &= Y_{e_3} - Y_{s_3} = Y'_{e_3} - Y'_{s_3} \end{aligned}$$

which implies:

$$\begin{aligned} X'_{e_3} &= X'_{s_3} + V_x = X'_{s_3} + X_{e_3} - X_{s_3} \\ Y'_{e_3} &= Y'_{s_3} + V_y = Y'_{s_3} + Y_{e_3} - Y_{s_3} \end{aligned} \tag{6.5}$$

Therefore, from (6.4):

$$Y'_{s_3} = X'_{s_3}\tan\alpha - X_{e_1}\tan\alpha + Y_{e_1} \tag{6.6}$$

Replacing X'_{e_3} and Y'_{e_3} in (6.3) with only X'_{s_3} based on (6.5) and (6.6):

$$X'^2_{s_3}(1 + \tan\alpha^2) + 2X'_{s_3}(t1 + t2\tan\alpha) + t1^2 + t2^2 - r^2 = 0 \tag{6.7}$$

where

$$\begin{aligned} t1 &= V_x - X_{s_1} \\ t2 &= Y_{e_1} + V_y - X_{e_1}\tan\alpha - Y_{s_1}. \end{aligned}$$

X'_{s_3} can be calculated by solving the quadratic Equation (6.7) and Y'_{s_3} can be obtained through (6.6) based on X'_{s_3}. The end vertex (X'_{e_3}, Y'_{e_3}) of the shifted third segment can be further determined from (6.5). Thus, the entire reconstruction is finished.

When the first and third segments are merged, there could be self-intersection between the segments, in which case the crossover is repeated until a valid child polygon is evolved. In addition, the solution of Equation (6.7) may have the cases of double roots, single root and no real root. In case of no real root, the crossover is simply repeated. Otherwise, the actual directional angle of the second segment based on the solution will be checked against the value of α. If the two match, the solution is accepted. If no match occurs, the crossover is also repeated. With double roots, both solutions may be acceptable under which case one solution is randomly chosen with 50 percent chance. The approach illustrated above can be applied to any two matched parent polygons. Two degenerate cases may occur. One is that the second segment and fourth segment may collapse into one edge boundary, which means one edge has both worst matches of directional angles and distance ratios. Such an edge will be the chosen last, and either the first or the third segment will vanish and the appearing segment will participate in the regular crossover. The other degenerate case is when the second segment and the fourth segment are adjacent to each other. This will shrink either the first segment or the third segment to a single vertex which will be vertex (X'_{s_3}, Y'_{s_3}). In this case, all the calculations are the same, except the values of V_x and V_y are zeros.

As can be seen that, with the second heuristic technique, the maximum level of common features between any paired parent polygons can be properly exploited without being randomly disrupted. Figure 6.16 shows the outcome comparison of the crossover with and without this technique. The two initial polygons are the same as those in Figure 6.15. Each crossover generates two children polygons. The polygons shown are arranged so that the left children are more like the left parent polygon. The boundary match between Edge $e6$ and Edge $E6$ is the worst directional angle match and the boundary match between Edge $e3$ and Edge $E3$ is the worst distance ratio match. The two child polygons in the middle row of the figure result from the crossover with the last edge randomly picked among the six edges. The left polygon is evolved with the last edge picked on Edge $e1'$. Its new directional angle ends up to be different from the values that both parents share.

The right polygon is evolved with Edge $E5'$ as the last edge. In this case, the new distance ratio for this edge is quite different from the region between its parents. In both

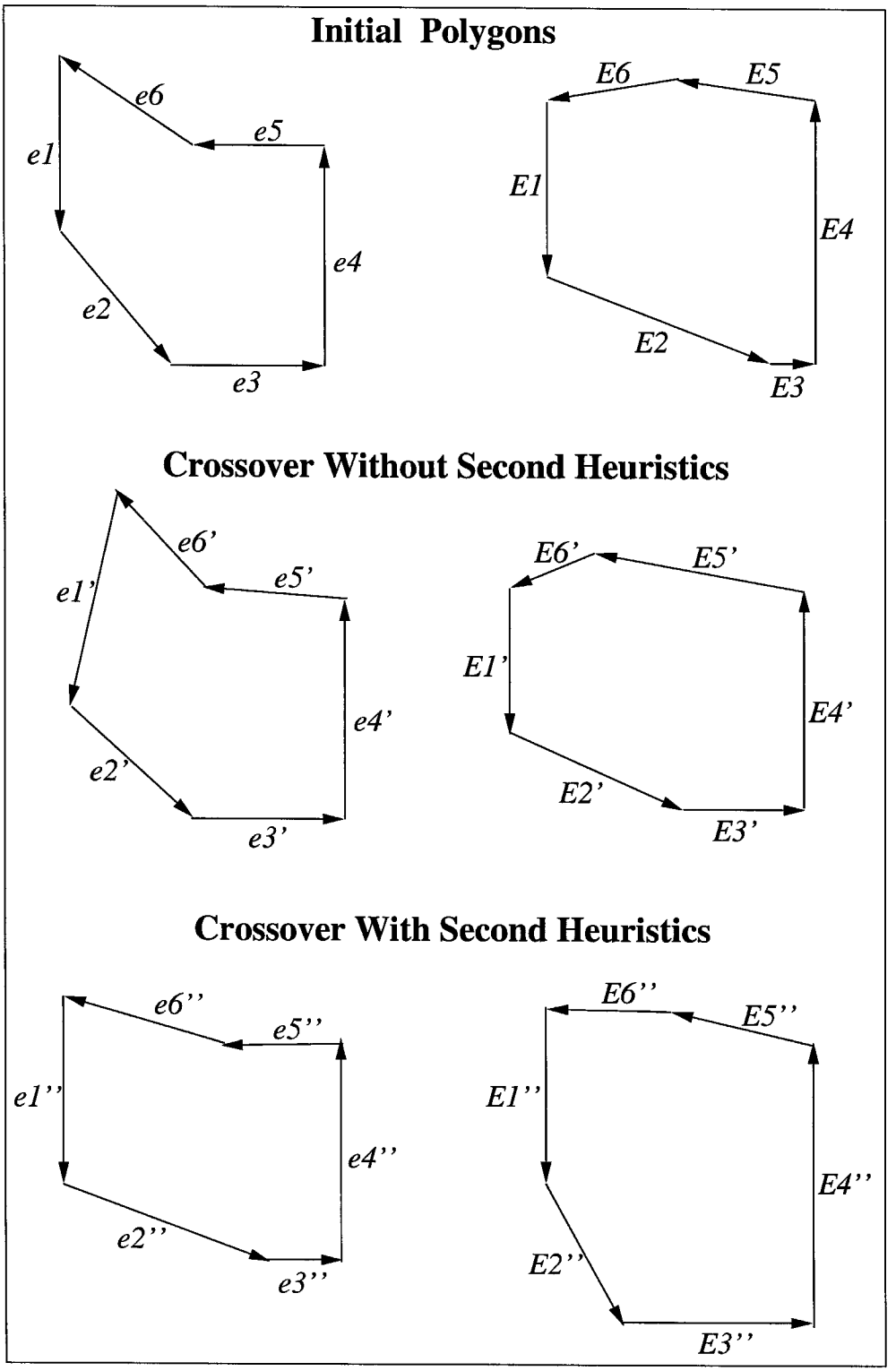


Figure 6.16: An Outcome Comparison on Crossovers with And without Second Heuristics.

cases, the evolved child polygons break the common features of their parents. The two polygons shown in the last row are the results of the crossover updated with the heuristic techniques. The improvement is that the common directional angles of the first, third, and forth edge boundaries of the parents are all inherited through their children, and in addition, the initial closely matched distance ratios of the first, forth, fifth and sixth edge boundaries are also properly exploited. Therefore, it gives an overall appropriate level of exploitation and exploration to individual boundaries according to their geometric features.

At this point, there is an additional note should be clarified. Based on the above crossover, it can be seen that the actual control over the evolution of edge distance ratios is not "direct". During initialization, the size of all the generated polygons are scaled to 100. However, after the crossover, the actual size of each evolved polygon may vary depending on all the evolved distance ratios. A scaling is needed to bring each evolved polygon size back to 100. So there is a discrepancy between the crossovered value and the actual distance ratio after the scaling. The discrepancy may be largely due to some unpredicted values caused by aggressive exploration on certain edge boundaries. This may raise questions on the effectiveness of evolving distance ratios under such a crossover operation. Actually, this discrepancy does not influence the performance of this crossover operation. The key concern here is to preserve the closely matched edge boundaries between any two parents, which include edge directional angles and relative edge distance ratios. Note that what matters is the relative distance ratios rather than the absolute distance ratios. As shown in Figure 6.15, during the reconstruction stage, the distance ratio of the second segment is generated without control which could vary a lot under the expected aggressive exploration. However, the edge lengths of the remaining edges are unchanged during the reconstruction process and thus the relative distance ratios remain at their crossovered values. So, if there are any closely matched features among those edges, they will be preserved.

The third heuristic involved here is the particular crossover operation on the directional angles. The operation of a normal BLX-0.5 will cut off the crossovered value to the closest boundary value whenever it exceeds the valid range. The valid boundary for edge directional angles is from 0 to 2π. Because the actual value interval for BLX-0.5 to randomly pick during the crossover is double the initial interval between the parent values, the angle

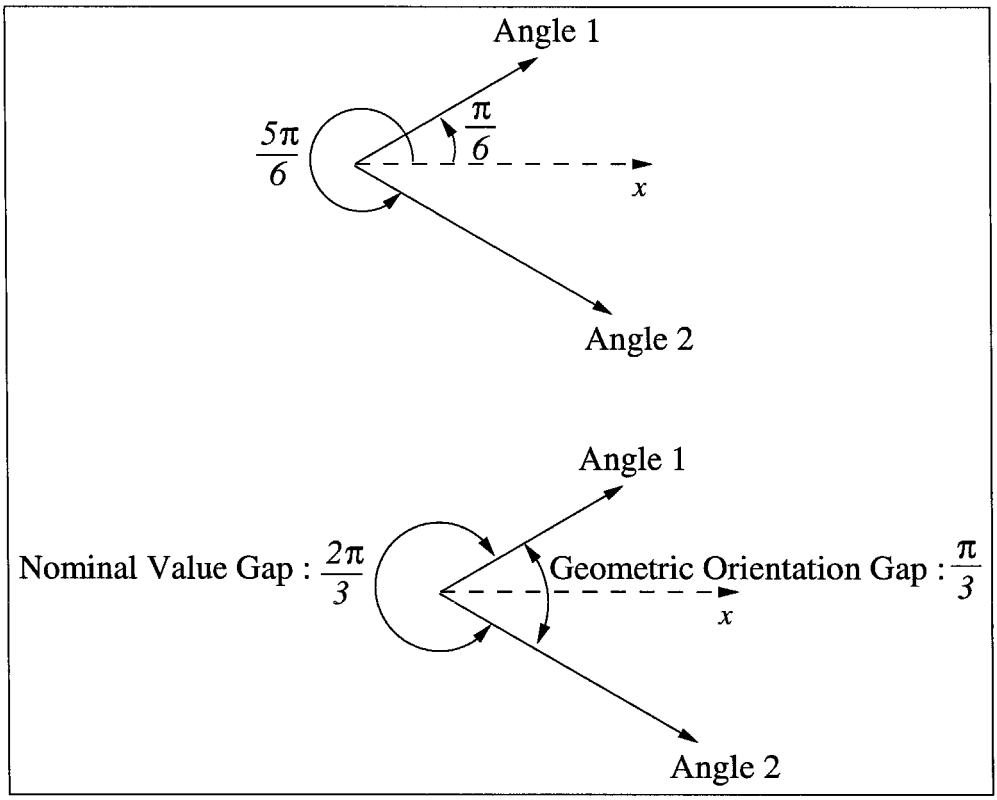


Figure 6.17: Difference Between Nominal Value Gap and Geometric Orientation Gap of Directional Angles.

value picked could be either negative or larger than 2π. Under normal operation, any negative angle value will be assigned as zero and an angle that exceeds 2π will be set to 2π. This treatment, however, is inappropriate because the edge directional angles are wrapped around. Any angle value has a one-to-one corresponding value between 0 to 2π with the same geometric orientation. So every time an evolved directional angle is outside the valid range, it will be converted back within 0 to 2π (mod(2π)) instead of being cut off.

Another heuristic treatment on a directional angles is involved here. Although it is fine to simply determine the initial parent interval as the nominal angle value gap, a more effective way is to use the geometric orientation gap, which is always smaller than or equal to π. Figure 6.17 shows the difference between the two. The geometric orientation gap measures the actual orientation difference between two directional angles and therefore is the appropriate interval value to use here.

In summary, the developed crossover focuses on a balance of searching effort between exploitation and exploration according to the edge boundary match of any two parent polygons. By using the BLX-0.5 operator, the searching range of the crossover is directly related to the initial parent value gaps, which allows the different search efforts to be continuously mapped to the levels of match between the parent edge boundaries. With the help of the first and the second heuristic techniques, the closely matched edge boundaries can be found and properly exploited, which further encourages the growth of common features among searched polygons. The rationality of this comes from analogy with the role of "schemata" for the searching of an optimal binary string illustrated in section 3.6. The edge boundary of a polygon can be pictured as a binary string with each edge corresponding to a binary bit. Any closely matched edge boundaries among a generation of polygon shapes can be used to define the "schema." The goal of the entire evolutionary searching process is to search and evolve the optimal "schema" with more and more concrete "bits." In particular, among any "well-performed" individuals, it is overall more likely that the carried "schemata" will serve as promising basis for the evolution of children with even better performance and more concrete "schemata." Here, by recognizing the close features of polygon boundaries among the parents and preserving them through the generation of children, it is more likely to evolve better performed polygon shapes with more and more edge boundaries becoming closely matched. However, just like any "schemata" building processes, under such an edge boundary building process, it also possible for the search process to be trapped into local optimal shapes if the trusted common features among individuals actually represents the non-global optimal ones. The overall judgment of such risk also depends on other factors such as selection scheme and genetic operations. Especially with the use of the mutation operation designed in the next section, such risk can be effectively reduced.

6.3.5 Mask Mutation

As described in section 3.5, the purpose of mutation is to alter the current features of phenotypes through random disruptions of the genetic structures of the corresponding genotypes, and with that the sampling variety can be maintained for a current generation. The mutation developed is not the main force to drive the entire searching process. Thus, the

design of mutation is subordinated to the design of the crossover. Under an appropriate evolutionary strategy, the supplementary role of mutation is rather important, which could well amend the weakness of crossovers. The crossover developed in the previous section is rather conservative in preserving the common features among polygon shapes. If such common features turn out to be a dominant portion of a local optimal polygon shape, the entire search is likely to be trapped. Therefore, the goal of the mutation here is to break such common features among the currently sampled polygon shapes. There are two kinds of mutations constructed to fulfill such purpose.

However, before the design of mutation, two issues need to be addressed. One is the satisfaction on the polygon simplicity constraint. Similar to the crossover, self-intersection is incrementally checked for all the mutated edges to ensure this constraint is not violated. The other is the choice of coding scheme. Again, the edge-coding scheme is used because of its effective genetic representation of polygon edge boundaries. Mutation is applied on a single parent genotype each time. Under the edge-coding scheme, the "Angle String" and "Distance String" are mutated independently. During the mutation on each string, one or more string elements are replaced with the new values randomly regenerated within a value boundary. All the elements of "Angle String" correspond to edge directional angles and thus the value boundary is always from 0 to 2π. The elements in "Distance String" represent the edge distance ratios with the value range from 0 to 50 as mentioned in section 6.3.3. Actually, the upper bound for the distance ratios needs to be further adjusted for two reasons. First, just like crossover, there is a discrepancy between the mutated value and the actual distance ratios due to the polygon size scaling. Second, to actually mutate a parent distance ratio up to around 50 will lead to a rather odd shape which is hard to construct and useless for most applications unless the parent distance ratio is already close to 50. So the upper bound used is set so that it varies for each parent distance ratio x with the following formula:

$$u = \frac{(5000 + 100 * x)}{(150 - x)}, \tag{6.8}$$

where

$$x = \text{the parent distance ratio to be mutated}$$

$$u = \text{the mutation upper bound.}$$

The above formula is based on the assumption that all the parent polygons have the common size of 100. The upper bound of the actual distance ratio is set as the middle value between the parent value x and 50.

The first kind of mutation is designed to directly disrupt some existing common edge boundaries of a parent polygon. During the mutation, a polygon shape is picked to be mutated, and is called the mutation polygon. Then one or several other polygon shapes are chosen from a current generation and are called companion polygons. The companion polygons are usually the best or several top performing individuals. Then the mutation polygon is compared against each of the companion polygons. Each comparison gives the differences in their edge directional angles and edge distance ratios. Note that the prealignment of the edge boundaries is needed before each two polygons are compared. Then one final comparison result is taken as the average of all the individual comparisons. From the final comparison result, the edge corresponding to the smallest difference in edge directional angles is selected as the "Angle Mutation Edge," whose directional angle will be modified with the mutation operation. The edge with the smallest difference in distance ratios is selected as the "Distance Mutation Edge," and its distance ratio will be mutated directly. After both mutations are implemented, a reconstruction is needed to form the whole mutated polygon shape. The reconstruction is exactly the same as the crossover with the "Angle Mutation Edge" as the second segment, and the "Distance Mutation Edge" as the fourth segment. Figure 6.18 shows an example of the mutation process. In the mutation polygon, the "Angle Mutation Edge" corresponds to Edge $e4$ and the "Distance Mutation Edge" is Edge $e1$. The mutation operation will mutate the directional angle of Edge $e4$ to be α, and the distance ratio of Edge $e1$ to be r. During the reconstruction, both α and r will be the constraints for the formation of the final mutated polygon.

The effect of such a mutation is shown in both Figures 6.19 and 6.20. Figure 6.19

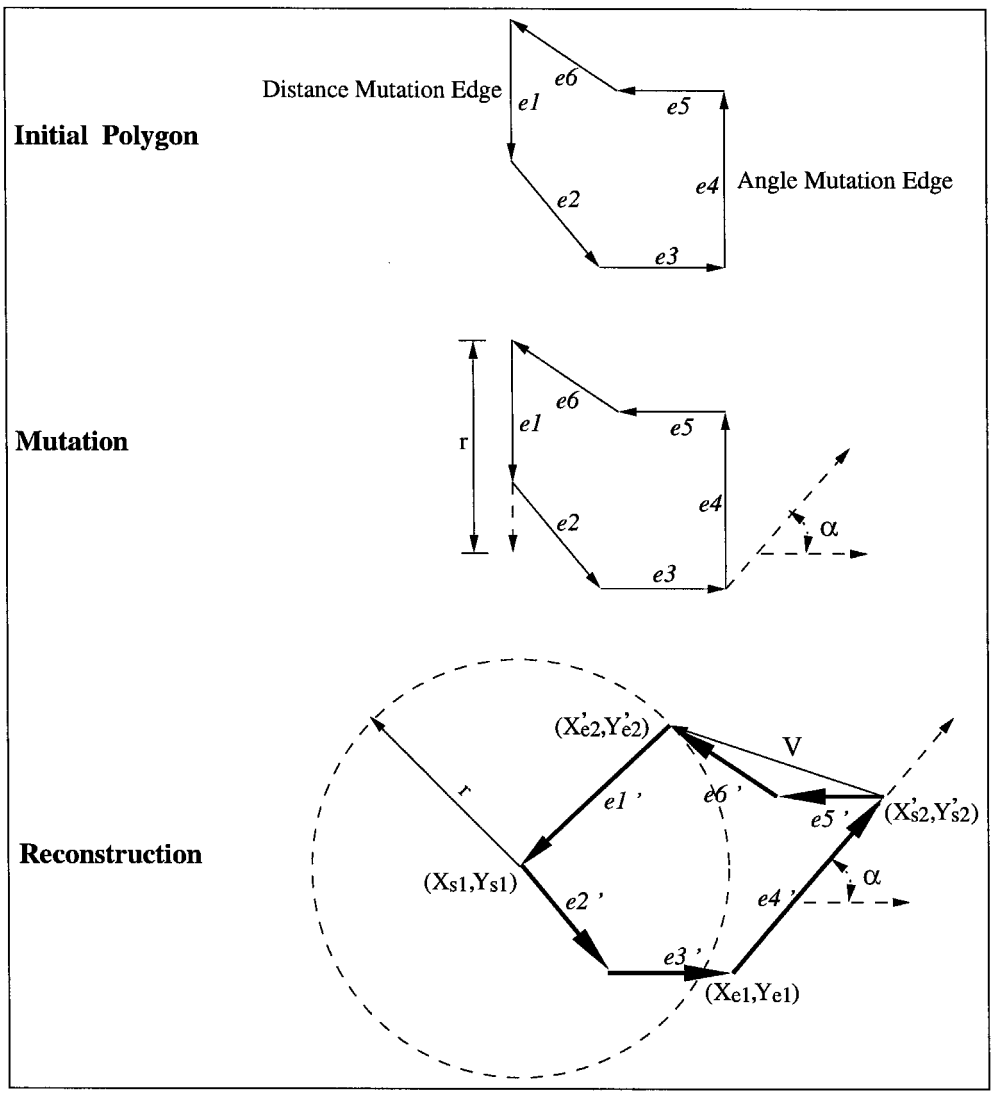


Figure 6.18: The First Kind of Mutation.

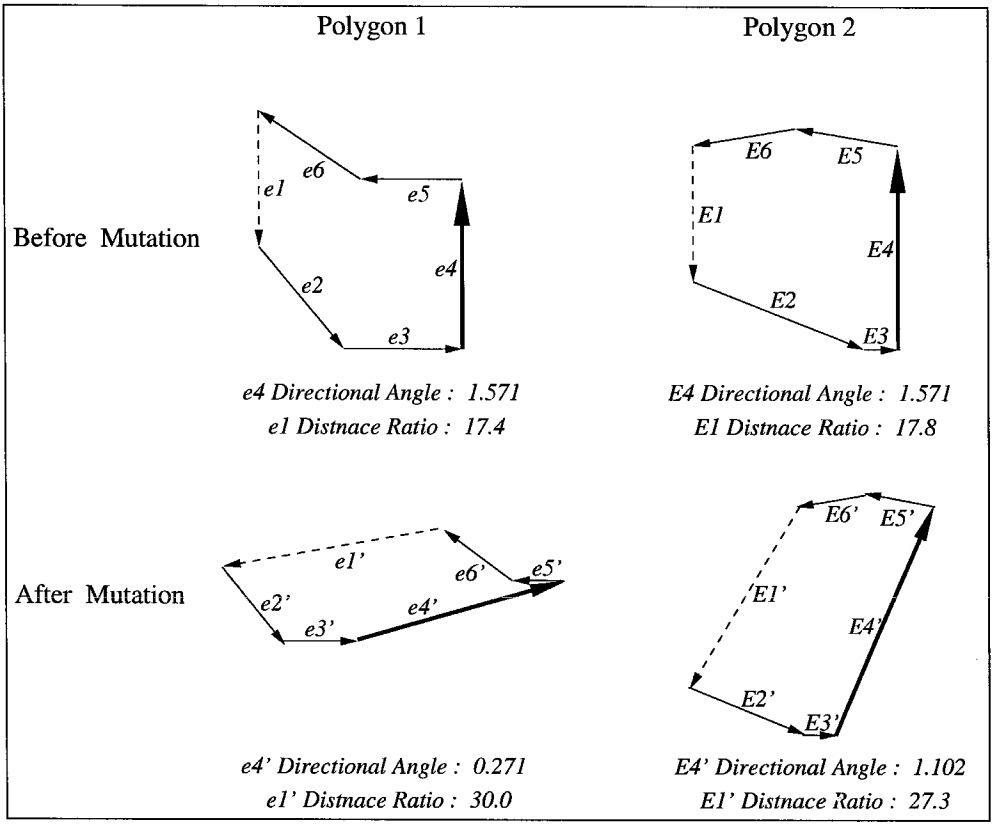


Figure 6.19: The First Kind of Mutation to Break Down Common Features Between Two Polygons.

shows the mutation results on two polygons whose edge boundary match is shown in Figure 6.14. The darker edges indicate the "Angle Mutation Edge"s, which have the same vertical directional angles. The dotted edges mark the "Distance Mutation Edge"s, which have the closest match of distance ratios. During the test, the two polygons are aligned so as to automatically detect the "Angle Mutation Edge" and "Distance Mutation Edge" for each. In the mutation results presented, since all the polygon sizes are scaled to the same value, the edge lengths can indicate the relative distance ratios. The results show that both the directional angles and the distance ratios of the mutated edges have been altered aggressively.

Figure 6.20 illustrates the disruptive power of this operation through a comparison of two evolutions with and without the participation of the mutation. Both evolutions are based on two randomly generated 8-side polygons and last 150 iterations. The first evolution is

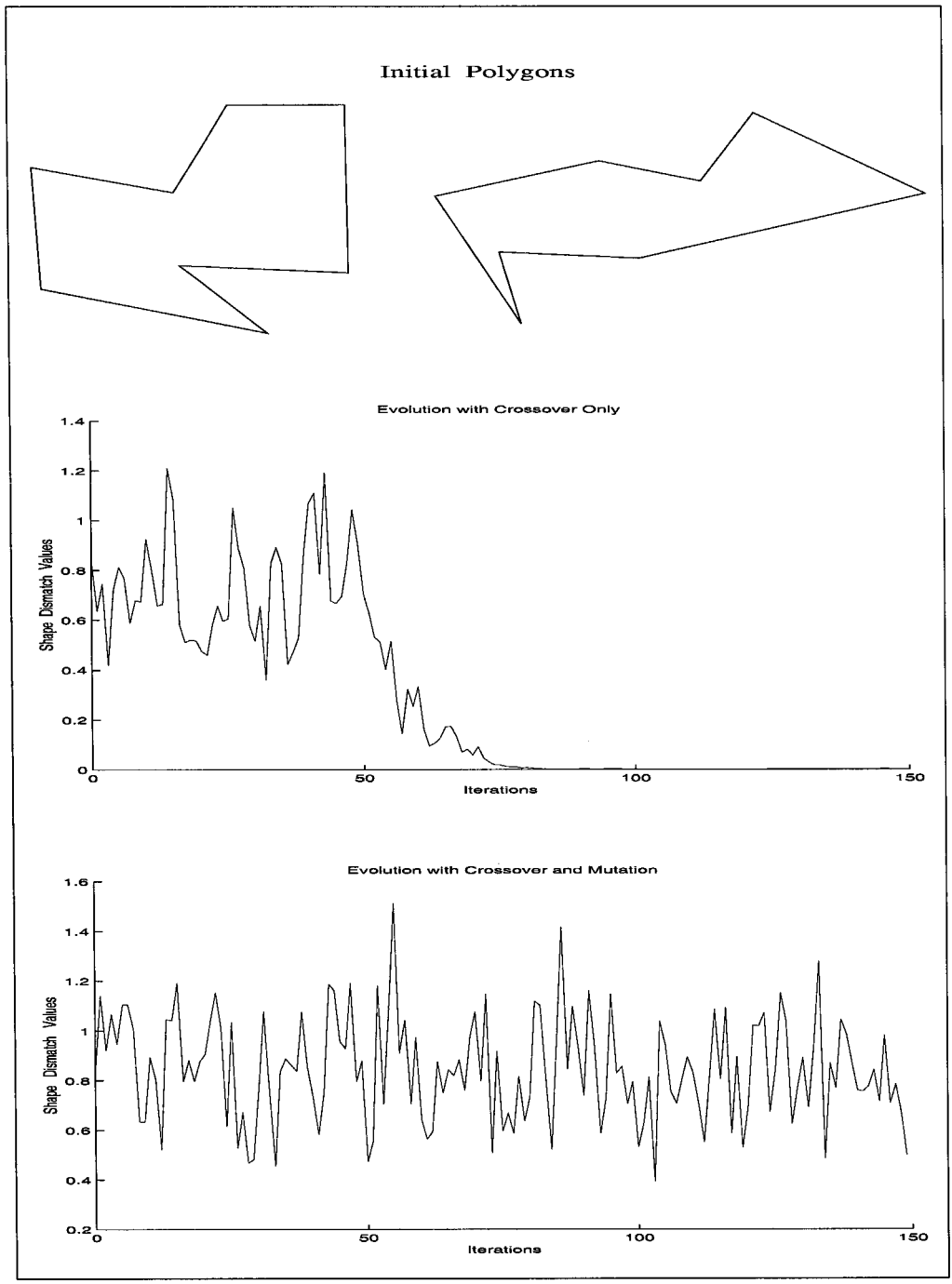


Figure 6.20: The First Kind of Mutation to Maintain the Shape Difference Between Two Randomly Generated 8-Sided Polygons.

purely carried by the previously developed crossover. During each iteration, the current two parent polygons are applied with the crossover, and the generated two children polygons will serve as the parent polygons in the next iteration. The second evolution follows the similar procedures except that during every iteration, the two child polygons generated are mutated and then serve as the parent polygons for the next iteration. The initial two parent polygons of both evolutions are the two shown at the top of the figure. The middle plot shows the result from the first evolution. It can be seen that the difference between the two polygon shapes varies a lot during the initial stage of evolution, which covers about the first 50 iterations. This indicates the dominant effect from the exploration on those largely mismatched edge boundaries, which can be expected because the edge boundaries of the two initial polygons are quite different. After the initial stage, the evolution starts converging in terms of the shape difference between the two parent polygons until saturation occurs when the difference almost vanishes. This converging stage indicates that the dominant searching effort has been switched from exploration to exploitation once enough common boundary features of the two parent polygons have been evolved. This evolution process shows the typical behavior of the crossover applied to any two polygon shapes. The bottom plot shows the result from the second evolution with the mutation. This time, the entire evolution process is dominated by the exploration on the edge boundaries of any two parent polygons. It is the mutation that effectively breaks any common boundary features evolved and preserved by the crossover.

So far, the first kind of mutation is developed to actively disrupt some existing common boundary features in the current generation. The real purpose of doing this is to break any "bad" edge boundary "schemata" which are evolved and preserved by the crossover so as to prevent the entire evolution from being trapped into local optimal polygon shape. During the early or even middle stages of evolution, with the effective power of disruption, such mutation will be helpful to prevent premature convergence. However, if such mutation is constantly used throughout the evolution, the ultimate goal of proper convergence will be slowed or even never be achieved. Furthermore, it should be noted that, only "bad" edge boundaries need to be destroyed. With such mutation, however, any edge boundaries could be disrupted as long as it represents common features among individuals, which could

include the "good" ones. Therefore, such mutation should be used with caution. With this, the second kind of mutation is designed here to particularly break the edge boundaries that could be "bad" ones with high likelihood.

The entire mutation process consists of a detection stage and a disruption stage. The detection stage is to detect two situations that will lead to the local stagnancy of the evolution process, which have been most frequently encountered during the actual testing. Figure 6.21 gives a schematic illustration. The figure shows a global optimal polygon on the left and two local optimal polygons on the right. Note that the overall edge boundaries of both local optimal polygons are fairly close to the globally optimal one. However, in the first local optimal polygon, the directional angles of Edge $e4$ and Edge $e5$ are closely aligned. In the second local optimal polygon, the distance ratio of Edge $e4$ is very small and stuck in the corner between Edge $e3$ and Edge $e5$. In both cases, Edge $e4$ and Edge $e5$ effectively match Edge $E5$. The boundary of Edge $E3$, on the other hand, is not matched by any of the local optimal ones due to its relatively small distance ratio. The problem here is that some boundary features in the local optimal polygons are represented through two or more edges, but should be obtained with just a single edge. With such an edge misalignment, due to the limited number of edges provided, there is not enough room for the rest of the edges to capture all the local boundary features of the global optimal polygon. So the detection here is to check if there is sufficient directional angle alignment or very small distance ratio difference between any consecutive edges. If any is detected, the mutation is used to shortcut the two edges by a single edge with a consistent winding direction. As shown in the two mutated polygons, the original Edge $e4$ and Edge $e5$ are replaced by Edge $e5'$. Such treatment requires an additional new edge to be added to maintain the constant number of edge boundaries. A split operation is used to cut another single edge into half size and mutate the directional angle of the first half within a specified small range to form a new edge and another new edge is formed to close the entire polygon. Usually the edge to be cut is the furthest edge away from the detection site of the edge boundaries. As shown through the mutated polygons in the figure, the original Edge $e2$ is chosen to be cut, after which Edge $e2'$ is mutated away from the original orientation, and Edge $e3'$ is formed to close the polygon.

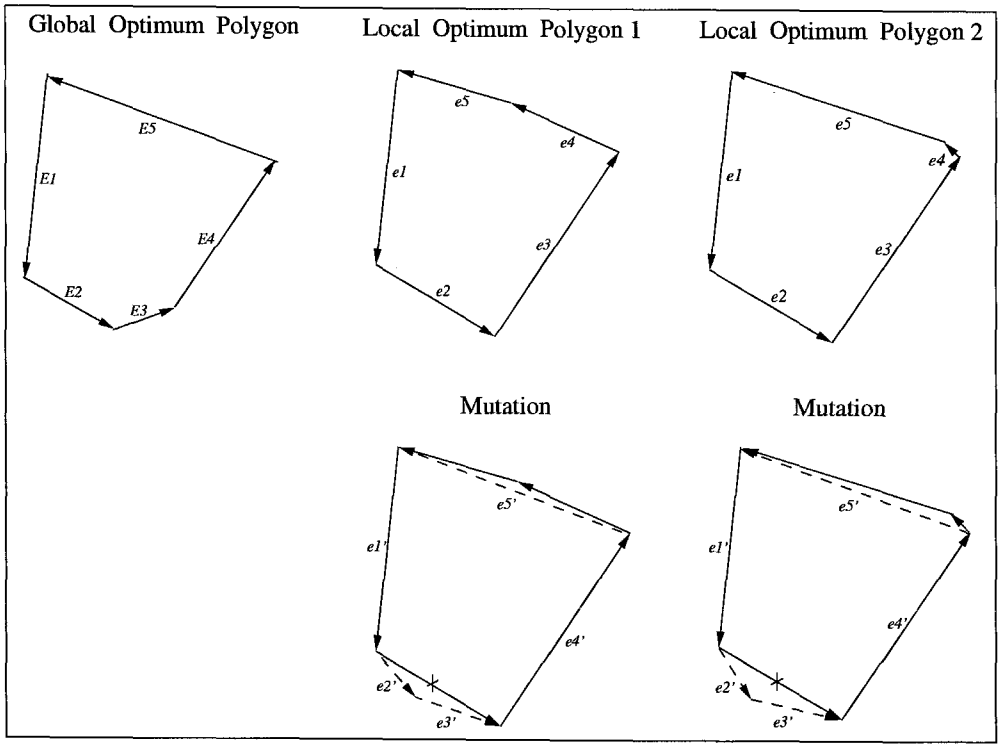


Figure 6.21: The Second Kind of Mutation.

In summary, the two developed mutations focus on the disruption of partial features of a single polygon shape. The first kind of mutation directly breaks the most commonly shared edge boundaries in the current generation and thus exhibits strong disruptive effect on evolution processes. So it can be used in the early stage of evolution to maintain the variety of the sampled solutions and prevent premature convergence. The second kind of mutation detects any aligned consecutive edge boundaries as the "bad" ones and modifies it without varying the rest of the boundary features too much. So it should particularly be used when the entire generation starts consistently converging and more and more of the polygon shapes are closer to each other, in which case, the detected edge alignment will have a strong indication of being a local trap and thus with the use of such mutation, the evolution power can be regained for the further searching.

6.3.6 Symmetry

In some applications, the evolved mask-layout may be known to have certain symmetry. As an example, an evolutionary algorithm will be constructed below to evolve mask-layouts whose etched shapes under a simulated silicon etching process are close to a desired shape. As mentioned in Chapter 4, most anisotropic etching rates have a high level of symmetry such as four-fold symmetry. If a desired shape has symmetry consistent with the symmetry of the etching rates, the searching of any optimal mask-layouts can be limited to those which have the same symmetry as both the desired shape and the etching rates. By doing so, the searching effort can be greatly reduced, simply because the necessary number of edges to be evolved becomes much smaller. In this section, only symmetry with respect to four coordinate axes are discussed, along with the modified genetic operators. There are two types of symmetries associated with coordinate axes. One is called quadrisymmetry in which case a mask-layout is symmetric to both x and y axis, or in other words, the edge boundaries in each quadrant are exactly the same. The other is called semisymmetry in which case a mask-layout is symmetric to either x-axis or y-axis but not both.

Quadrisymmetry

There are four types of quadrisymmetries as shown through the four instances of polygons in Figure 6.22. The solid line in each polygon indicates the portion of the edge boundaries in the first quadrant. This portion further consists of two kinds of edges as marked with different line thickness. The thinner line indicates the constraint edges whose directional angles are aligned with one of the two axes and the thicker line marks the rest of the edges which are named free edges. The two types of quadrisymmetries shown in the first row are for the polygons with the number of edges divisible by 4 and the other two types are for the polygons with even number of edges but not divisible by 4. So during an evolution with a fixed number of edges, only two types of quadrisymmetries could be involved. Unfortunately, each one of the two types has incompatible features and has to be considered separately. The incompatible features come from the existence of those distinct constraint edges in each type.

Due to such symmetry, only the edges in the first quadrant need to be evolved. However,

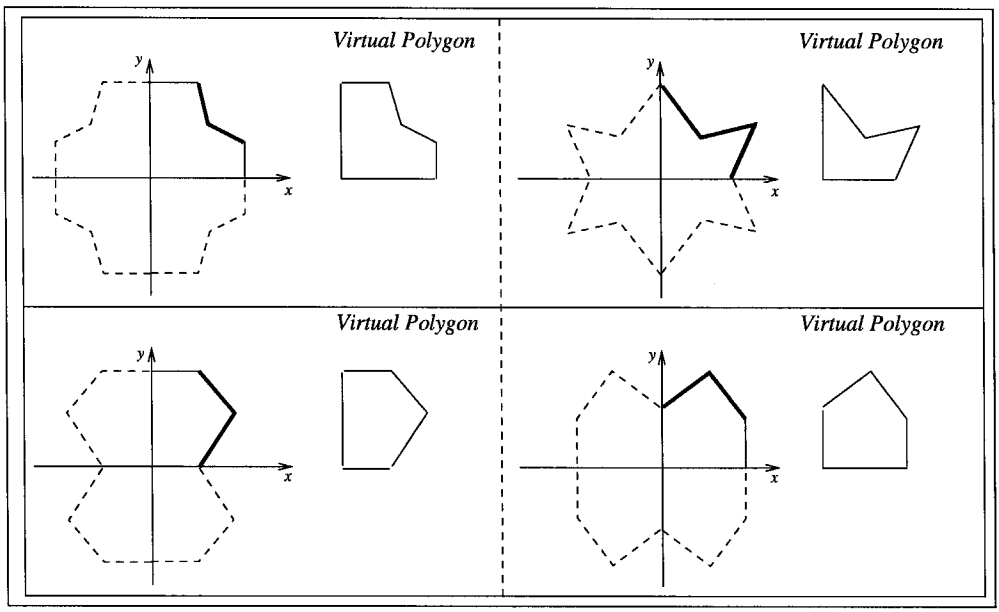


Figure 6.22: Four Types of Quadrisymmetries.

the genetic operations need to be modified slightly in order to take advantage of that. In the coding scheme, the edge-coding scheme is still used but with only the edges in the first quadrant need to be maintained. Every time an underlying mask polygon needs to be decoded, all the stored edges can be reflected about the axes to reconstruct the whole polygon. During the initialization, the first thing to be done is to randomly choose one of the two possible symmetry types and then, according to the chosen types, normal initialization techniques can be used to generate those free edges. Only the edge distance ratios need to be randomly generated for those constraint edges. During crossover, the first thing is to detect whether the two paired parent polygons belong to the same type. The crossover can only be applied to the two polygons with the same symmetry types. Then, the same developed crossover as shown in Figure 6.15 can be applied here. However, in this case, the polygon to be used is formed by all the first quadrant edges plus the two enclosed portions of the positive x-axis and positive y-axis. Such a polygon is named a virtual polygon, as shown in Figure 6.22 for each symmetry type. For a crossover between any two virtual polygons, only the free edges can be possibly chosen as the worst directional angle match. However, the constraint edges and the two axis edges can be chosen as the worst edge distance ratio match. Normal mutations can be applied here without changing the directional angles of

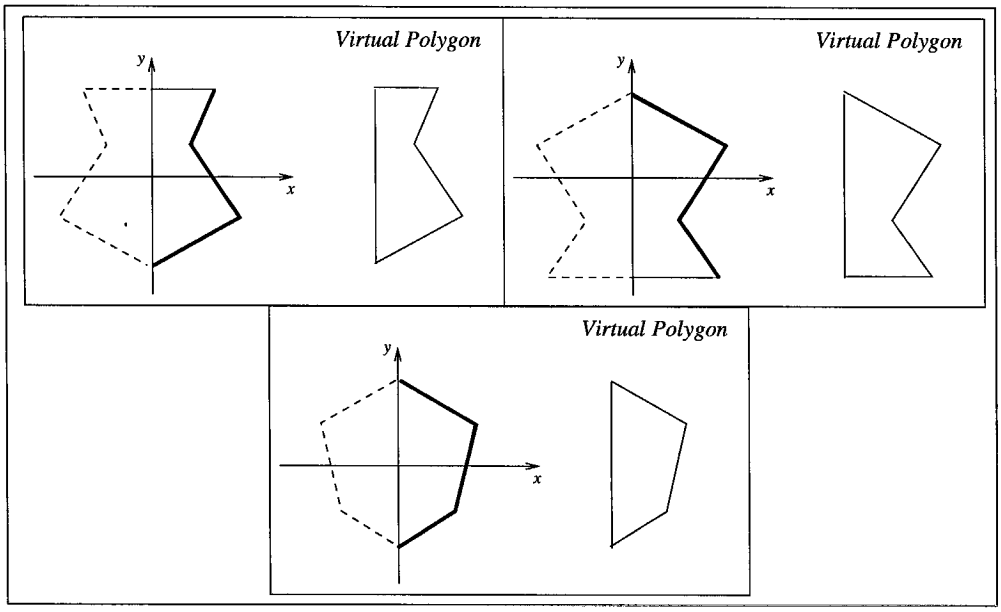


Figure 6.23: Three Types of Semisymmetries.

those constraint edges.

Semisymmetry

There are two major kinds of semisymmetries: the symmetry about the x-axis and the symmetry about the y-axis. All the used techniques are exactly the same for the two. So the discussion is only focused on one kind. Consider the semisymmetry about the x-axis. There are three types of semisymmetries as shown through the three instances of polygons in Figure 6.23. The solid line in each polygon indicates the portion of the edge boundaries in the positive x-axis side. Again, this portion further consists of two kinds of edges as marked with different line thickness. The thinner line indicates the constraint edges whose directional angles are aligned with x-axis and the thicker line marks the rest of the edges which are named free edges. The two types shown in the first row are for polygons with an odd number of edges, while the third type is for polygons with an even number of edges. So in this case, during an evolution with an even number of edges, there is no separation of different types involved. However, in case of with an odd number of edges, there are still two types of semisymmetries involved, and again they have to be considered separately during an evolution.

Due to such symmetry, only the edges in the positive x-axis side need to be evolved. All the modifications of genetic operators on quadrisymmetry can be applied here except that the virtual polygons only include the enclosed portion of y-axis, which are shown in Figure 6.23 for each type.

6.4 Evolutionary Strategy Module

6.4.1 Overview

The goal of the evolutionary strategy is to control the convergence of the searching process so as to maximize the overall likelihood of finding the global solution region within a limited searching effort. As mentioned in Chapter 3, there are many factors that affect the convergence of an evolutionary process which mainly include generation size, genetic operations with crossovers and mutations, selection schemes and stopping criteria. So the task of each evolutionary strategy becomes to determine all these factors so as to realize a proper convergence. However, there does not exist a single evolutionary strategy which is suitable for all applications. Therefore, the major effort made in this section is to provide a few effective and easily used designs of particular strategies.

A deeper analysis of the dynamics of mask-layout evolution is particularly important, because it explores issues applicable to any application that uses mask-layout evolution. Furthermore, the understanding from such analysis will shed light on answering some of the fundamental questions like why and how an evolutionary algorithm can be used to solve mask-layout synthesis, and what challenges will such a technique will face, and more importantly how to attack those challenges.

Three convergence control methods are provided here, based on the two most influential factors that affect the entire evolution performance: the selection scheme and genetic operations. Actual tests are used to demonstrate the effectiveness of these methods through various comparisons. In the end, evolution strategies are developed with the focus on how to overcome one kind of deception that is most likely to be encountered in most mask-layout evolutions. Even though effort has been made to design the strategies without being limited to specific applications so as to maximize their reuse level, they are not appropriate

for all applications. The main purpose of presenting them is to show in a real case how effective strategies can be constructed through the use of the three methods. Of course, both the strategies and the real tests reflect the actual dynamics of mask-layout evolution, and thus can serve as guidelines for future applications.

6.4.2 Evolution Analysis

The schemata theorem introduced in section 3.6 can be extended here to analyze the dynamics of an evolutionary process on mask-layouts. The underlying searching space of each evolution consists of all the 2D polygon shapes with the same specified number of edges. Every polygon shape is composed of a series of edge boundaries each of which is described with a distance ratio and directional angle pair. A "schema" can be interpreted as a set of polygon shapes with similarities at certain edge boundaries. Those similar edge boundaries are considered as defined edge boundaries and the remaining portion is taken to be undefined edge boundaries. Such a "schema" is called an edge boundary schema. Thus, the order of an edge boundary schema is the number of the defined edge boundaries. The defining length refers to the largest number of undefined edge boundaries between any two defined ones. Building blocks are specifically the short low-order edge boundary schemata with above-average fitness values and are named edge boundary building blocks. So the entire evolution of an optimal mask-layouts can be regarded as the process of searching edge boundary building blocks with increasing order. The true power of such an iterative evolution can be appreciated from the concept of "Implicit Parallelism". Each polygon shape with l sides is an instance of 2^{l-1} edge boundary schemata. Therefore, for an evolution with a generation size N, even though only N polygon shapes directly participates in the evolution during each iteration, there are somewhere from 2^{l-1} to $N2^{l-1}$ edge boundary schemata which are implicitly involved in the parallel schema processing. It is such a large amount of available schemata flow that provides the opportunities for an evolutionary process to effectively explore the polygon shape space. Any successful evolution also requires the search effort between exploration and exploitation to be balanced in the sense that every new exploration direction taken is based on proper exploitation of currently preserved edge boundary schemata. This turns out to be rather challenging due to the polygonal geometry

which is likely to cause a high level of epistasis (defined below) as well as deceptions in most applications. Any epistasis will increase the performance variances of edge boundary schemata, which aggravates the sampling errors. Any deception will mislead the exploration direction, which traps the evolution to a local optimum. So it is worthwhile to give further analysis of each of them.

The phenomena of epistasis here refers to the situations where one portion of the edge boundary affects the performance contribution of another portion. Further illustrations of epistasis must be based on specific applications where the performance evaluations are given. As an example, Figure 6.24 illustrates a 2D wet etching process. The target corner is the desired corner to be etched, which is bounded by two (111) planes and a (100) plane in between. The performance evaluation is given by the shape match between the actually etched corner and the target corner. "Boundary 1" and "Boundary 2" shows the edge boundaries of two tried masks. The normal vector of each edge boundary indicates the associated etch rate and etch direction. The only difference between the two edge boundaries comes from Edge $i-1$ whose normal vector is aligned with the <111> direction in "Boundary 1" and is aligned between <111> and <100> in "Boundary 2". The potential performance contribution of Edge i is to form the (100) plane of the target corner. However, Edge i fails to give such a contribution in "Boundary 1", and succeeds in "Boundary 2" simply because the different formations of Edge $i-1$. So in this case, the performance of Edge i is affected by Edge $i-1$ and thus there is epistasis between the two edge boundaries. Not all the edges always have the same level of epistasis. In this example, both Edge $i+1$ and Edge $i+2$ are aligned with (111) plane, which are quite stable during the etching process regardless how the rest of edges are formed. Therefore, those two edge boundaries are almost free from epistasis. In general, the epistasis tend to decrease for the edge boundaries including more number of edges. The epistasis of an edge boundary schema is referred to the epistasis of its similar edge boundaries. Therefore, high-ordered edge boundary schemata tends to contain lower level of epistasis.

The deception analysis described in section 3.7 can be applied here. Deceptions are mainly caused by some low-order deceptive edge boundary building blocks which exist from the early stages of an evolution and "consistently" having new individual instances

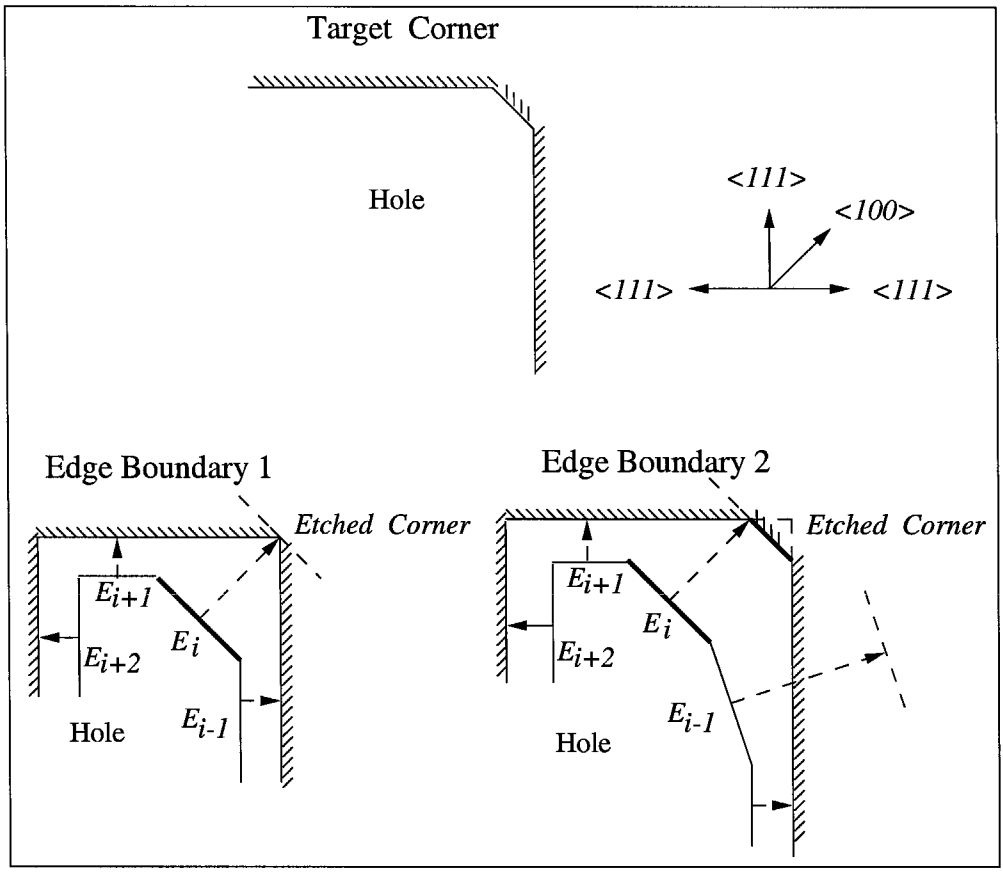


Figure 6.24: An Example for Epistasis in Edge Boundaries of a Mask-layout.

with competitive performance. Such performance-rewarding further attracts stronger exploitation and eventually misleads the entire evolution towards sub-optimal high-order building blocks. The individuals here are polygon shapes. The distance between two individuals is the mismatch between the two polygon shapes. There are mainly two types of local optima. One can be imagined as a local hill and the other as a local cliff.

For a hill-like local optimum, its neighborhood contains some polygon shapes that are closer to the global optimum. In other words, from the top of the local hill, it is possible to immediately "walk" closer to the global optimum. For a cliff-like local optimum, all its neighborhood polygon shapes are further away from the global optimum, which means that there is no immediate step from the local optimum towards the global optimum. This leads to a hill-like local optimum, and is referred to as a deceptive hill. The deception associated with a cliff-like local optimum is called as a deceptive cliff. Figure 6.25 gives

a schematic illustration of the both types of deceptions. G is the global optimum. LH is the local optimum trapped by a deceptive hill and LC is the local optimum misled by a deceptive cliff. Two dotted trajectories illustrate the misled searching paths by the two local traps. As shown through the first searching path, deceptive hills can mislead the exploration region away from the global optimum. In other words, as the search "climbs" over the local hill, the best found individual becomes more and more away from the global optimum and thus, the search direction is completely wrong. In this case, the defined edge boundaries of the deceptive building blocks usually carry little or no similarity to any portion of the global polygon shape. The persistence of such premature edge boundaries will restrain the future evolution of individual polygon shapes towards the global optimum. The attraction of performance-rewards is likely to be rather strong to propagate the such "alien" schemata. Therefore, if the evolution process is trapped by a deceptive hill, premature convergence is likely to occur.

Deceptive cliffs, on the other hand, will consistently lead the search process towards the global optimum, as illustrated by the second search path in the Figure 6.25. However, due to edge boundary misalignment, the evolutionary process is trapped to a local polygon shape unable to move closer to the global optimum.

The two cases illustrated in Figure 6.21 are the examples of deceptive cliffs. In this case, the deceptive building blocks usually define the boundary features that match a major part of the global polygon shape however, with a different number of edges. The deceptive attraction of rewarding better is mainly due to this type of major feature matching, which is likely to grow new polygon shapes closer to the global optimum. Since the number of edge boundaries is fixed throughout the evolution process, the initial edge boundary misalignment will eventually constrain any further growth of the global optimal features. In general, the best evolved polygon shape is likely to be fairly close to the global optimum, and deceptive cliffs are likely to locate on the global optimum hill as shown in Figure 6.25. If such deception occurs, it is likely that the evolution process has detected the global optimum hill and started "climbing". Therefore, the deceptive cliffs usually occur during the middle stages of an evolution.

It can be seen that the harmful effects of deceptive hills are more likely to occur in indi-

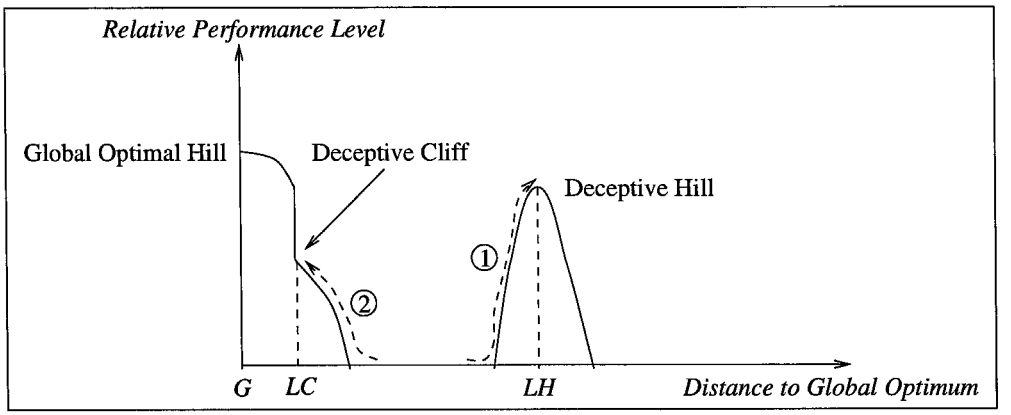


Figure 6.25: An Illustration on Two Types of Deceptions.

vidual applications where a performance evaluation is provided. A performance evaluation could strongly reward the performance of any kind of edge boundary building blocks. On the other hand, deceptive cliffs tend to commonly appear close to a global optimum, which exist in most applications as long as a relatively well formed global optimum hill can be identified during the searching (as compared to the situations like "needle-in-a-haystack", where the solution space contains many locally isolated optimums). So it is worthwhile to construct a virtual application which only contains deceptive cliffs. By using such an application as a testbed, evolutionary strategies to battle with deceptive cliffs can be developed and demonstrated, which can be used in real applications. One such instance is constructed as follows. A target polygon shape is specified as the unique global optimum. The performance of each individual (polygon shape) is determined by its distance to the target shape. The closer an individual is to the target shape, the higher performance value it will receive. Because the performance is directly tied to the distance to the global optimum, no building blocks will mislead the searching away from the global optimum and thus no deceptive hills exists in this application. Such application is called a hill-free test.

In order to demonstrate the hill-free test results, the convergence has to be measured to show the evolutionary process. Generational crossover disruptiveness R_d is introduced to provide such measurement. The connection between the convergence and R_d can be appreciated starting with the following definitions:

Definition 6 Paired Crossover

Let I be an individual space and p_1, p_2, c_1, c_2 be four different individuals in I. A Paired Crossover is defined as the operation of generating c_1 and c_2 through twice independent single crossover operations between p_1 and p_2. p_1 and p_2 are called Parents. c_1 and c_2 are called Children.

Definition 7 Paired Crossover Disruptiveness

Let function $D : I \times I \rightarrow R$ be a metric distance function. The Crossover Disruptiveness of a paired-crossover is defined through the following formula:

$$r = \frac{min(D(c_1, p_1), D(c_1, p_2)) + min(D(c_2, p_1), D(c_2, p_2))}{2}.$$

where

$$\begin{aligned} r &= \textit{the paired crossover disruptiveness} \\ p_1,\ p_2 &= \textit{parents of the paired crossover} \\ c_1,\ c_2 &= \textit{children of the paired crossover.} \end{aligned}$$

The r defined above essentially measures the disruptive effect from a paired crossover through the average distance between the parents and their children. Such distance also serves as an indication of average feature disruption and thus reflects the level of exploration provided through the crossover.

Definition 8 Generational Crossover

Let G be a set of continuously iterative generations within an evolutionary process and G_i be the ith generation. Assume all the generations in G have the same size N, which is an even number. The Generational Crossover of G_i is defined as the operation with $N/2$ independent paired crossovers whose parents are drawn from G_i without replacement and whose children altogether form G_{i+1}.

Definition 9 Generational Crossover Disruptiveness

Let C denote a generational crossover of a generation with N individuals. Also let c_i be all the paired crossovers of C, and r_i be the crossover disruptiveness of c_i, where $i = 1, 2 \ldots N/2$. The Generational Crossover Disruption of C is defined as:

$$R_d = \frac{\sum_{1 \le i \le N/2} r_i}{N/2}.$$

R_d is simply the average value of all r_i, which can be interpreted as the average distance between an individual in the parent generation and its corresponding children in the next generation. This distance reflects the average amount of feature disruption between two generations and thus indicates the level of exploration effort undertaken by the generational crossover. The connection between the crossover disruptiveness and the evolution convergence can be inferred from the following chain effect: a disruptive crossover provides a strong level of exploration which leads to a large feature difference between the two consecutive generations which further indicates a weak convergence. Such reverse relationship indicates that the convergence of an evolution process can be demonstrated through a function plot of R_d vs. iterations, and as the process converges, R_d decreases towards zero.

6.4.3 Selection Scheme

The main task of a selection scheme is to determine the population pool for the next iteration. Consider that each individual represents instances of many implicit schemata, and such individual selection process also determines the presence of schemata in the future evolution. So the main goal of each selection scheme is to differentiate "good" schemata from "bad" ones based on the previous iterations and make sure that those "good" ones will be carried by the selected individuals into the next iteration. One important criterion used to differentiate schemata is the performance of individuals. It is believed that well-performing individuals are likely to contain "better" schemata than those poor-performing ones. By selecting and duplicating well-performing individuals, those expected "good" schemata will have more opportunities to be exploited and grow during future iterations. Accordingly, by eliminating some bad-performing individuals, the expected "bad" schemata will have little

chance to be exploited or survive. In this way, the individual selection bias, also referred as selection pressure, effectively induces bias on the level of schemata exploitation and further leads to the control of evolutionary convergence. So the key here is to develop a selection scheme with a conveniently adjustable selection pressure.

The entire selection scheme is divided into two stages starting with a selection stage which is followed by a sampling stage. During a selection stage, a sampling rate is assigned to each individual in the current generation based on the performance values. In a sampling stage, each individual is sampled according to a sampling rate to form the new population pool as the next generation. Since the two stages are completely separated, various selection schemes can be constructed with different combinations of them. In addition, the selection pressure is directly reflected through individual's sampling rates and thus it is solely controlled in the selection stage. A parametric rank-based selection algorithm is constructed during the selection stage to assign the sampling rates based on individual's performance rank. At the beginning, all the individuals are ranked from 1 to N according to their performances, where N is the generation size. Then, the sampling rate for each one is determined by the following formula:

$$S(i) = 1.0 + \frac{i - \frac{N}{2}}{1 - \frac{N}{2}} S_b \,, \tag{6.9}$$

where

$$\begin{aligned} i &= i^{th} \text{ individual } i = 1,\ 2 \ldots N \\ S(i) &= \text{sampling rate of individual } i \\ S_b &= \text{selection bias.} \end{aligned}$$

The physical meaning of the selection bias S_b is the excess amount of sampling rate awarded to the top-ranked individual as compared to the middle-ranked one. From Equation (6.9), the sampling rate of the middle-ranked individual is always 1.0. Figure 6.26 illustrates three plots of function (6.9) under three different settings of S_b to show how the selection pressure can be controlled. The left plot shows the case when no selection bias

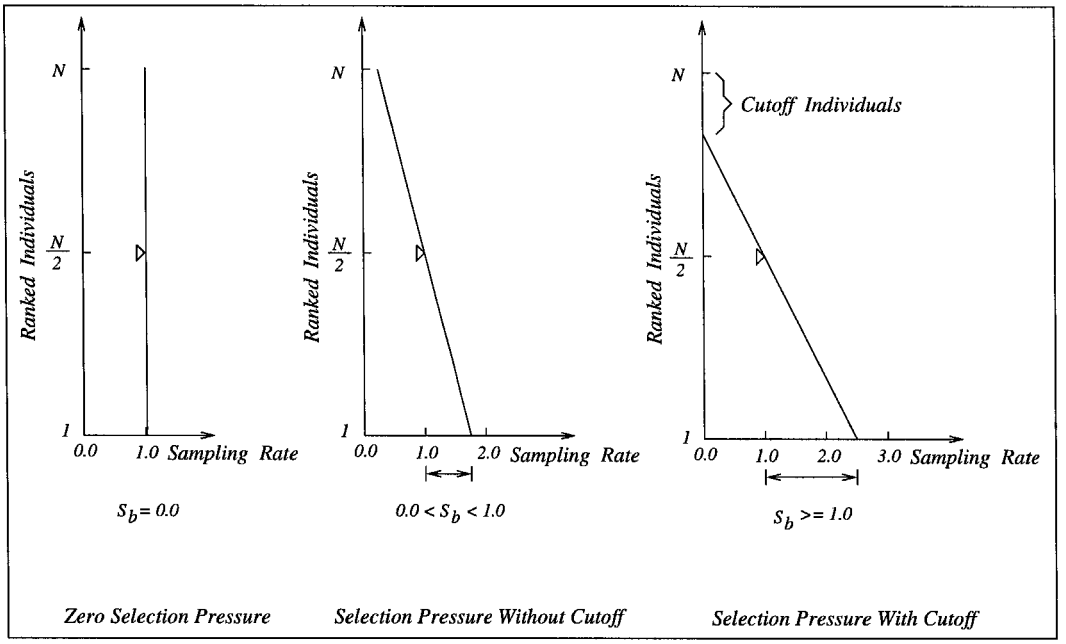


Figure 6.26: Selection Pressure Controlled by Selection Bias S_b.

is applied and therefore every individual's sampling rate is 1.0, which means no selection pressure exists. The middle plot shows the case where the selection bias is between 1.0 and 2.0. In this case, every individual still receives a positive sampling rate, but with slight bias towards the better ranked individuals, which indicates a moderate amount of selection pressure. The right plot shows the case when the selection bias is greater than or equal to 1.0, which means the top-ranked individual is almost surely to be duplicated in the next generation. Any duplications starting from the top-ranked individual will cause eliminations beginning with the worst-ranked individual. In this case, as shown in the plot, selection pressure is high enough to cut off some poor-performing individuals. In this way, selection bias S_b can be used as a method to effectively adjust the selection pressure during the course of evolution so as to control the convergence. The stochastic universal sampling algorithm introduced in section 3.8 is used here in the sampling stage to eliminate sampling bias and reduce sampling spread [8].

The effectiveness of the selection bias can be appreciated through a hill-free test. The results are illustrated in Figure 6.27. The top four function plots of R_d vs. iterations show the evolution processes under four different settings of S_b. The bottom four pairs of polygon

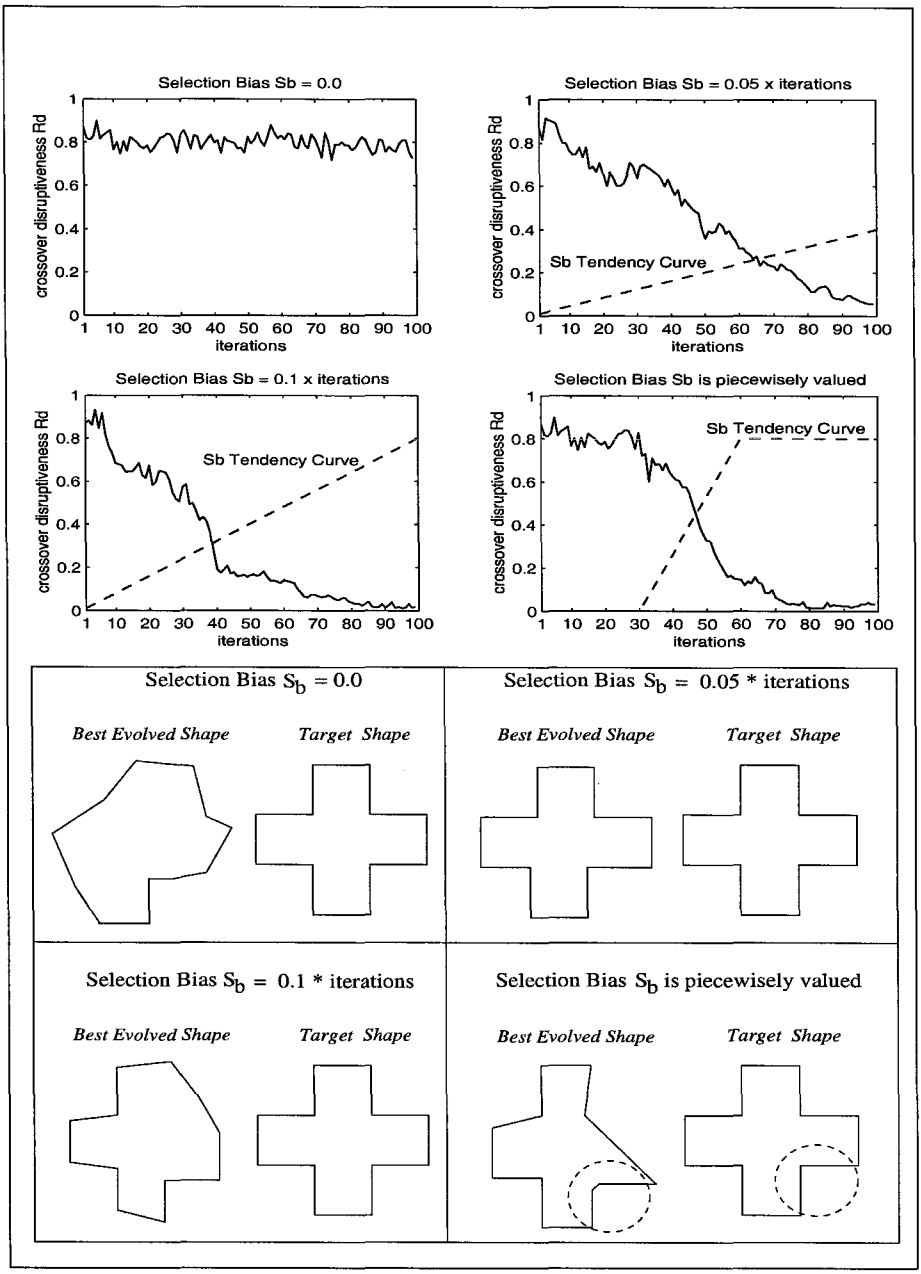


Figure 6.27: Effect on Convergence From Selection Bias S_b.

shapes show the corresponding evolution results for each S_b. All evolution processes have the same target shape and use random pairing as their genetic operation. The generation size is fixed at 50. When $S_b = 0$, no selection pressure is applied and thus no convergence is expected. The best evolved shape is almost like a random generated one and bears no obvious similarity to the target shape. The remaining three cases have nonzero S_b all of which lead to the convergence. The dotted line in each function plot indicates the relative change of S_b values during each evolution process. Consider both the top right and lower left cases, their selection biases are gradually increased as the evolution goes on. However, the slope of the top right S_b curve is smaller than the lower left one. Therefore the selection pressure is increased more gently in the top right case whose evolution process converges more slowly, just as expected. In the lower right case, the strategy to set S_b is such that the evolution process will initially have an aggressive exploration, and then quickly converge the search region towards the best explored outcome and finally exploit the converged region. Therefore, S_b is increased piecewisely. During the first 30 iterations, S_b is zero and then is quickly increased between 30 and 60 iterations and afterwards, it stays unchanged. The resulting convergence curve is almost ideal. However, by comparing the evolved results, the top right case gives the best result, which is closest to the target shape. Such outcomes provide some useful clues for a proper control on the evolution convergence, which will be recapped in a later section.

6.4.4 Genetic Operation

The stage of genetic operations is the process of using genetic operators mainly including genetic crossover and mutation to produce a new generation from an existing one. During an evolution, genetic operations are used right after a selection scheme for every iteration. From the point of view of schemata, it is the genetic operations that explore and exploit all the available schemata provided by any selection scheme. Some schemata are likely to have more chances to be exploited than others because they have more instances among the selected individuals. Such distribution bias on the available schemata is mainly provided by the selection pressure. The exploitations among these schemata are the key to ensure evolution convergence. So the selection pressure has an important influence on the over-

all performance of the evolution process. However, using the selection pressure alone to control the convergence may not be effective for two reasons. First, the selection pressure only increases the likelihood for the expected amount of exploitations to occur, while what really counts is the actual pairing of individuals. If without any strategy on the genetic operations, and only random pairing is used, the amount of exploitation versus exploration can not always be achieved as expected. Second, due to sampling errors, the selection pressure can not always reflect the schemata performances correctly, which will induce local traps to mislead the evolution to converge to a local optimum. Under such cases, an extra way to control the convergence is needed to make a self-correction. With the above two reasons, the design goal of the genetic operations is to provide useful methods which can be finely tuned to control the rate of convergence during the generation of new individuals.

With the above-illustrated connection between an evolution convergence and the disruptiveness of a generational crossover R_d, the control over the convergence can be converted to the control over R_d. Note that R_d is mainly affected by two factors. One is the crossover operator and the other is the actual pairing of the parent individuals. In most cases, the same crossover operator will be implemented throughout an evolution process, therefore once the design of a crossover operator is fixed, the crossover disruptiveness R_d is only affected by how the individuals are paired among the parent generation. The pairing algorithm presented here is based on the assumption that the disruptiveness of the crossover operator used is inversely related to the distance between the parents. In other words, as the distance between parents gets smaller, the crossover will tend to exploit more and thus exhibit weaker disruptiveness. Such an assumption can be satisfied in most applications with the proper design of the distance function.

The algorithm starts with a given selected generation with some of the individuals duplicated. During each pairing, two individuals are determined to participate in a paired crossover and withdrawn from the given generation without replacement. The pairing process stops once the generation is empty. Figure 6.28 illustrates the step to determine each paired parents. All the individuals in the selected generation are stored in the individual slots. Due to the duplications and eliminations in the selection stage, some individual slots may have more than one individual and some may be empty, such as the one marked with

a "$\times$" in the figure.

The procedure starts by picking an individual in the smallest nonempty individual slot. Such an individual is named as "Groom", which will directly be one of the paired parents. Then, N_b individuals are randomly picked from the remaining nonempty slots. Note that the individuals can be duplicated. Such N_b individuals together are named as "Bride Pool" and each one is called a "Bride." In the figure, the "Groom" is individual G and the "Brides" are individual $B1$, individual $B2$, etc. Then the distance between the "Groom" and each "Bride" is calculated and sorted in a nondecreasing order which is marked "Sorted Bride Pool" in the figure. Finally, a distance bias ratio R_b is used to select the final "Bride" from "Sorted Bride Pool" as the other parent. As shown in the figure, the finally selected "Bride" is individual $B2$ and the corresponding array index in the "Sorted Bride Pool" is the smallest integer of $R_b * N_b$. There are two possible exceptional cases. One is that at some point of the crossover operation, the individuals left in all individual slots may be less than N_b, under which cases, "Bride Pool" will just have to take what is available; the other is that there may be a single nonempty individual slot left at the end of the pairing, in which case, all its individuals will participate in mutation alone.

The above pairing algorithm automatically avoids the random chance that the same individuals are paired together. In addition, the algorithm also provides two methods to control the distance between each paired parent. One is the size of "Bride Pool" N_b, and the other is the distance bias ratio R_b. The distance bias ratio controls the bias between exploitation and exploration. If $R_b = 0$, the "Bride" with the closest distance in "Bride Pool" is always selected and thus has strong bias towards exploitation. If $R_b = 1$, the "Bride" with the largest distance in "Bride Pool" is always selected and thus has strong bias towards exploration. The "Bride Pool" size N_b gives the rate of bias defined by R_b. Larger N_b brings more competition among the "Brides" and thus increases the bias towards either exploitation or exploration. In particular, with $N_b = 1$, the bias effect completely vanishes, which is just like random pairing. Therefore, the distance bias ratio combined with the "Bride Pool" size essentially controls the direction and level of convergence through its effect on the balance between exploitation and exploration. However, the control power from R_b and N_b relies on feature competitions among the individuals in "Bride Pool", which means the

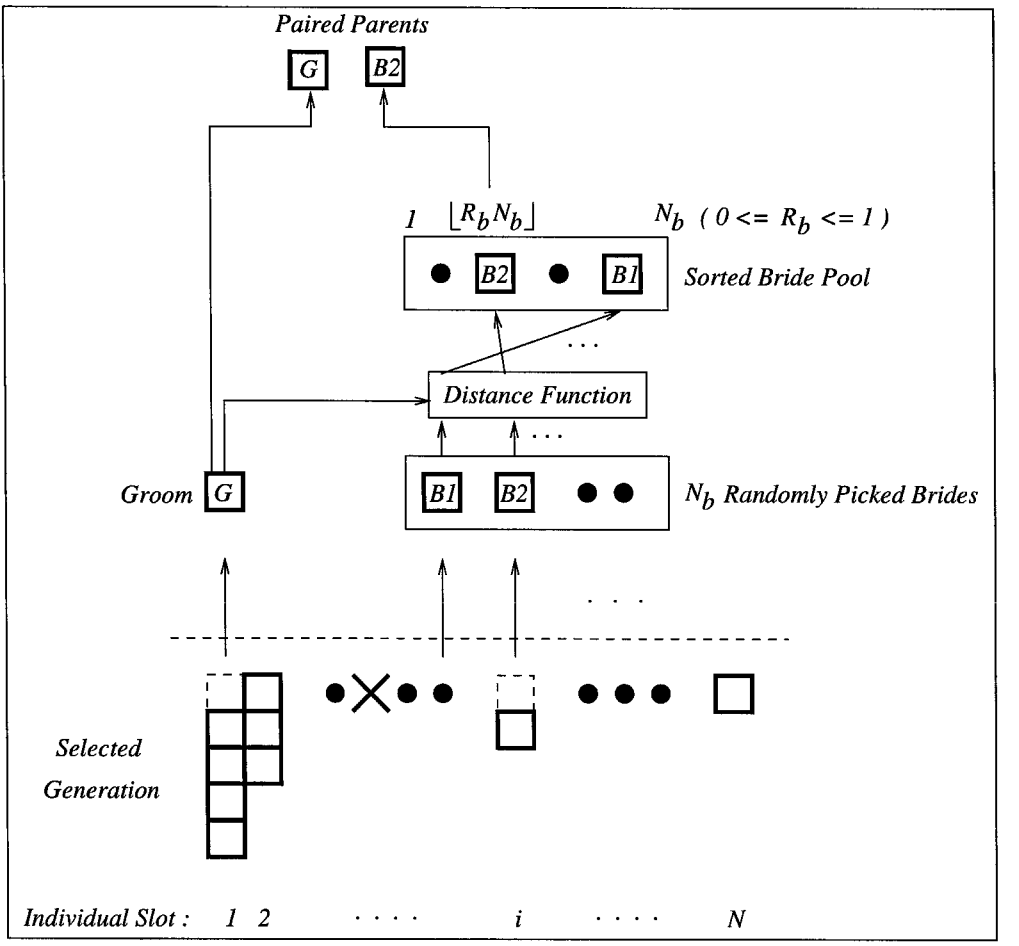


Figure 6.28: The Step to Determine Each Paired Parents in Pairing Algorithm.

varieties of individuals need to be maintained in a generation. Therefore, the convergence control through these two handlers is more effective during the early and middle stages of an evolution process where individuals are expected to have different features.

Two hill-free tests are carried to demonstrate the effectiveness of the convergence control through a combination of the distance bias ratio R_b and the "Bride Pool" size N_b. Both tests are the same as the previous one for the selection bias S_b except that now S_b is fixed at zero to turn off the selection bias effect and the comparisons are made for different settings of R_b and N_b. Figure 6.29 and Figure 6.30 show the results when $R_b = 0$ and $R_b = 1$ respectively. The structure of each figure is exactly the same as Figure 6.27. In both figures, N_b is increasingly set to 1, 10, 25 and 50. Note that $N_b = 1$ is equivalent to random pairing whose evolution plot and result are directly repeated from Figure 6.27 to serve as the com-

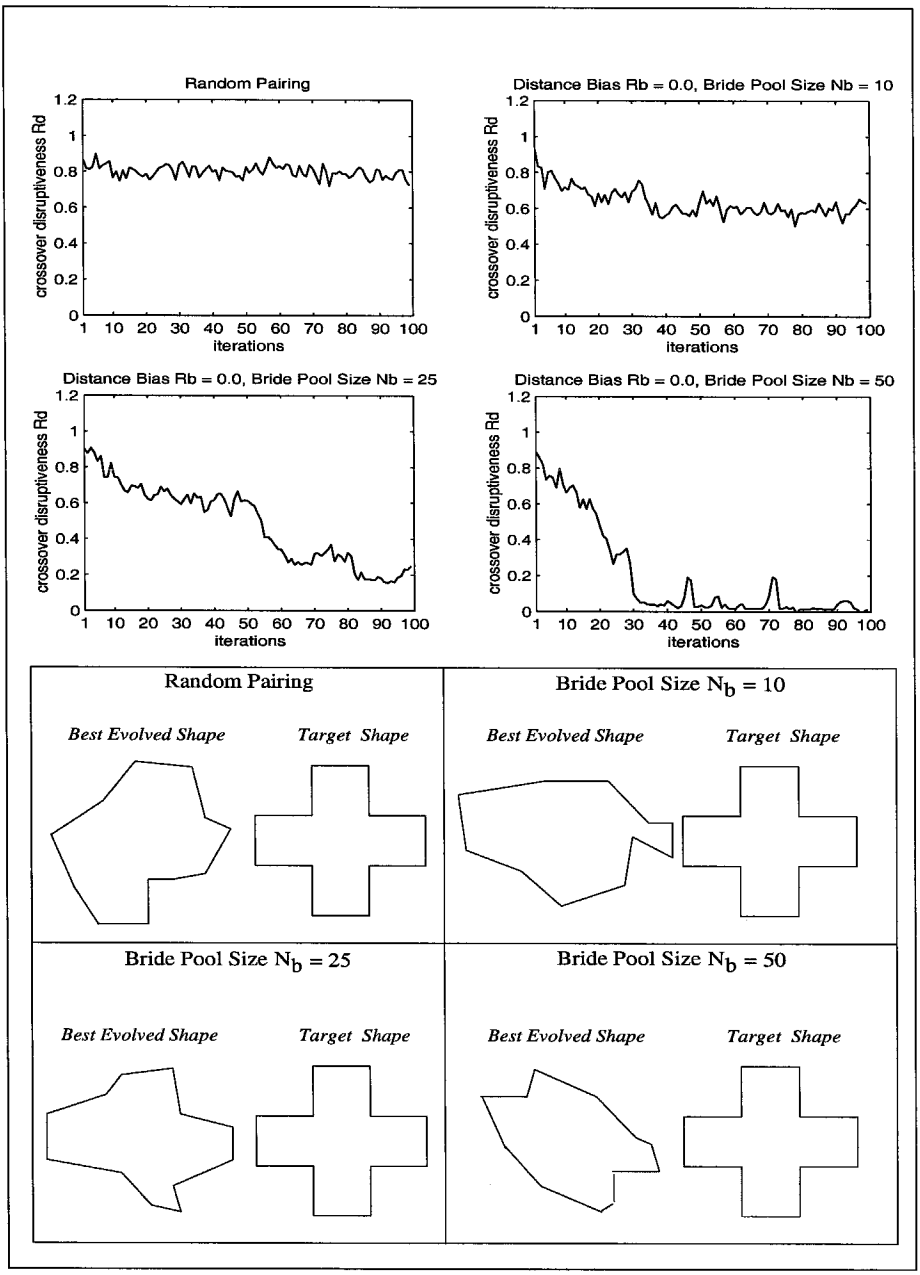


Figure 6.29: Converging Effect on Convergence From Smallest Distance Bias Ratio R_b Combined with Various Bride Pool Size N_b.

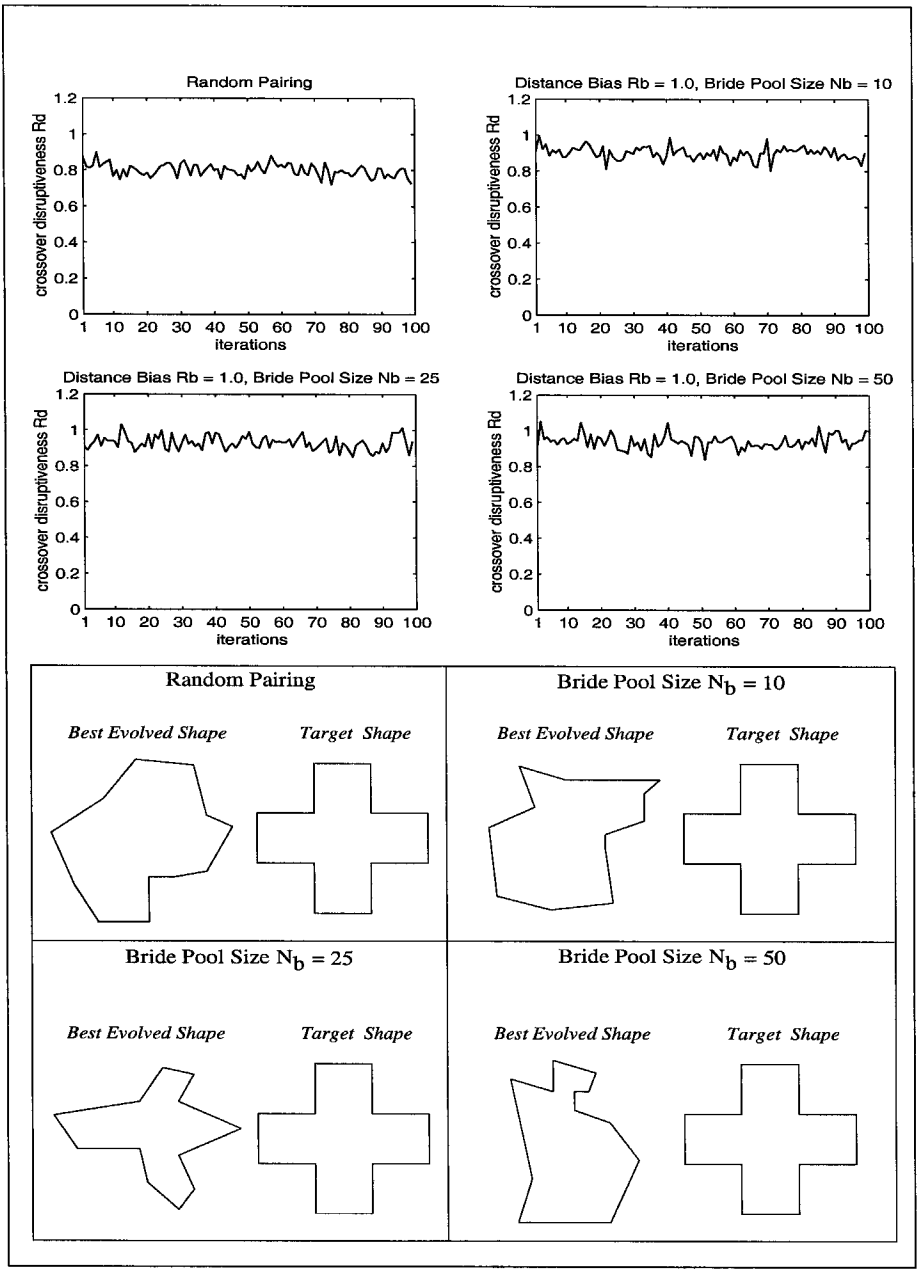


Figure 6.30: Disruptive Effect on Convergence From Largest Distance Bias Ratio R_b Combined with Various Bride Pool Size N_b.

parison basis. In Figure 6.29, with R_b equal to 0, the remaining three evolution processes start converging because of the complete bias towards exploitation during each genetic operation. In particular, as N_b increases, the evolution converges faster because of stronger exploitation bias. In Figure 6.30, with R_b equal to 1, the remaining three plots indicate the strong disruptiveness throughout the evolution processes. In addition, larger values of N_b also increase the overall level of disruptiveness. In both figures, none of the presented evolution results is close to the target shape, which is also expected because there is no selection pressure here and all the genetic operations are carried blindly for both exploitation and exploration.

6.4.5 Evolution Strategy

Even though each of the three handlers provided by the above selection scheme and genetic operation can be used alone to control the convergence of an evolution process, none of them is able to create enough effort to the obtain proper convergence. With selection bias alone, the evolution is purely driven by rewarding performance, which makes the evolution outcome rather sensitive to deceptions where individuals have misleading performances. With either distance bias ratio or "Bride Pool" size alone, none of them is capable of bringing performance convergence as seen in both Figure 6.29 and Figure 6.30. With the use of all three together, however, the chance to achieve a desirable evolution convergence can be much increased. Selection bias controls the selection pressure based upon an individual's performance, which can serve as the vital force to ensure performance convergence, while distance bias ratio and "Bride Pool" size controls the local balance between exploitation and exploration, which can provide effective effort to fight deceptions. Therefore, selection bias can be considered as the evolution strategy maker to create a desirable convergence profile. Both distance bias ratio and "Bride Pool" size serve as the coordinators to ensure the effective control from the selection bias.

A typical evolution profile can be divided into three stages with different convergence behaviors as shown in Figure 6.31. During the early stage, little convergence is obtained due to the dominant exploration effort. In the middle stage, exploitation effort increases and exceeds exploration so that the search region becomes narrower and the evolution is grad-

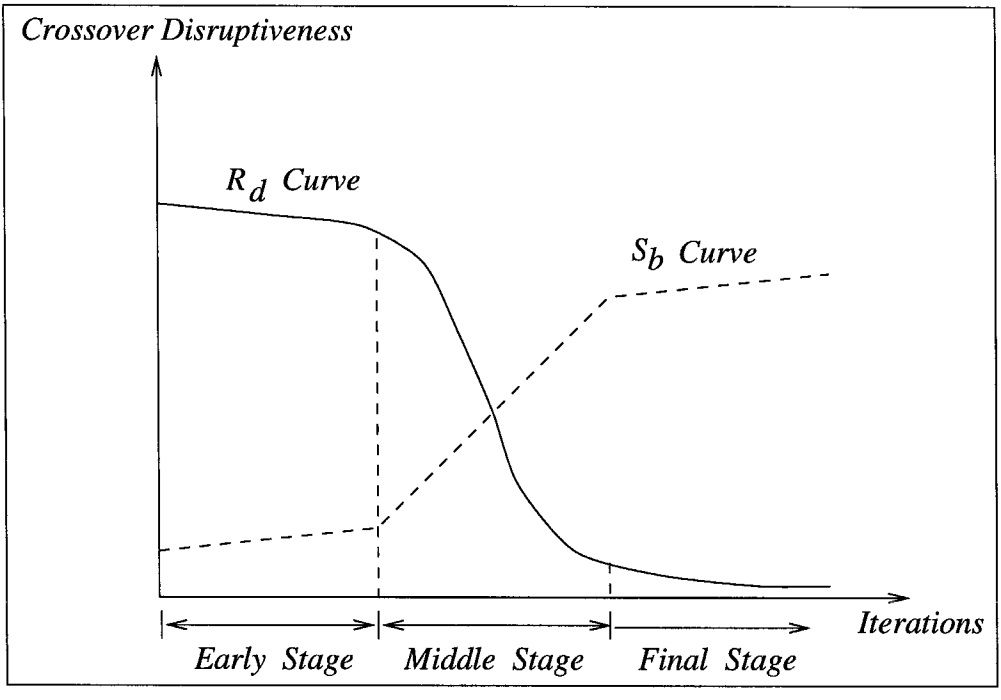


Figure 6.31: Different Convergence Stages of An Evolution Process.

ually converged. The final stage is where exploitation dominates and the narrowed search region is further converged to a single best found individual. However, such a profile does not always indicate a successful evolution. In Figure 6.27, all three cases with nonzero selection biases have the convergence profiles more or less close to what is shown in Figure 6.31. However, only the top right case gives a good evolution outcome, while the other two fail. In particular, the case with piecewise linear S_b leads towards a local optimum with misaligned edge boundaries indicated by dotted circles. Therefore, without some proper strategy to control the profile, an evolution process can easily fail. In the following, several strategies corresponding to different evolution stages are presented to serve as general guidelines for real applications.

One particular major concern during the initial stage is how to preserve emerging edge boundary building blocks. Due to the richness of geometry, the polygon shapes in the initial generation are rather different from each other, which leads to highly disruptive exploration. So any low-order schemata are likely to be destroyed. Without conservative protection on the good ones, the initial evolution will become inefficient. However, the selection pressure

from selection bias S_b is not appropriate here, because duplication will cause too much bias on particular schemata, which is likely to induce premature convergence. Instead, the elitist selection described in section 3.8 is a better choice. In such a selection method, the individuals with good performance will always be preserved. Since it is at the initial stages, the varieties of the preserved individuals are expected to be maintained, which means each individual is likely to represent a different promising search region rather than all individual together represent a preconverged one. Under such a conservative selection scheme, in order to cover more area, aggressive exploration becomes even more desirable. To ensure that, the distance bias ratio is set as 1 and the "Bride Pool" size is assigned to be the generation size.

During the middle stage, fewer and fewer winners start competing with each other in all the preserved search regions. This is the place where the evolution process is likely to be misled by deceptions, especially deceptive cliffs. Several protective strategies are taken to reduce the chance of being trapped. First, the selection bias S_b is turned on to replace the elitist scheme so that the selection pressure can be effectively controlled. The S_b value is increased slowly within the range from 1 to 4 in most cases so that the evolution can converge gently. As shown in the example in Figure 6.31, the major reason that the top right case out-performs the lower left and lower right cases can be seen by comparing their convergence profiles, the top right one has gently decreased slope during the middle stage from iteration 30 to iteration 60, while the other two have rather steep downward slopes. Slowly increased selection pressure can maintain a certain level of exploration throughout the middle stage and therefore increases the chance for the evolution to make a self-correction in case of being trapped. Second, the distance bias ratio R_b is gradually decreased from 1 to 0 and the "Bride Pool" size N_b is also reduced from the generation size to a small value such as 5. The value of R_b directly controls the balance between the exploitation and the exploration. As R_b decreases, the search effort gradually switches from exploration towards exploitation so as to keep pace with the change of selection pressure. Only small N_b is needed in the end since most individuals are expected to be fairly close to each other. Third, the top-ranked individual is copied and altered with the second kind of mutation described in section 6.3.5. If the individual is modified, it will replace the worst

performing individual in a current generation. This process will effectively increase the chance of breaking possible paths towards potential deceptive cliffs from the beginning.

During the final stage, most surviving individuals are expected to fall in a converging region and therefore are fairly close to each other. A stronger convergence is needed to effectively shrink the region. So, the selection bias is increased towards 8 in most cases so that most exploitation will be carried out among top performing individuals. The control effect from the distance bias ratio and the "Bride Pool" size are not expected because the remaining individuals are expected to be fairly close to each other. The entire evolution can be stopped when the crossover disruptiveness R_d is smaller than a specified threshold value for several consecutive iterations. A typical threshold value is 0.05.

The above strategies have been implemented in the hill-free tests to examine their effectiveness. Two particular cases are presented in Figure 6.32, both of which experience overcoming of deceptive cliffs during the middle stages of the evolution processes. Each case has different target shape. The left one is a cross shape and the right one is a star shape. Both cases have the same generation size equal to 50 and the same elitist selection up to the first 20 iterations, which is followed by the control of selection bias S_b with its tendency curve shown in the top plot. The S_b curve consists of two line segments with different slopes: k_1 and k_2, to control the increasing convergence rate during the middle and final stages respectively. In this case, the first segment covers from iteration 20 to iteration 70 with k_1 equal to 0.05 and the second segment covers after iteration 70 with k_2 set to 0.1. The exact values of S_b are given by the following relationships:

$$S_b = \begin{cases} 0.0 & I < I_e \\ k_1(I - I_e) & I_e \leq I < I_m \\ k_1(I_m - I_e) + k_2(I - I_m) & I \geq I_m \end{cases} \tag{6.10}$$

where

S_b = selection bias

I = iteration number

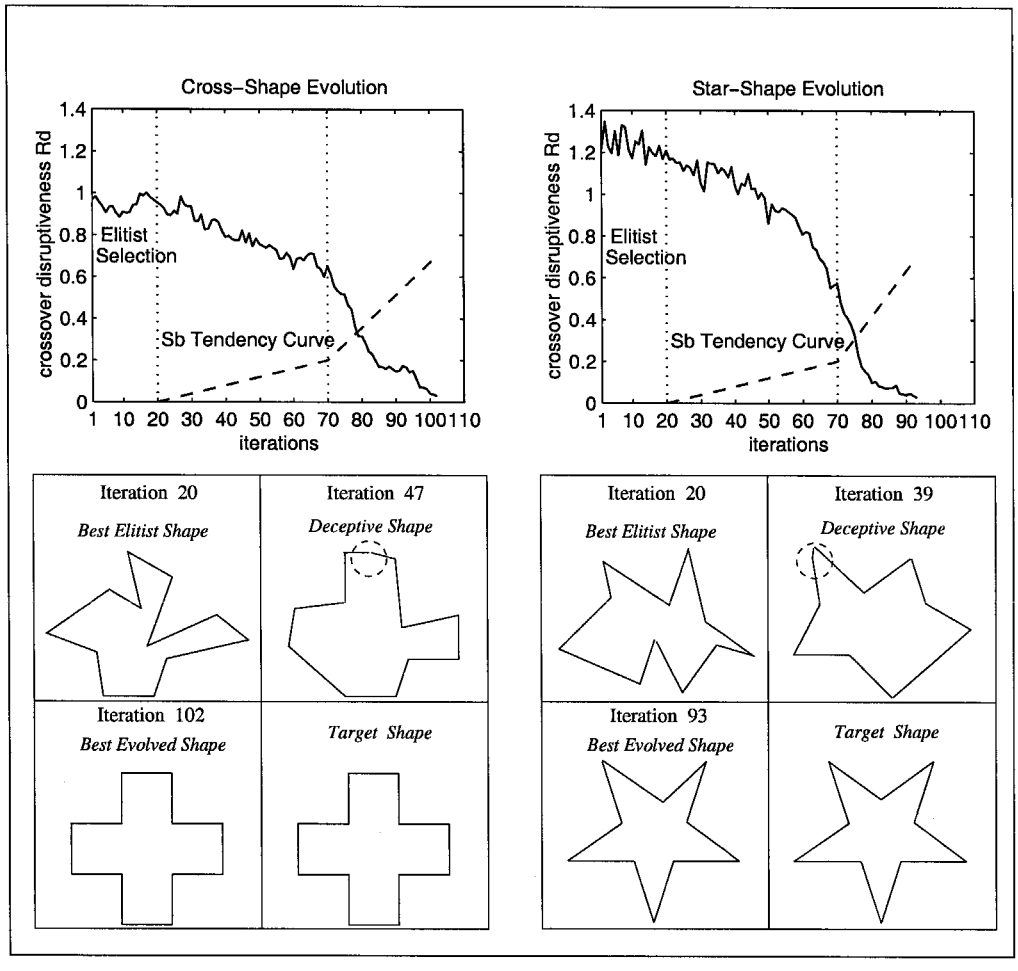


Figure 6.32: Two Hill-free Tests with Deceptive Cliffs Overcome by Developed Evolution Strategies.

$$
\begin{aligned}
I_e &= 20 \\
&= \text{last iteration of the initial stage} \\
I_m &= 70 \\
&= \text{last iteration of the middle stage} \\
k_1 &= 0.05 \\
&= \text{slope of selection bias segment in the middle stage} \\
k_2 &= 0.1 \\
&= \text{slope of selection bias segment in the final stage.}
\end{aligned}
$$

The exact values of R_b and N_b are given as:

$$R_b = \begin{cases} 1.0 & I < I_e \\ 1.0 - \frac{I - I_e}{(I_m - 10 - I_e)} & I_e \leq I < I_m - 10 \\ 0.0 & I \geq I_m - 10 \end{cases}$$

$$N_b = \begin{cases} N & I < I_e \\ N - \frac{I - I_e}{(I_m - 10 - I_e)}(N - 5) & I_e \leq I < I_m - 10 \\ 5 & I_m - 10 \leq I < I_m \\ 1 & I \geq I_m \end{cases} \tag{6.11}$$

where

$$\begin{aligned} R_b &= \text{Distance bias ratio} \\ N_b &= \text{“Bride Pool” size} \\ N &= \text{generation size.} \end{aligned}$$

The convergence curves of both evolutions are shown in the top plots. The vertical dotted lines separate the three stages. Both curves exhibit strong exploration in the initial stage, gentle convergence during the middle stage and quick finish-up at the final stage, all of which are expected. In particular, the bottom part of the figure shows the best searched outcomes at each critical iterations. Iteration 20 is right after the elitist selections. In each case, the approximate resemblance between the presented shape and the target shape indicates the best promising search region defined at that stage. Deceptive cliffs appears in both cases. In the left case, the deceptive shape is the best evolved shape at iteration 47. The edge boundary misalignment is highlighted in the dotted circle, which is caused by the direction angle alignment between two consecutive edges. In the right case, the deception occurs at iteration 39 in which case the edge boundary misalignment is caused by very small edge stuck at the tip of the circled corner. Fortunately, both cases successfully overcame the deceptions in the end, which can be seen from the best evolved shapes compared to their

Chapter 7

Mask-layout Wet-etching Synthesis Test

7.1 Overview

An evolutionary algorithm has been constructed by incorporating a bulk wet etching simulator along with a performance evaluation scheme into the application module. The algorithm is used to synthesize mask-layouts for the simulated bulk wet etching process so that the etched shape closely matches a specified target shape. A forward etching simulator called SEGS and introduced in Chapter 4, is used to produce an etched shape for each candidate mask-layout. The purpose of the test is to demonstrate the effectiveness of the evolutionary techniques developed here.

7.2 Performance Evaluation

The evaluation of performance in an evolutionary algorithm is to provide a fitness value for each individual through an objective function. The performance of each candidate mask-layout is measured by the shape closeness between its etched shape and the target shape. So the crucial task here is to obtain an algorithm to do the 3D shape matching between two shapes. As mentioned in Chapter 5, no existing algorithm has been found to efficiently implement general 3D shape matching. Inspired by the output format of the etching simulator SEGS [38], the desired matching of 3D shape is decomposed into a series of 2D shape matches on successive contours. Figure 7.1 shows the output representation of SEGS simulator for a hypothetic etched shape. Each etched shape is represented as a series of

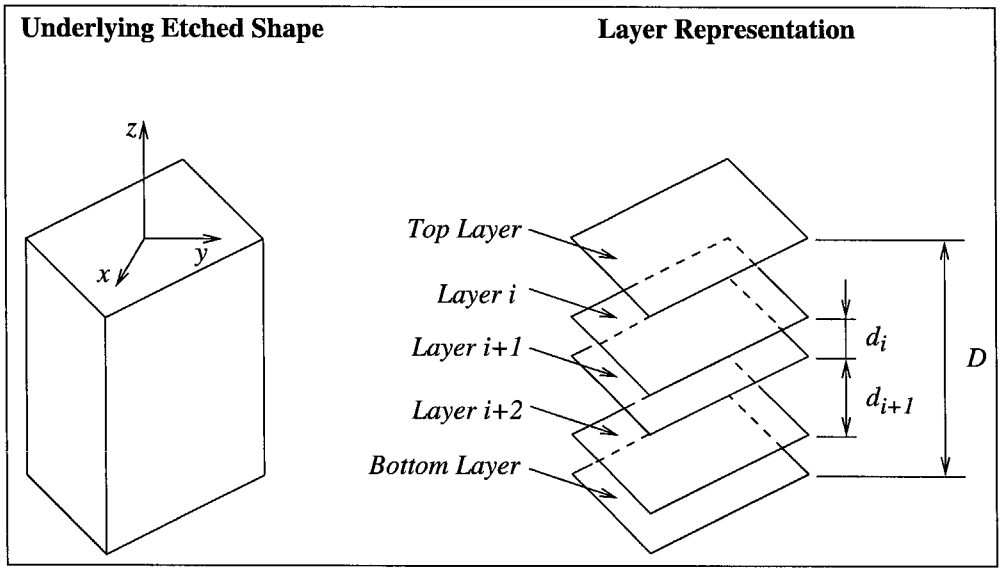


Figure 7.1: Illustration on Layer Representation of 3D Etched Shapes.

horizontal sectional layers. The shape at each layer is represented by a 2D simple polygon. All the polygon layers are vertically stacked. The distance D between the top layer and the bottom layer defines the depth of the etched shape, which is also the etching depth. The vertical distance between any two layers can be different, and is controlled by specifying the corresponding display time step. This layer representation provides enough freedom to describe all 3D etched shapes. Any number of section layers can be used to approximate an underlying 3D object depending on the desired resolution. The vertical location of each section can be chosen to capture critical boundary features, which will focus the evolutionary algorithm on those important features.

With the shapes represented by layers, a 3D shape matching algorithm can be realized by a series of section matchings. This shape matching scheme is applied here with layer representations for both target shapes and etched shapes. Since the layers of a target shape are user specified and those of an etched shape are generated through etching process, it is important to ensure the vertical correspondence between the two sets of layers. To do so, the vertical layer distances in the target shape are used to control the display time steps used in the etch simulator so that the generated etched layers will be located at the same vertical locations.

With that, the 3D shape matching between a target shape and an etched shape is defined as a weighted sum of the polygon matchings between their polygon layers, which can be expressed as:

$$D(T,E) = \sum_{1 \le i \le M} w_i d_i(T,E), \tag{7.1}$$

where

$$\begin{aligned} T &= \text{target shape} \\ E &= \text{etched shape} \\ M &= \text{total number of polygon layers in each shape} \\ w_i &= \text{weight} \\ d_i(T,E) &= \text{polygon mismatch between } i^{th} \text{ polygon layers in both shapes.} \end{aligned}$$

The weights are introduced to give an extra freedom to specify the particular importance of the feature matching of certain layers.

The shape matching algorithm between any two polygons are introduced in Chapter 5. The $L2$ distance function developed by Arkin *et al.*, is used here [22]. Since the absolute orientation of each edge defines the features of the target shape, a slight modification is needed to make the $L2$ distance function rotational variant. The modified form Equation (5.2) can be directly used here. Lastly, since Equation (7.1) measures the mismatch value for each etched shape compared against a target shape, each candidate mask-layout should receive a higher fitness value when its etched shape has smaller mismatch value. An upper-limit value is used to convert every mismatch value into a fitness value. The final fitness function is summarized below:

$$f_k = F_{max} - \sum_{1 \le i \le M} w_i \left\{ \min_{\Theta \in R \; t \in [0,1]} \int_0^1 |\Theta_{E_i}(s+t) - \Theta_{T_i}(s)| ds \right\}, \tag{7.2}$$

where

f_k = the fitness of individual k

T = target shape

E = etched shape

M = total number of polygon layers in each shape

F_{max} = upper bound for all mismatch values

w_i = weight

$\Theta_{T_i}(s)$ = turning function of i^{th} polygon layer in target shape

$\Theta_{E_i}(s)$ = turning function of i^{th} polygon layer in etched shape.

7.3 Test Results

7.3.1 Overview

It is known that because of anisotropic crystal etching, sometimes extra boundary features are needed in the mask-layout to form some critical local geometries of MEMS structures, such as sharp convex corners on a rectangular mesa or peg and a bent V-groove. As an example, Figure 7.2 illustrates two real instances of mask-layouts with compensation structures. The left one was used by Puers and Sansen to compensate the convex corner formed by two $\{111\}$ planes [63] and the right one was used by Zhang *et al.*, [80] to compensate the convex corner in a restricted area surrounded with <110> edges. Even though in both cases, the same type of convex corners are compensated, the particular advantage of the right case is to provide larger etch depth with a similar amount of compensation area. In general, the overall feature size of a compensation structure is proportional to the etch depth and the exact relationship can be analytically developed based on the geometry of the structure as well as the etching rates. For the <100> bar shown in the right case of the figure, Zhang *et al.*, were able to show the relationship between the bar length L and the etch depth. However, if the compensation structure changes, the entire relationship has to be redeveloped, which may also be more complicated due to different geometries. On the

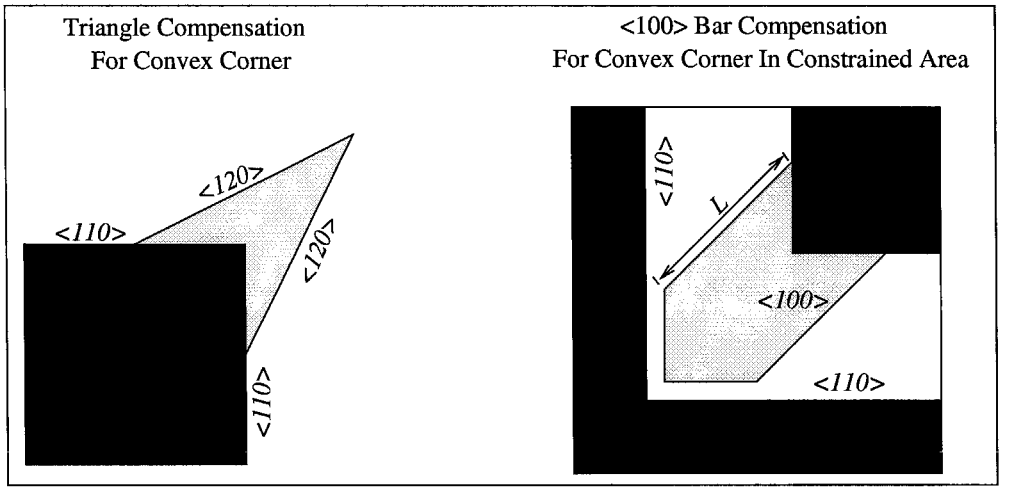


Figure 7.2: Two Examples of Compensation Structures in Real Applications.

other hand, in some applications such as groove-etching, the design area is constrained in a narrow region where a compensation structure with efficient area usage is badly needed. Therefore, an effective way to search for different compensation structures and see the etched outcomes becomes rather desirable. With that, the goal of this section is to show that how the developed evolutionary algorithm can be applied to meet this objective.

In the following tests, all the results are obtained for etching on (100) wafer with KOH etching solutions. The etch rate data are provided by Shikida *et al* [70], and are listed in Table 7.1. The main results of each test are the best evolved mask-layout and its etched shape, both of which are presented together with the mask-layout located on the top layer and highlighted with a darker outline. During each test, the entire etching duration time is equal to the depth of each specified target shape divided by the etching rate in <100> direction. The implementation parameters along with the CPU time is included for each test. Table 7.2 lists the symbols for all the parameters. Among all the parameters, three of them are fixed for all the tests. Both N_b and R_b are defined in (6.11). S_d is fixed as 0.03. All the tests are run on 333MHz Solaris Ultra 10.

Two target shapes are used. One is the solid mesa (peg) shape and the other is the void cross (hole) shape, both of which are shown in Figure 7.3. For the solid mesa shape, because all the sidewalls are (111) planes whose etching rates are the slowest, without any compensation structures, all the four convex corners will be exposed to other faster planes

Orientation	Etching Rate (μm/min)	Orientation	Etching Rate (μm/min)
100	0.629	311	1.065
110	1.292	320	1.285
210	1.237	331	0.845
211	0.983	530	1.273
221	0.586	540	1.283
310	1.079	111	0.009

Table 7.1: Used Etching Rates for 34.0wt.%, 70.9 °C KOH Solution.

Symbol	**Description**
N	Generation Size
I_t	Total Iterations
I_e	Last Iteration of the Initial Stage
I_m	Last Iteration of the Middle Stage
S_b	Selection Bias; See Equation (6.10)
N_b	Bride Pool Size; See Equation (6.11)
R_b	Distance Bias Ratio; See Equation (6.11)
S_d	Stopping Threshold Value of Crossover Disruptiveness
CPU	Total CPU time for each evolution process

Table 7.2: Symbols for Implementation Parameters of Each evolution Test.

and will be immediately attacked especially by the (210),(320) and (530) planes. The same is true for all the four inner convex corners in the void cross shape whose sidewalls are also oriented in the <111> direction. Note that, since all the top-layer edges of the two target shapes are aligned with <110> directions, the dominant etched sidewalls are expected to be aligned with (111) planes because of their slowest etching rates. So by intentionally setting all the (111) sidewalls for the target shapes, the evolution tests are expected to capture

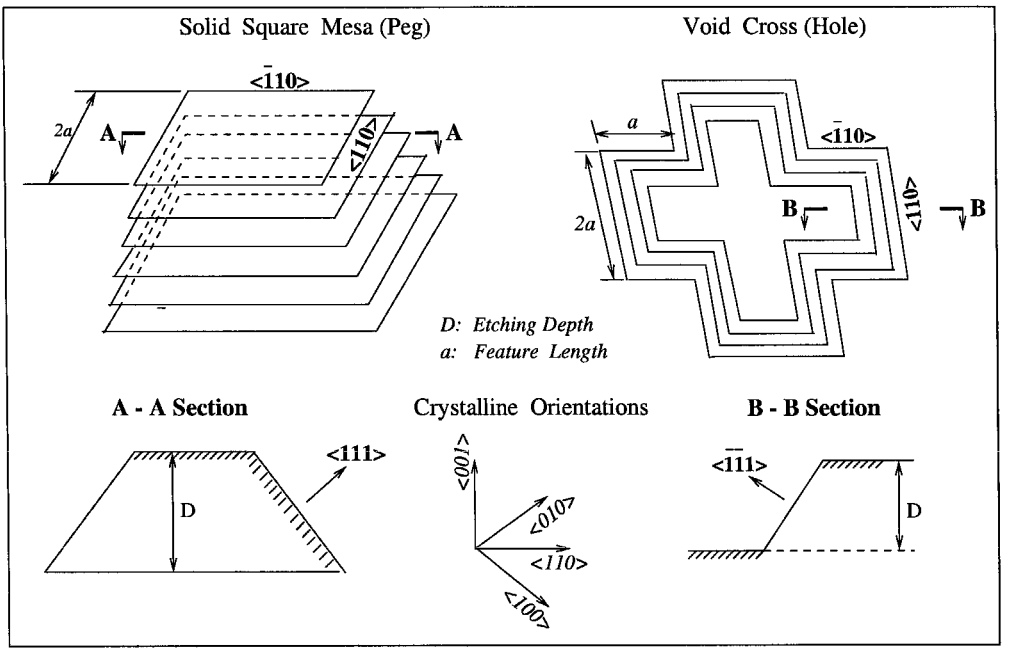


Figure 7.3: Two Target Shapes Used to Test Evolutions of Compensation Structures.

the sidewall feature easily and thus be fully devoted to evolving compensation structures. In addition, both of the target shapes and the etching rates are quadrisymmetric, therefore for each evolved mask-layout, only the edge boundaries within the first quadrant participate in the evolution processes with x and y axes as the constraint boundaries (Appendix D demonstrates an evolutionary process of 8-side mask-layout for an asymmetric target shape). As shown in the figure, the size of each target shape is defined by its feature length and its etching depth. The aspect ratio of each target shape is defined as the ratio of etching depth *vs.* feature length. The solid mesa shape is used to match the left instance in Figure 7.2. And the void cross shape is used to match the right instance in that figure with the particular feature of the area constraint. Note that the void area within the cross boundary limits the size of any future evolved mask-layout. Since the void area is quadrisymmetric, the constraint area for the mask evolution can be effectively taken as the area formed by the positive x and y axes which are aligned in the <110> directions, and the edge boundary of the cross shape in the first quadrant. This constraint area exactly reflects the constraint defined in the right instance of Figure 7.2.

The test results are divided into two sections according to the two target shapes. In each

section, it will be first demonstrated that the evolutionary algorithm is capable of evolving the desired compensation structures. Furthermore, various evolved compensation features for each target shape are presented to show the advantage of using the algorithm to explore alternative mask design solutions. Finally, the algorithm is used to evolve the mask-layouts as the aspect ratio of each target shape is gradually increased.

7.3.2 Solid Square Mesa (Peg)

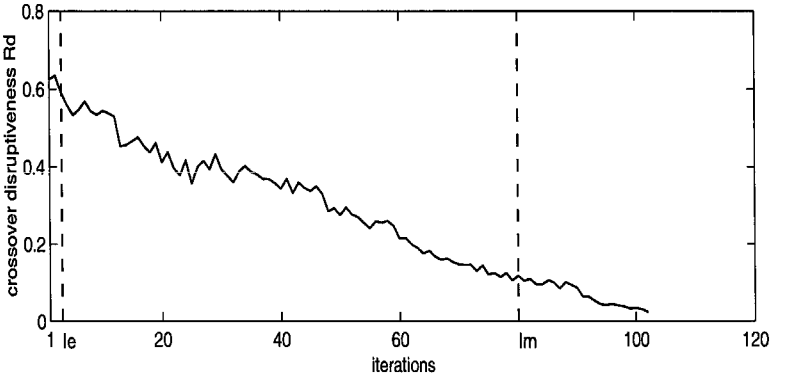
For all the tests below, the target shape is a solid square mesa, consisting of five polygon layers with all the vertical intervals equal to 5 μm. Thus the etching depth D is fixed to 20 μm. With some preliminary runs, the implementation parameters are chosen as the following: the generation size is 80; I_e and I_m are 3 and 80 respectively; k_1 is set to 0.01 and k_2 is 0.02. The chosen generation size is found to be large enough for most runs. The initial elitist selection is used only during the first three iterations so that enough individuals with good performance are collected without suffering too much loss of convergence. The middle stage is set rather long with both small k_1 and k_2 to provide good chance to obtain gentle convergence. These parameters are fixed for all the testing runs in this section so that comparisons can be made on the convergence of different runs. In fact, they are chosen such that the most challenging runs can also obtain fairly good results. Therefore, this parameter setting may not be optimal for all the runs, especially considering the computation cost.

As an initial start, the first test is run to show how the evolved mask-layout is improved during an entire evolution. The results are shown in Figure 7.4. The feature length of the target shape is set as 75 μm. As shown from the convergence curve, the initial dip occurs right after I_e because of the strong convergence from the elitist selection and the effective control by the small selection bias afterwards. In the remaining stages, the evolution converges quite gently, as was hoped. The results presented contain five snapshots of the best evolved mask-layouts and the etched shapes at critical evolution stages. More snapshots of the evolution process are presented in Appendix B. The key is to observe how each etched shape becomes increasingly close to the target shape. Iteration 1 gives the best one out of all the randomly generated mask-layouts. Iteration 3 gives the best preserved mask-layout from the elitist selection, which does contain well-distributed corner boundaries to grow the

Parameter Table

a (μm)	D (μm)	N	I_t	I_e	I_m	S_b	CPU (min)
75	20	80	104	3	80	k_1=0.01; k_2=0.02	96.7

Evolution Convergency Curve



Evolution Results

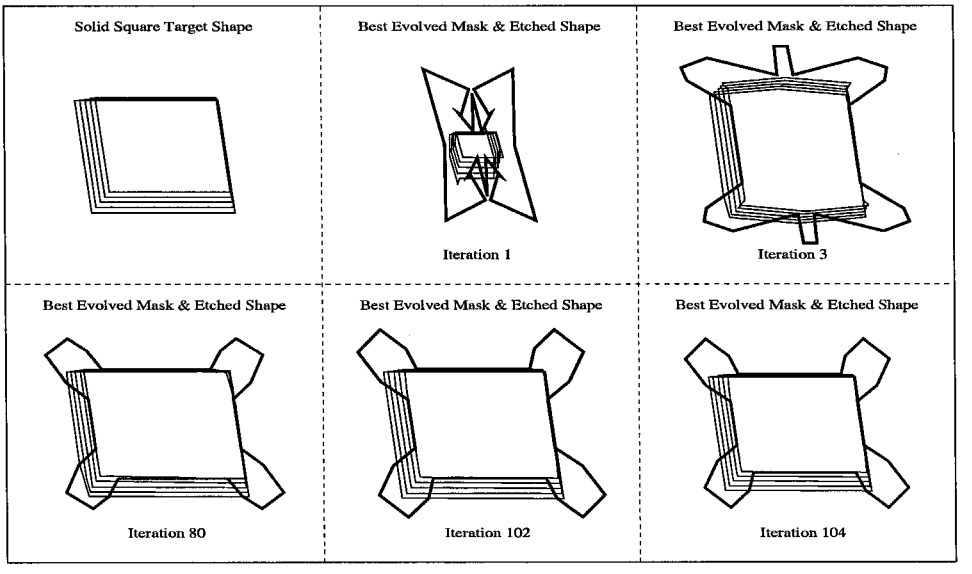


Figure 7.4: Evolution of 24-Side Mask-layout with Compensation Structure for Solid Square (Peg) Target Shape under KOH Etching.

compensation features in future iterations. Iteration 80 illustrates the outcome at the end of the middle stage. The etched shape is fairly close to the target shape, which indicates a success in finding the global region. Iteration 102 gives the final shape evolution result and the performance improvement comes from the final stage of exploitation. At last, the size of the mask-layout found in Iteration 102 is scaled so that the size of its etched shape becomes closer to the target shape. Iteration 104 shows the final best evolved result.

The second test is to evolve mask-layouts with different number of sides so as to obtain various compensation structures for the same target shape. The feature length of the target shape is fixed as 75 μm in all the runs. The side numbers of the evolved mask-layouts are varied from 8 to 24, which are increased by 4 each time due to the quadrisymmetry. The evolved results are shown in Figure 7.5. The evolution parameters and convergence curves for each run are listed in Figure 7.6. The results presented show the best evolved mask-layout and its etched shape for each number of sides. The top left cell shows the target shape, for reference. The focus here is to observe the compensation features and the convex corners formed. As can be seen, all the etched shapes catch the convex corners fairly closely. From the convergence curve of each run, it can be observed that as the number of sides increases, the corresponding evolution starts with a higher crossover disruptiveness and converges more gently. This is because as the number of sides increases, the size of the underlying searching space also increases and thus it is more likely to generate sampling points further away from each other, which leads to the higher initial crossover disruptiveness. In addition, with larger searching space, many more shape features carried through the edge boundary schemata are exposed to the algorithm which requires more searching effort to properly process them. Since the outcomes of all the runs are successful, it indicates that the multiple global optimum exist in different areas of the searching space defined by the different number of sides in the mask-layouts. Note that it is not always better to only search the space with a lower side numbers to save computation time because the resulting global optimum may not be the best in terms of the amount of space used by the evolved compensation features. Among the results presented, the compensation structures evolved from the runs with 8-side or 12-side mask-layouts require more area outside the solid mesa than those from 16-side and 24-side runs. Figure 7.7 illustrates the evolved compensation

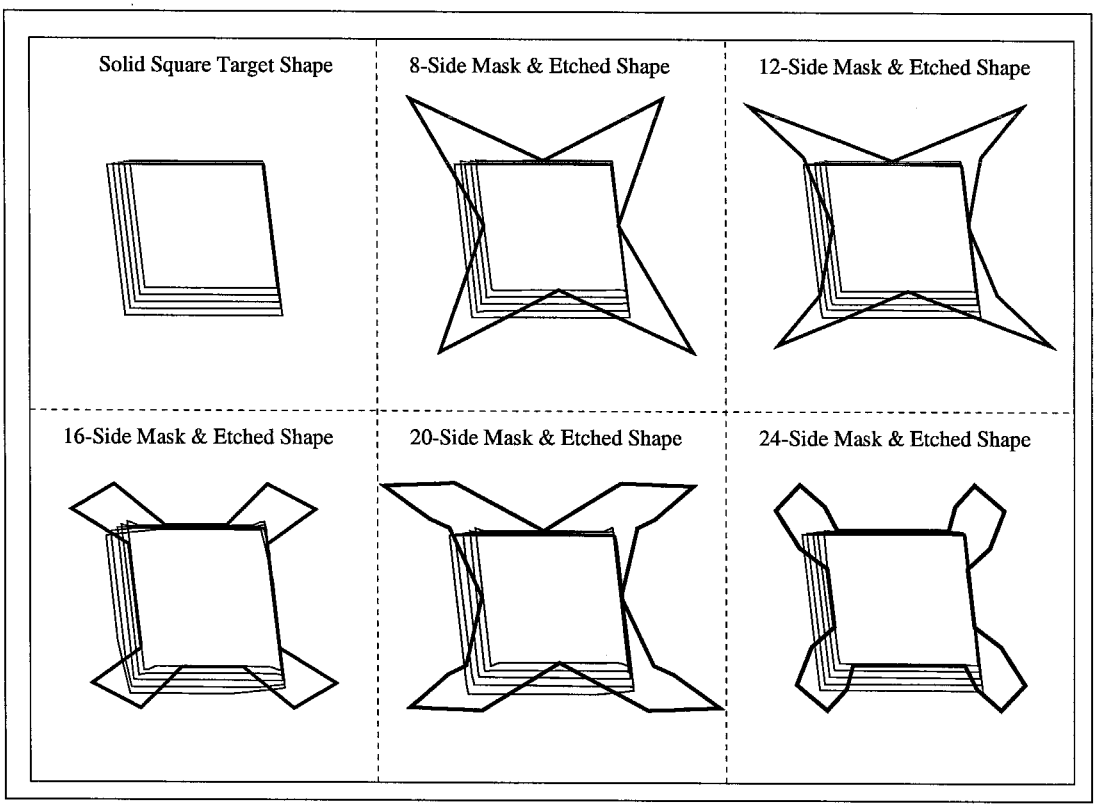


Figure 7.5: Best Evolved Mask-layouts with Various Compensation Structures for Solid Square (Peg) Target Shape under KOH Etching.

Parameter Table

Side Number	a (μm)	D (μm)	N	I_t	I_e	I_m	S_b	CPU (min)
8	75	20	80	49	3	80	k_1=0.01; k_2=0.02	24.0
12	75	20	80	78	3	80	k_1=0.01; k_2=0.02	43.0
16	75	20	80	86	3	80	k_1=0.01; k_2=0.02	63.2
20	75	20	80	98	3	80	k_1=0.01; k_2=0.02	79.0
24	75	20	80	104	3	80	k_1=0.01; k_2=0.02	96.7

Evolution Convergence Curves

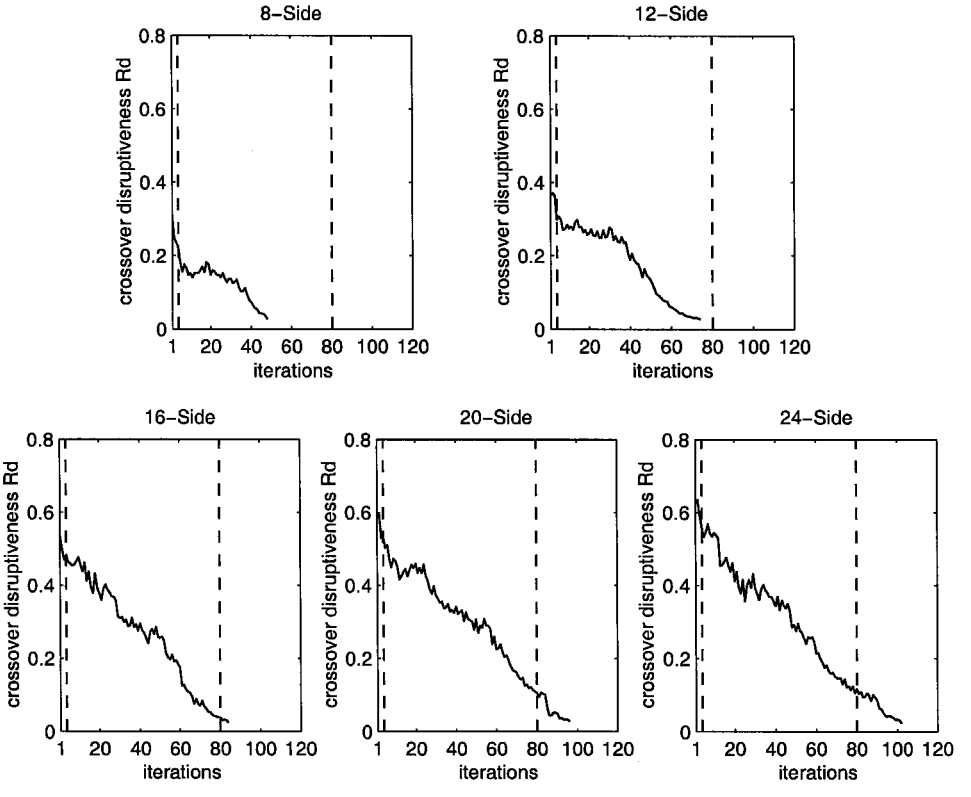
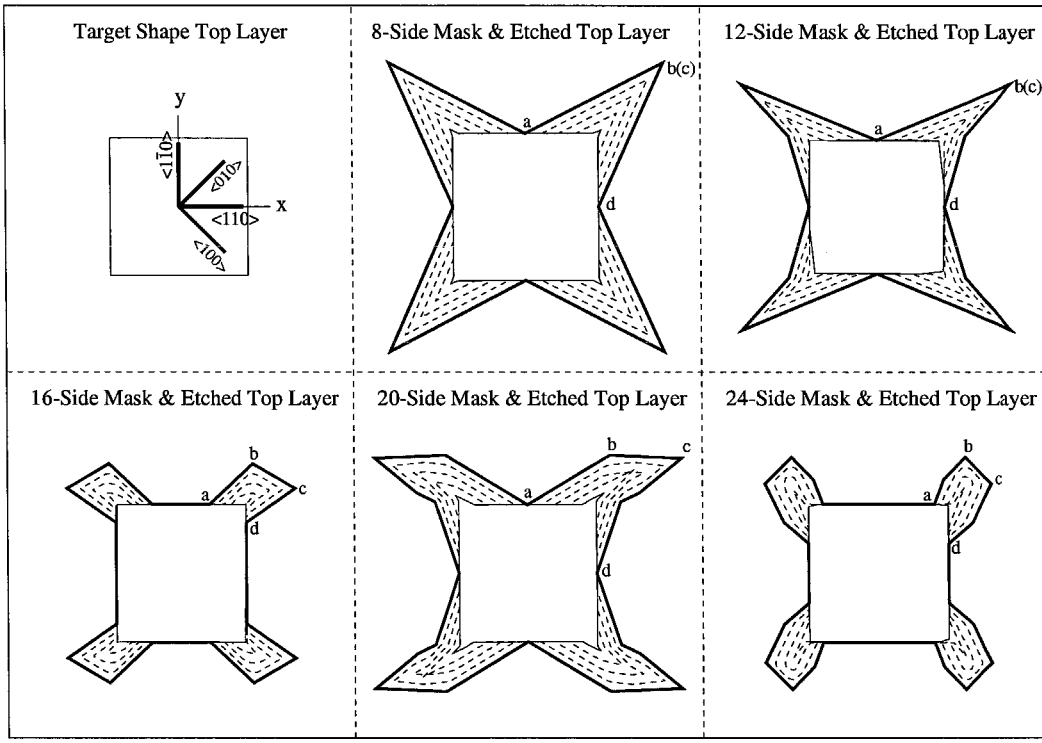


Figure 7.6: Evolutions of Various-Side Mask-layout with Compensation Structure for Solid Square (Peg) Target Shape under KOH Etching.

Etching Profile



Crystal Orientations of Lateral "Edges"

Side Number	$\vec{ab}$		$\vec{dc}$	
	Directional Angle (degree)	Crystal Orientation	Directional Angle (degree)	Crystal Orientation
8	27	<310>	64	<310>
12	23	<310>	61	<310>
16	43	<100>	35	<100> to <310>
20	30	<310>	54	<100> to <310>
24	56	<100> to <310>	54	<100> to <310>

Figure 7.7: Etched Contours at Different Time Steps for Top Layer of Solid Square (Peg) Target Shape under KOH Etching.

features in more detail. The top portion of the figure shows the etching process of the top layer of each etched shape presented in Figure 7.5. In the top left cell, the top layer of the target shape along with the coordinate system and some of the crystal orientations are presented for reference. Each etching process starts with the optimally evolved mask-layout. The dotted lines are the etched profile corresponding to each time step. The duration of each time step is determined by the vertical interval between each two polygon layers of the target shape divided by the etching rate along <100> direction. In this case, the vertical interval is 5 μm and the etching rate is 0.629 μm/min which yields 7.95 minutes for each time step. There are totally 4 time steps. The illustrated etched contours shows how the compensation structures form convex corners through the etching process.

It can be further noticed that even though the compensation features are different as shown in different cells, they all can be roughly decomposed into three portions, namely ab, bc and cd. bc is the free end and ab and cd are the two lateral "edges." All the three portions are marked on each mask-layout. In some cases, the free end is collapsed to one point where b and c are overlapped. Also in some cases, one lateral "edge" corresponds to more than one physical mask edges. The free end is the portion exposed to all the fast etching planes while the lateral "edges" are the key features to slow down the etching process. The goal here is to measure the crystal orientations of the two lateral "edges." To do so, the two "edges" are approximated by the vector $\vec{ab}$ and vector $\vec{dc}$ respectively. As an example, in the case of 12-sides, even though the lateral "edge" cd corresponds to two physical mask edges, it is approximated by the vector $\vec{dc}$. In this case, since the angle between the two physical edges are small, such approximation can be used to reflect the orientation of the overall lateral edges. Fortunately, this is true for all other cases where the approximation is needed. Both of the vectors are defined in the coordinate system shown in the figure. The directional angle of each vector is defined as the angle between the vector and the positive x axis and is calculated from the coordinate values of each end point. Note that this coordinate system is 45 degrees apart from the coordinate system that defines the crystal orientation. Thus, all the calculated directional angles need to have 45 degrees added (or subtracted due to the symmetry) to match the crystal orientation. The directional angles obtained and the approximated known crystal orientations are listed in the bottom table of Figure 7.7. As

an example, in the case of 12-sides, the directional angle of $\vec{dc}$ is calculated to be 61 degrees. Then subtract 45 degrees from 61 degrees to get 16 degrees. The crystal direction of <310> is 18 degrees and therefore can be used to approximate the orientation of $\vec{dc}$. All the approximations are made within 3 degree to a known crystal direction as listed in Table 7.1. In case of exceeding 3 degree range, the closest boundary defined by the two known crystal directions are presented. The results in the table shows that all the crystal orientations of the lateral "edges" in the various evolved compensation structures essentially fall into the range from <310> to <100>. From Table 7.1, both <310> and <100> have relatively small etching rates, under the linear interpolation used by SEGS simulator, any orientations in between also have small etching rates. Therefore, the range defined by <310> and <100> represents a stable etching region which means that according to the etch rate model, all the evolved lateral "edges" are oriented in slow etching directions, which are needed to effectively slow down the undercut etching during corner compensation. The searching power of the evolutionary algorithm can be appreciated in the sense that during each evolution process, the information about the stable etching region can be evolved and then by trying different edges oriented within that region the likelihood of achieving the globally optimal compensation features can be maximized. In this case, such stable etching regions can be regarded as the building blocks which define the promising region that includes the global optimum. Each evolutionary process essentially explores and exploits such a region.

In the third test, seven target shapes with different aspect ratios have been tried. The aspect ratios are gradually increased by decreasing the feature lengths from 75 μm to 30 μm while fixing the etching depth as 20 μm. For each feature length, several runs on different mask-layout side numbers have been tried, and the best evolved outcome is selected. The seven best results are presented in Figure 7.8. In each column, the target shape with different feature length is shown in the top cell and the best evolved mask-layout and its etched shape are shown in the bottom cell. Each etched shape is fairly close to the corresponding target shape. For the target shapes with 75 μm and 45 μm feature lengths, the evolved compensation features with relatively higher side numbers appear to have more efficient area usage. Overall, since there is no area constraint to limit the growth of compensation structures, this test is not expected to be particularly challenging. However, the various evolved

compensation features further justify the benefits of using the evolutionary algorithm. The implementation parameters and the convergence curves are included in Figure 7.9. Observing the convergence curves, again, it appears that the number of sides mainly determines the searching effort involved. Some runs on small side numbers converge fairly quickly, such as the one with a 65 μm feature length which corresponds to an 8-side mask-layout. Some runs on large side numbers involve more searching effort and converge slowly, such as the one with a 75 μm feature length which corresponds to a 24-side mask-layout. This further indicates that increasing the aspect ratios does not bring much extra burden to the algorithm.

7.3.3 Void Cross Shape

For all the tests below, the target shape is a void cross, and consists of four polygon layers with all the vertical intervals equal to 10 μm. Thus the etching depth D is fixed as 30 μm. Again, with some preliminary runs, the implementation parameters are chosen as the following: the generation size is 150; I_e and I_m are 3 and 100 respectively; k_1 is set as 0.01 and k_2 is 0.03. Compared to those in the previous section, the generation size is larger because the runs on mask-layouts with larger number of sides are involved in this section, which requires more sampling points in a bigger search space. Because of the larger generation size, more iterations are expected and thus the middle stage is increased. The value of k_2 is also slightly increased to bring final stage convergence for some of the longer runs. Again, these parameters are used for all the testing runs in this section.

The first test is to show feature improvement of the evolved mask-layout as an evolution process is converged. The results are shown in Figure 7.10. The feature length of the target shape is set as 80 μm. Observing the convergence curve, the middle stage setting appears a little too wide for this run. The process starts with a fairly large amount of explorative searching effort up to around Iteration 60 and then converges quickly. Before reaching the end of the middle stage, the process has been nearly converged. The results presented contain five snapshots of the best evolved mask-layouts and the etched shapes at critical evolution stages. More snapshots of the evolution process are presented in Appendix C. Iteration 3 gives the best preserved mask-layout from the elitist selection, which contains

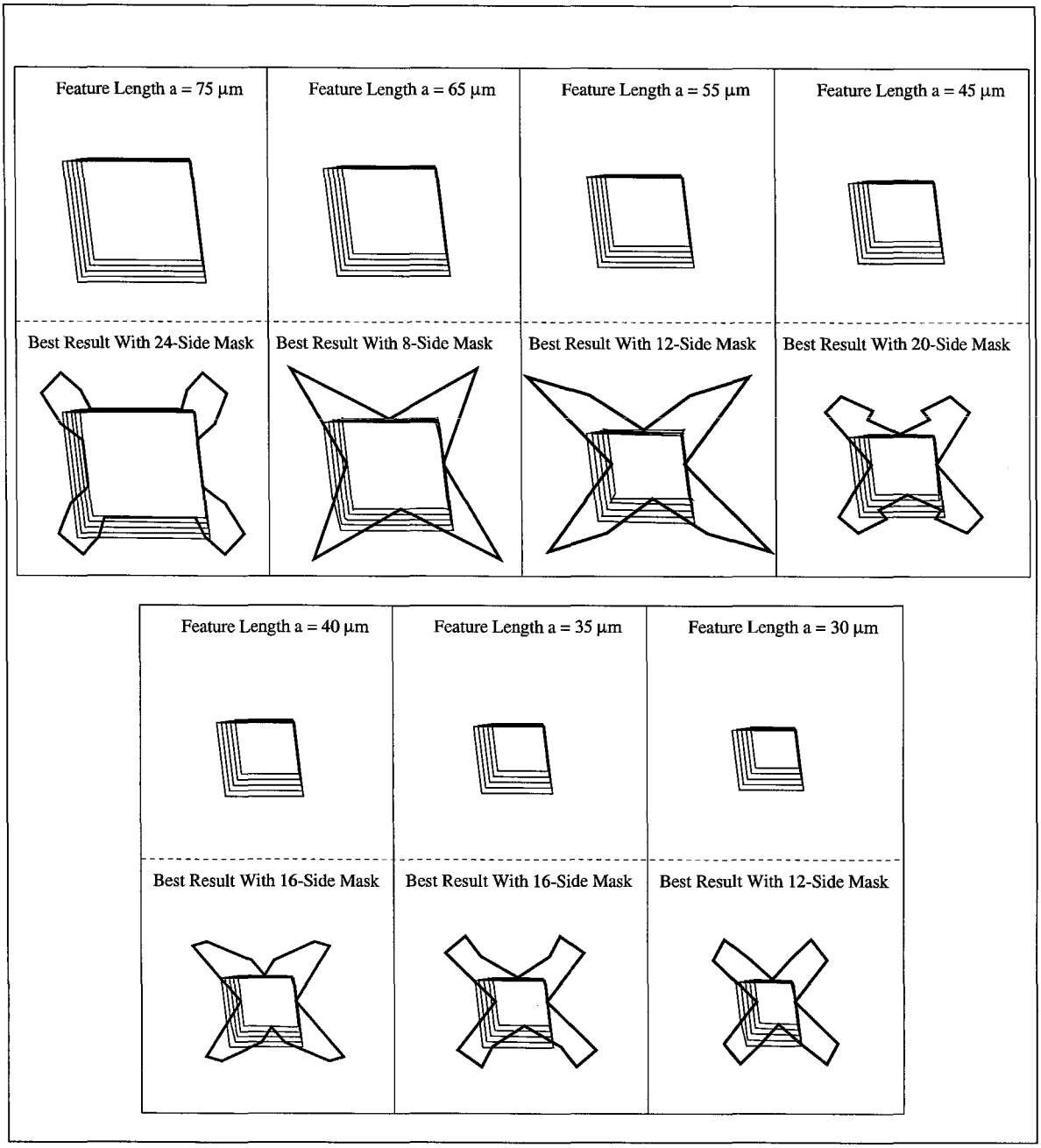


Figure 7.8: Best Evolved Mask-layouts for Solid Square Target Shapes with Decreasing Feature Length and Fixed Etching Depth (20μm) under KOH Etching.

Parameter Table

a (μm)	D (μm)	N	I_t	I_e	I_m	S_b	CPU (min)
75	20	80	104	3	80	k_1=0.01; k_2=0.02	96.7
65	20	80	62	3	80	k_1=0.01; k_2=0.02	37.2
55	20	80	86	3	80	k_1=0.01; k_2=0.02	80.1
45	20	80	113	3	80	k_1=0.01; k_2=0.02	97.6
40	20	80	89	3	80	k_1=0.01; k_2=0.02	88.3
35	20	80	94	3	80	k_1=0.01; k_2=0.02	92.4
30	20	80	92	3	80	k_1=0.01; k_2=0.02	89.1

Evolution Convergence Curves

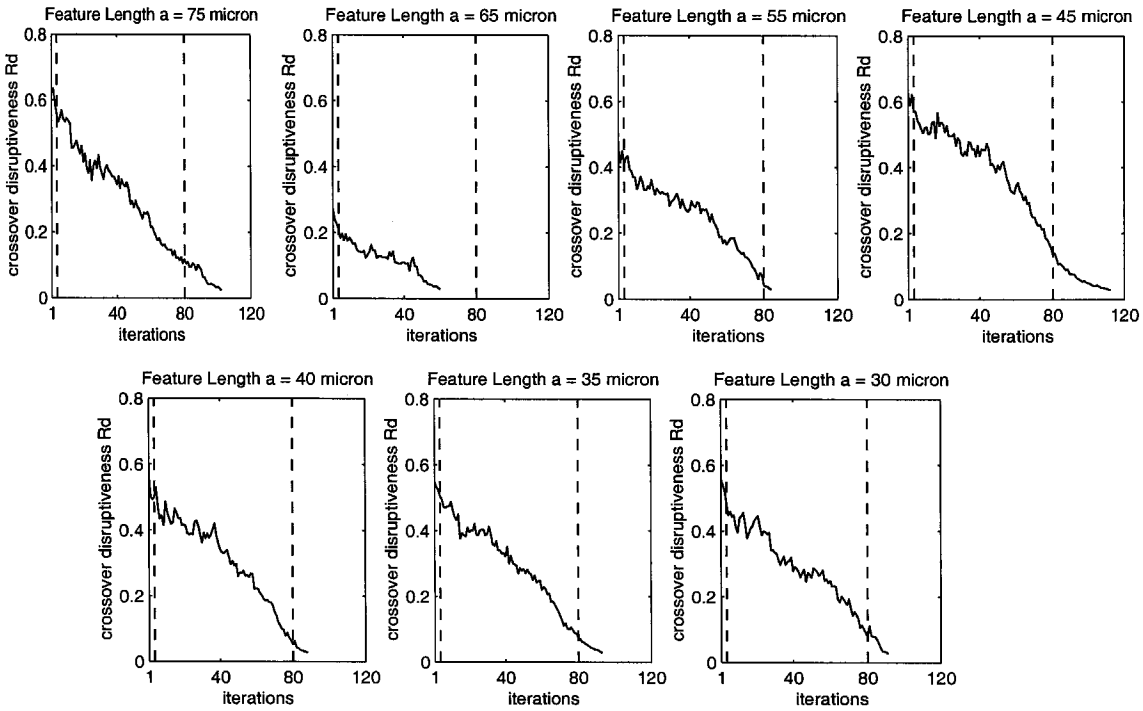


Figure 7.9: Evolutions of Mask-layouts For Solid Square Target Shapes With Decreasing Feature Length and Fixed Etching Depth under KOH Etching.

good corner features to grow the compensation structures later. Iteration 100 illustrates the outcome at the end of the middle stage. The etched shape is quite close to the target shape. Iteration 111 gives the evolved optimal shape. It can be seen that the inner convex corners of the etched shape become sharper as compared to the one at iteration 100 which resulted from the exploitation during the final stage. Finally, through further three iterations of size scaling, iteration 114 shows the final result. The success in the outcome shows that the convergence behavior examined earlier actually means the algorithm picks the "right" information fairly efficiently or in other words, this is an easy run for the algorithm.

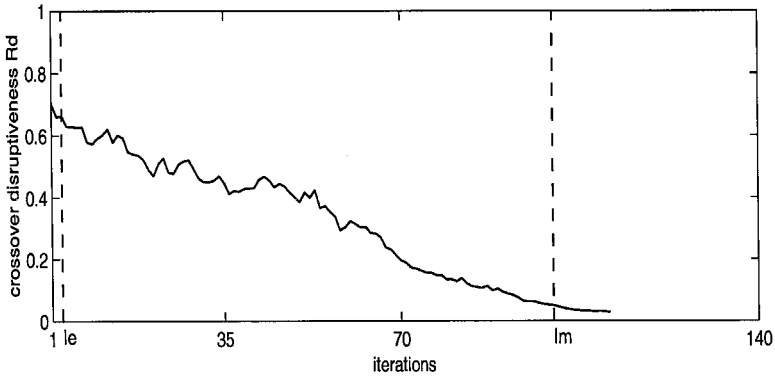
In the second test, the number of sides in the evolved mask-layouts is varied from 16 to 32 to obtain various compensation structures. The feature length of the target shape is fixed at 80 μm for all the runs. The side numbers are increased by 4 each time due to the quadrisymmetry. The test results are presented in Figure 7.11. The evolution parameters and convergence curves for each run are listed in Figure 7.12. The results presented show the best evolved optimal mask-layouts with their etched shape for different side numbers. The top left cell shows the target shape for reference. Again, the focus here is to observe the compensation structures to form the four sharp inner convex corners of each etched shape. By comparing the convergence curves of different runs, the same observation can be made as in the case of solid mesa. With the number of sides being increased, the corresponding evolution starts with a higher crossover disruptiveness and converges more gently. This means that larger searching effort is involved to evolve the compensation structures with higher side numbers. A further examination of the evolved compensation features can be seen from Figure 7.13. As illustrated in the case of the solid mesa, the same procedures are used to obtain the crystal orientation of the lateral "edges" in each evolved compensation structure. In this case, the lateral "edges" are approximated by vectors $\vec{ba}$ and $\vec{cd}$, as shown in the figure. The results are listed in the bottom table in the figure. Again, in this case, all the evolved lateral "edges" are oriented within the stable etching region between <310> and <100>, which further demonstrates the effectiveness of the evolutionary algorithm to process complex geometric information in two dimensional space.

The third test is to evolve compensation structures for different target shapes with increasing aspect ratios. The feature length of each target shape is gradually decreased from

Parameter Table

a (μm)	D (μm)	N	I_t	I_e	I_m	S_b	CPU (min)
80	30	150	114	3	100	k_1=0.01; k_2=0.03	110.5

Evolution Convergence Curve



Evolution Results

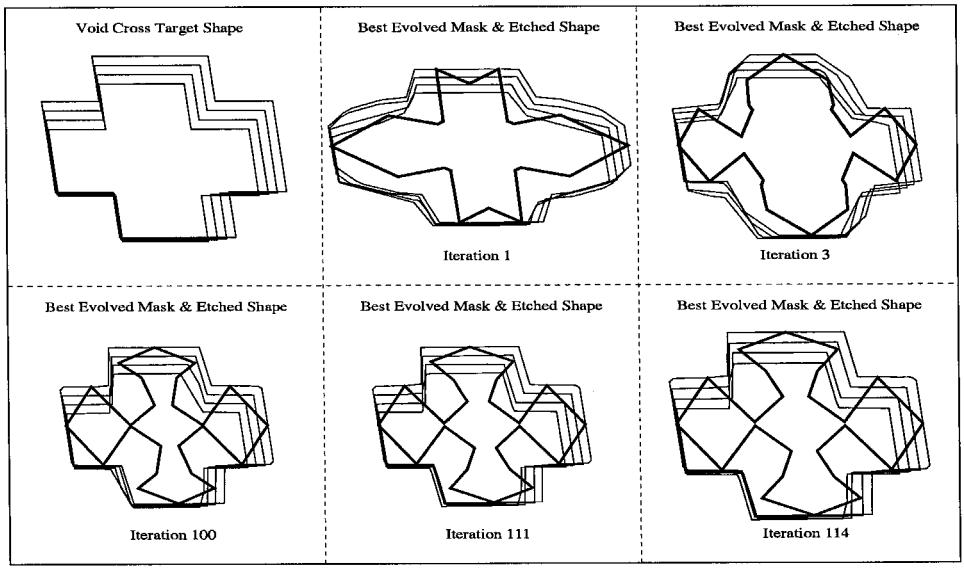


Figure 7.10: Evolution of 24-Side Mask-layout with Compensation Structure for Void Cross Target Shape under KOH Etching.

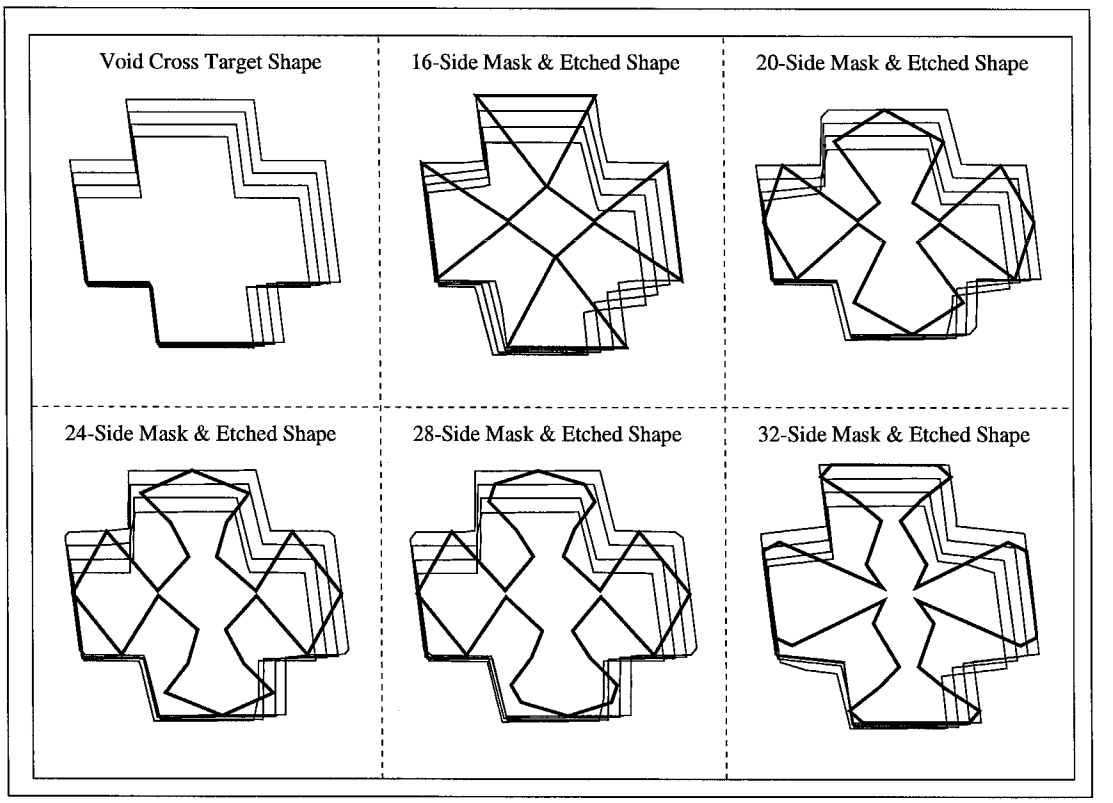


Figure 7.11: Best Evolved Mask-layouts with Various Compensation Structures for Void Cross Target Shape under KOH Etching.

Parameter Table

Side Number	a (μm)	D (μm)	N	I_t	I_e	I_m	S_b	CPU (min)
16	80	30	150	94	3	100	k_1=0.01; k_2=0.03	81.1
20	80	30	150	103	3	100	k_1=0.01; k_2=0.03	94.3
24	80	30	150	114	3	100	k_1=0.01; k_2=0.03	110.5
28	80	30	150	121	3	100	k_1=0.01; k_2=0.03	120.2
32	80	30	150	124	3	100	k_1=0.01; k_2=0.03	131.4

Evolution Convergence Curves

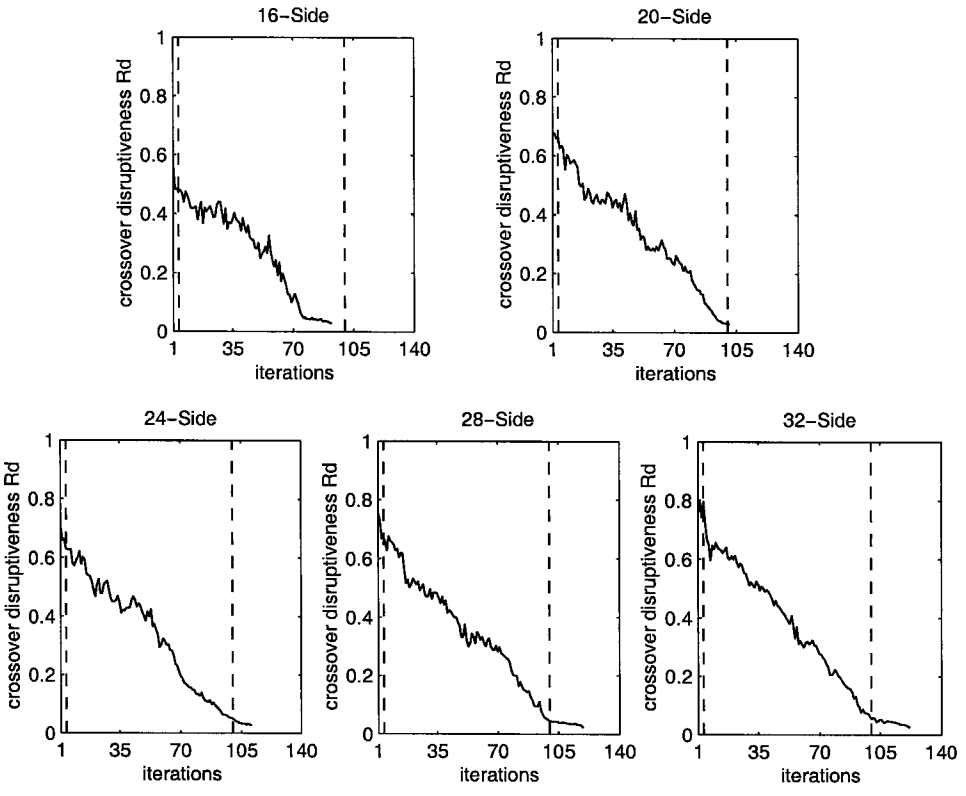
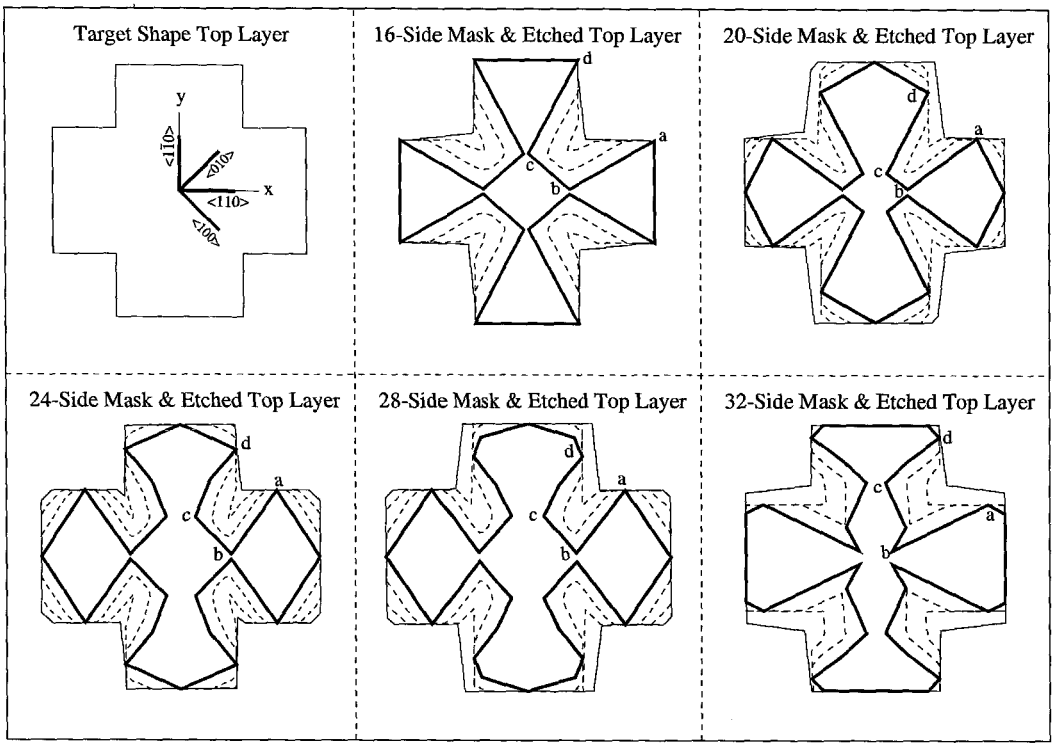


Figure 7.12: Evolutions of Various-Side Mask-layout with Compensation Structure for Void Cross Target Shape under KOH Etching.

Etching Profile



Crystal Orientations of Lateral "Edges"

Side Number	$\vec{ba}$		$\vec{cd}$	
	Directional Angle (degree)	Crystal Orientation	Directional Angle (degree)	Crystal Orientation
16	29	<310>	62	<310>
20	36	<100> to <310>	63	<310>
24	55	<100> to <310>	60	<310>
28	52	<100> to <310>	56	<100> to <310>
32	26	<310>	40	<100>

Figure 7.13: Etched Contours in Different Time Steps for Top Layer of Void Cross Target Shape under KOH Etching.

80 μm to 40 μm while the etching depth is fixed at 30μm. In this case, the particular challenge comes from the constraint area defined within the void cross boundary. Such constraint is expected to be limited to growing compensation features which are particularly needed as the aspect ratio increases. The results are presented in Figure 7.14. Overall, the etched shapes from the evolved mask-layouts match the corresponding target shapes, even though some of them experience overetching, especially those with high aspect ratios. The searching power of the algorithm can also be appreciated from the adjustments of the compensation features in response to the increase of the aspect ratios. The evolved features tend to grow towards the center point to give the best effort to form sharper convex corners. Furthermore, almost all of the best evolved mask-layouts have high side numbers, which indicates that with more sides, the algorithm is able to evolve more shorter edges so that more features can be formed in the constrained area. All the implementation parameters and the convergence curves are shown in Figure 7.15. In this case, the convergence curves indicate that as the aspect ratio increases, more total iterations are involved with more gentle convergence. This means that the increasing of aspect ratios challenges the algorithm to apply more searching effort.

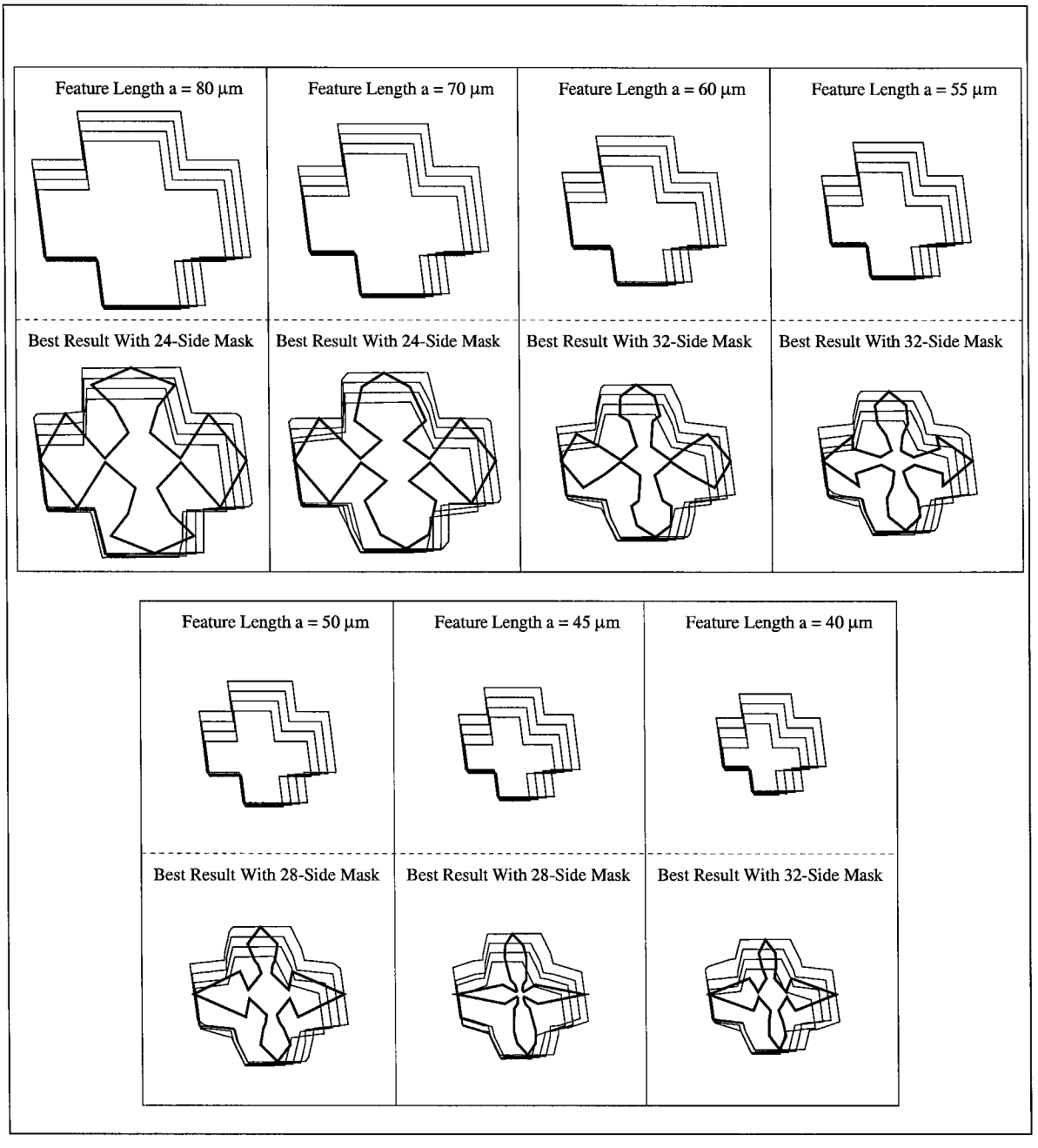


Figure 7.14: Best Evolved Mask-layouts for Void Cross Target Shapes with Decreasing Feature Length and Fixed Etching Depth (30μm) under KOH Etching.

Parameter Table

a (μm)	D (μm)	N	I_t	I_e	I_m	S_b	CPU (min)
80	30	150	114	3	100	k_1=0.01; k_2=0.03	110.5
70	30	150	116	3	100	k_1=0.01; k_2=0.03	111.3
60	30	150	136	3	100	k_1=0.01; k_2=0.03	142.4
55	30	150	140	3	100	k_1=0.01; k_2=0.03	147.1
50	30	150	148	3	100	k_1=0.01; k_2=0.03	152.3
45	30	150	154	3	100	k_1=0.01; k_2=0.03	176.6
40	30	150	159	3	100	k_1=0.01; k_2=0.03	184.3

Evolution Convergence Curves

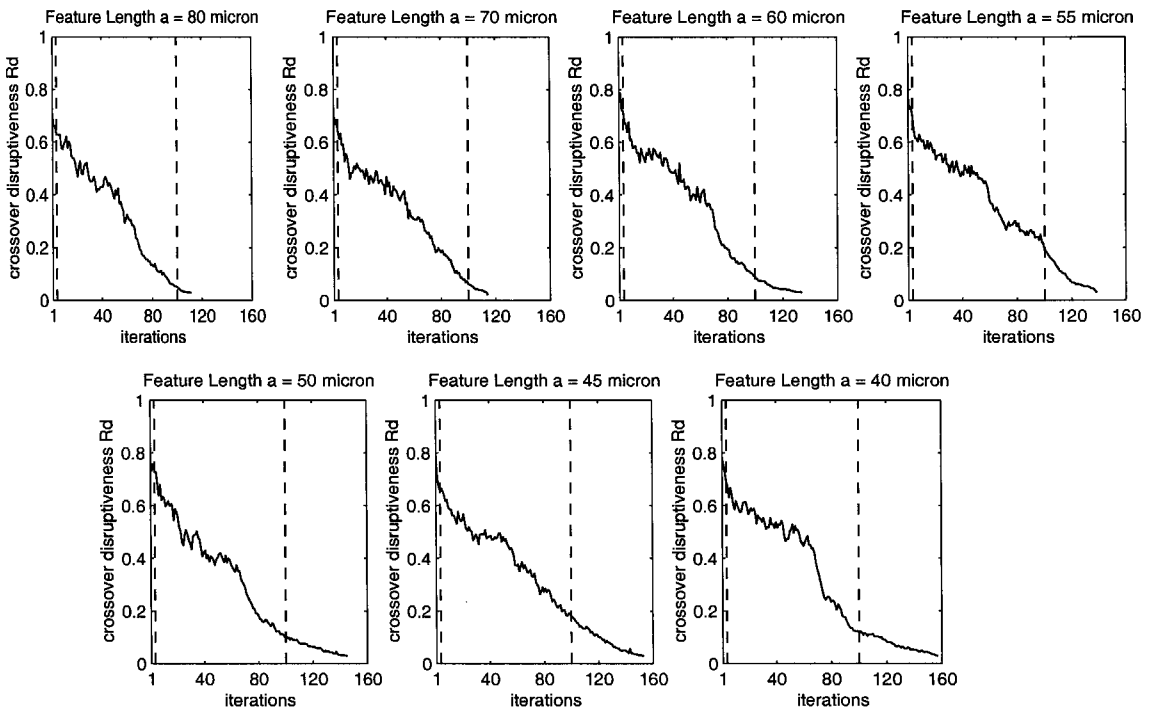


Figure 7.15: Evolutions of Mask-layouts For Void Cross Target Shapes With Decreasing Feature Length and Fixed Etching Depth under KOH Etching.

Chapter 8

Conclusion

8.1 Summary

In this thesis, evolutionary techniques have been developed for mask-layout synthesis. An object-oriented architecture has been implemented to ensure a high level of modularity during development and testing. All mask-layouts are represented as 2D polygons. Since curves can be approximated with a (large) number of polygon sides, utilizing a polygonal representation should not introduce any practical limitations. Genetic operators were constructed, including coding schemes, initializations, crossovers and mutations. With the use of edge boundary schema, the "schemata theory" is used to explain the dynamics of a mask-layout evolution. Two types of deceptions have been examined: "hill deception" and "cliff deception." Finally, the selection scheme and genetic operation have been developed and provide three control methods: selection bias, distance bias ratio and bride pool size. These provide a convenient and effective way to control evolution convergence. Evolution strategies based on the use of these three control methods to overcome the deceptions encountered illustrate the rate of convergence can be controlled.

An evolutionary algorithm has been constructed based on the evolution techniques developed to synthesize mask-layouts for a simulated wet etching process. The goal is that the etched 3D shapes closely match the specified target shape. Tests of this algorithm have been focused on the evolution of mask-layouts with various compensation structures. Even though analytic methods can be used to design certain compensation structures, they may not work in all situations where compensation features are desirable. Such situations may

arise from the need to obtain the most material-efficient compensation structure within a restricted area, or the absence of a known compensation solution for certain target shape. The need to identify optimal compensation structures provides a particular advantage to use an evolutionary algorithm, because it automatically provides an explorative search. The effectiveness of this approach has been demonstrated by the test results including additional examples in Appendices B, C, D, which further give confidence that these evolutionary techniques can be used to evolve complex features of mask-layouts for a wide range of applications.

8.2 Future Work

The evolutionary techniques developed here contain several limitations which need future improvement. First, each mask-layout evolution is currently restricted to a fixed number of sides, which adds the burden for users to guess the appropriate number of sides for the evolved mask-layout in advance. Ultimately, the evolutionary algorithm should be able to automatically evolve the global optimal mask-layout without the user knowing the number of sides in advance. This capability will require the development of new crossover operators, as well as a genetic operation strategy, so that within each generation, mask-layouts with various side numbers can be properly evolved.

Second, the evolution convergence can be affected by the size of each generation. Even though a large generation size always leads to better convergence with an increasing chance of hitting the global optimum, it may also significantly reduce the evolution efficiency, particularly when an expensive simulator is utilized to evaluate the performance of each individual. Future effort is needed to develop a proper strategy on the choice of generation size and incorporate it into the existing evolutionary strategies so that the evolution convergence can be controlled in a more complete and effective way, including the computational costs of the performance evaluation. An example of such a strategy can be varying the generation size as each evolutionary process is carried on. Initially, a large generation size is suggested to ensure sufficient amount of information flow being processed. As the evolution converges, especially during the middle towards late stages, most individuals are expected to

be quite similar and thus, the generation size can be reduced to minimize redundancy. Of course, further effort is needed to determine the proper generation sizes at each stage of the evolutionary process, and the effectiveness such an approach needs to be tested.

Third, each evolution process currently starts evolving an optimal shape based on mask-layouts scaled to a common size. This is then followed by a greedy search for the optimal size. This approach, however, may not always be appropriate. As an example, consider the mask-layout synthesis for the the wet etching process implemented in Chapter 7. Most iterations are devoted to evolving the shapes of mask-layouts under the same specified size and then at the end, the size of the mask-layout with the evolved optimal shape is either enlarged or reduced so that the overall size of the etched shape also matches the target shape. It is hoped that for each etched shape formed by a size variation, the shapes of all its polygon layers stay the same. However, in most cases, this will not be true since the size variation of a mask-layout only changes the lateral size of its etched shape not the etching depth. With fixed etching depth, the etching time is also fixed and therefore, all the etching vectors which represent the direction and magnitude of the lateral motion for each mask edge remain unchanged. Under these fixed etching vectors, the change of the mask size will directly cause a change of the relative motion among the edges which further changes the shapes of the polygon layers. The current approach to deal with it is to make a good initial guess on the final size of the mask-layout and set it as the common scaled size during the shape evolution. In this way, during the final size searching stage, only small size variations are involved and this leads to only small changes in the shapes of etched polygon layers. Clearly, this approach directly relies on the initial guess of the mask size which is not an effective way to solve the problem.

As an illustration, Figure 8.1 shows two cases where the common scaled sizes are initially set improperly. In each case, two target shapes are presented; one is a solid square (peg) and the other is a void cross (hole). The feature size and etching depth of each target shape are indicated in the figure. Each evolution yields the optimal shape of mask-layouts whose etched shape closely matches the target shape, as shown in the middle columns. However, due to the final stage variation in size, the corresponding adjustments of etched shapes experience either overetching or underetching. The top portion of the figure shows

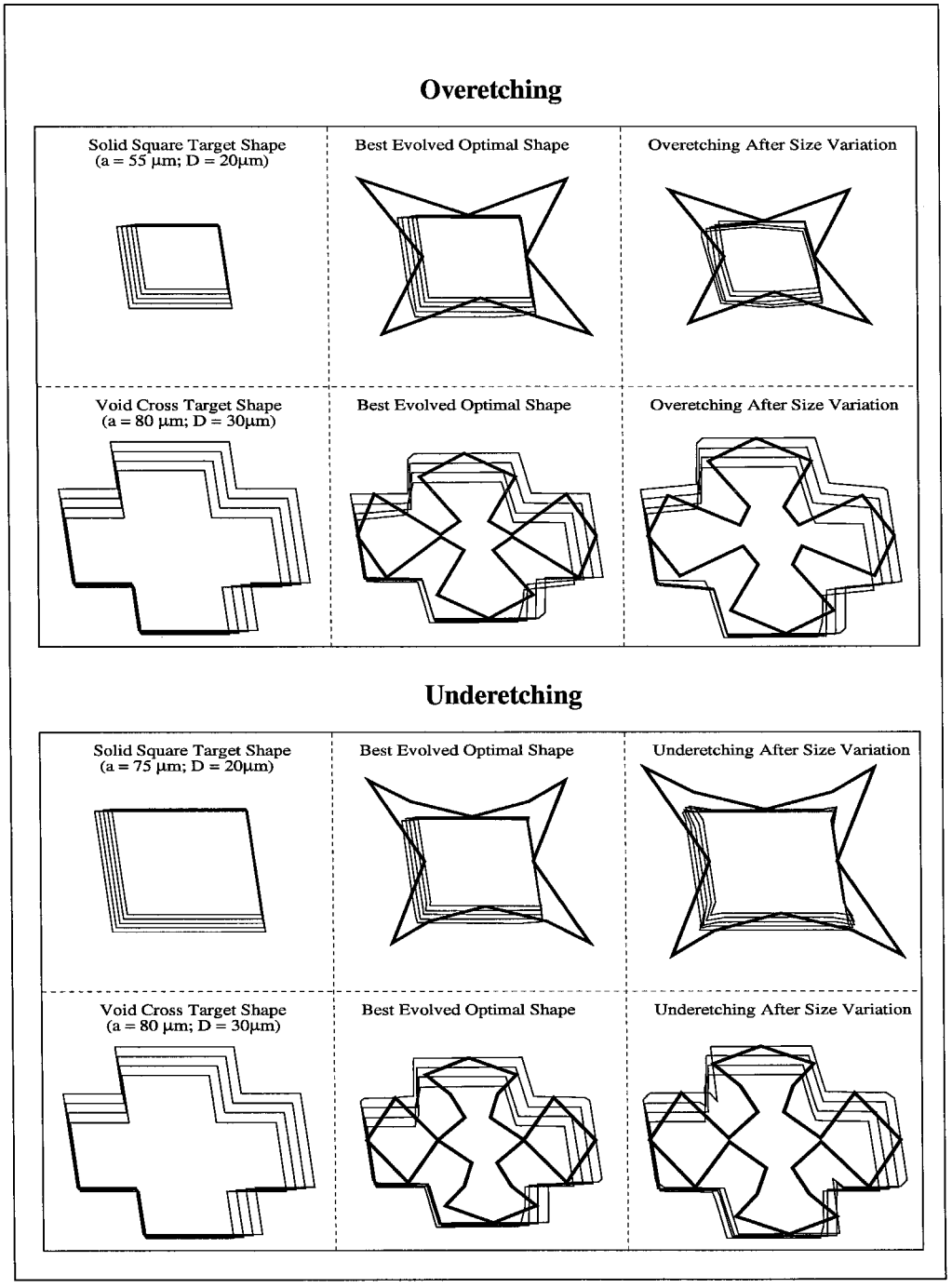


Figure 8.1: Illustration on Ill Effects from Size Variations Due to Initial Improper Guess on Commonly Scaled Size of Mask-layouts.

the case of overetching where all the convex corners of each etched shape are formed and then overetched due to the excess etching. The bottom portion of the figure shows the case of underetching where the etching process stops before all the convex corners are formed. Both cases show bad outcomes because the initially guessed sizes were either too large or too small. Therefore, future effort is needed to eliminate this problem. One possible way may be to evolve the shape and the size separately but simultaneously. Usually, the size quickly converges. However, such convergence may restrict the further evolution of the edge distance ratio of the mask-layouts. So the challenge comes from how to balance the two evolution efforts.

Finally and most importantly, future efforts are needed to extend the current application. Mask-layout synthesis on wet etching processes with two or more etching steps appears likely to be an important and power application of such evolutionary techniques because the connection between mask-layouts and their etch shapes can be rather difficult to conceive and thus the exploration of good mask candidates becomes more useful. In addition, future mask-layout synthesis may not be limited to the shape matching between the etched shape and target shape. The performance evaluations may be directly provided in terms of desired design functions. In other words, such an evolutionary algorithm based approach can ultimately be used to achieve both the process synthesis and shape synthesis mentioned in Chapter 1.

Appendix A

Conversion Between Binary Coding And Gray Coding

The conversion formula from binary coding to Gray coding is:

$$g_k = \begin{cases} b_1 & k = 1 \\ b_{k+1} \bigoplus b_k & k > 1 \end{cases}$$

where

$$\begin{aligned} g_k &= kth\ Gray\ code\ bit \\ b_k &= kth\ binary\ code\ bit \\ \bigoplus &= addition\ mod\ 2. \end{aligned}$$

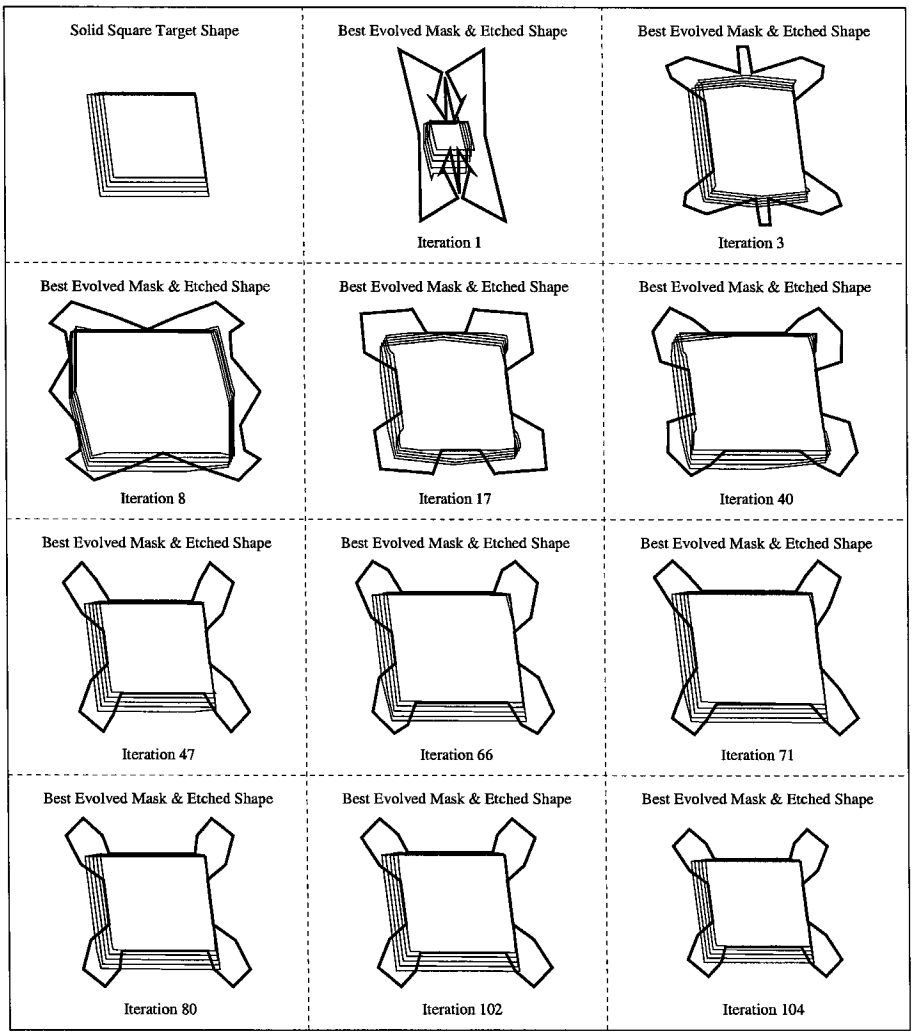
The conversion from Gray coding to binary coding is:

$$b_k = \sum_{1 \le i \le k} g_i.$$

The above summation is done modulo 2. As an example, the binary code 1101011 corresponds to the Gray code of 1011110.

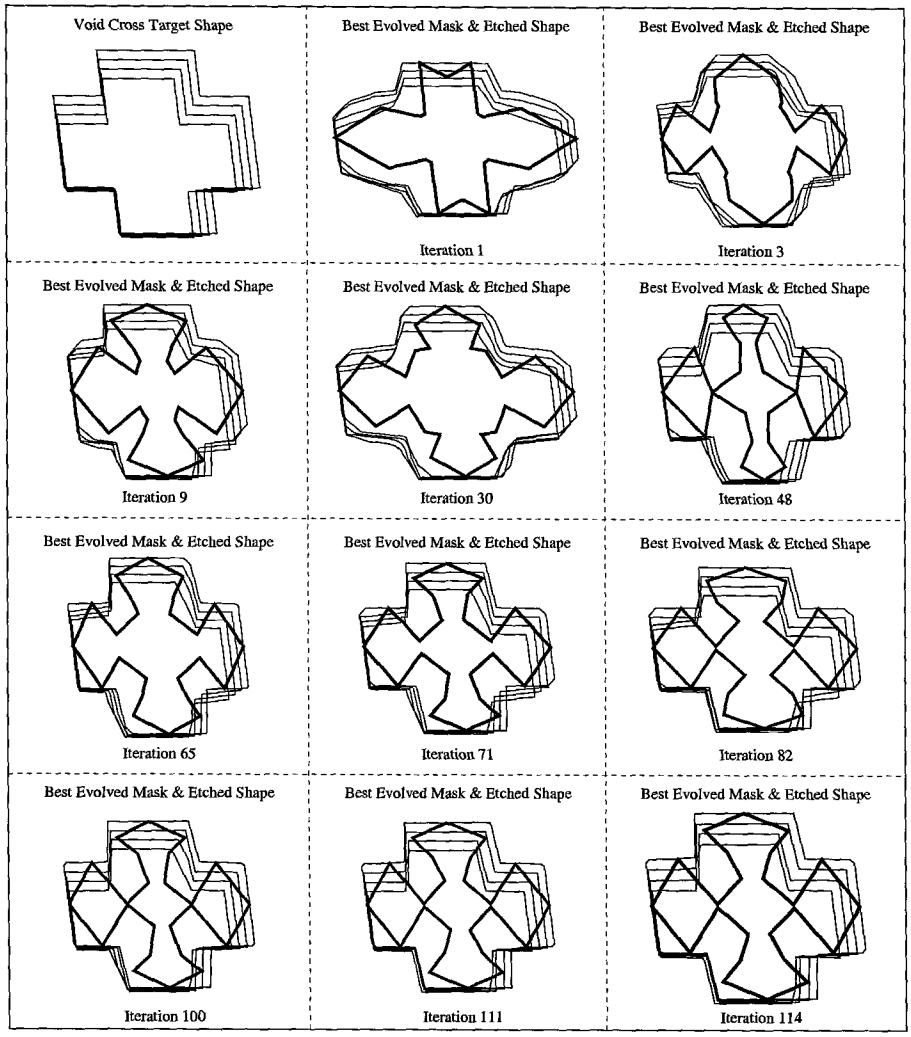
Appendix B

An Evolutionary Process of 24-Side Mask-layout for Solid Square Peg Target Shape under KOH Etching



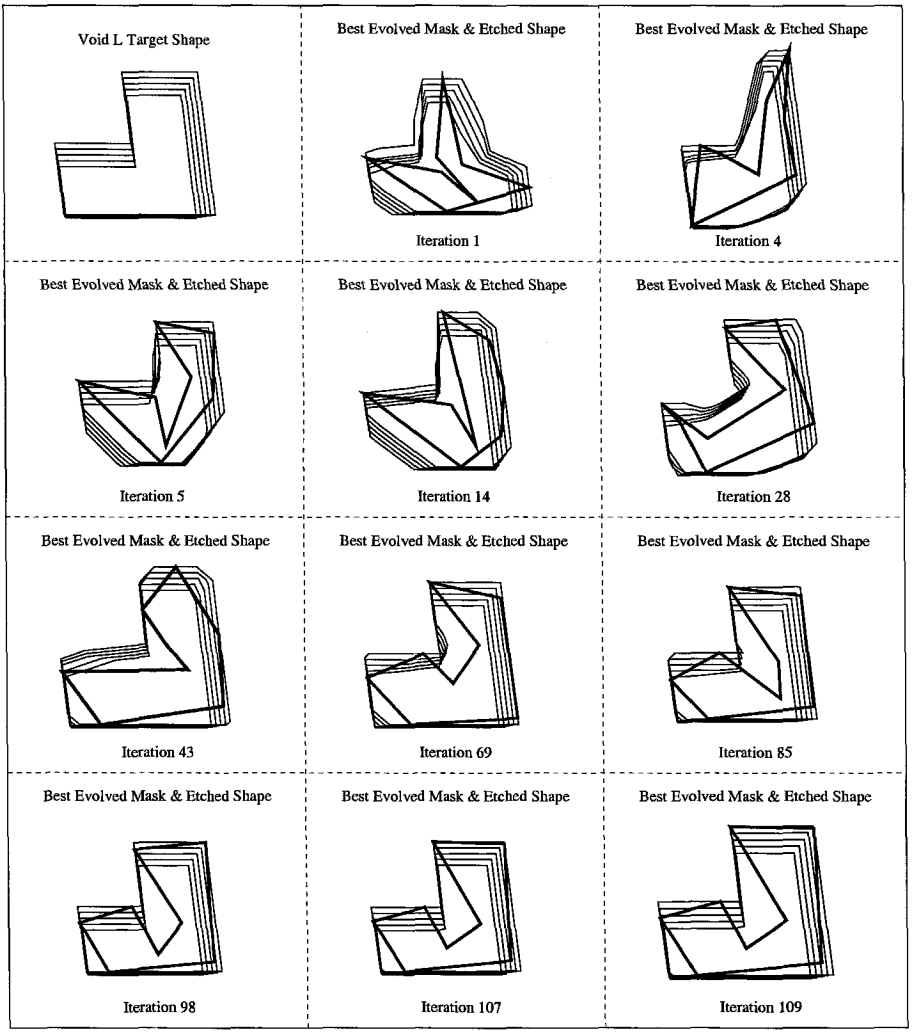
Appendix C

An Evolutionary Process of 24-Side Mask-layout for Void Cross Target Shape under KOH Etching



Appendix D

An Evolutionary Process of 8-Side Mask-layout for Void Asymmetric Target Shape under KOH Etching



Bibliography

[1] P. Adby and M. Dempster. *Introduction to Optimization Methods*. Chapman and Hall, London, 1974. 6.1

[2] Erik K. Antonsson. Structured design methods for mems final report. *NSF MEMS Workshop*, 1995. Nov.12-15,1995. 1.2

[3] Thomas Auer and Marin Held. Rpg - heuristics for the generation of random polygons. *Proc. of the 8th Canada Conf. Computational Geometry*, pages 38–43, 1996. 6.3.3, 6.3.3

[4] Thomas Auer and Martin Held. Rpg package for the generation of random polygons. *http://www.cosy.sbg.ac.at/ held/projects/rpg/rpg.html*, 1995. 6.3.3

[5] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. pages 57–62, 1994. 3.8.3

[6] T. Bäck and F. Hoffmeister. Extended selection mechanism in genetic algorithms. pages 92–99, 1991. 3.8.3

[7] James E. Baker. Adaptive selection methods for genetic algorithm. pages 101–111, 1985. 3.8.2

[8] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *The Second International Conference on Genetic Algorithms*, 1987. 3.8.2, 3.8.2, 6.4.3

[9] L. Booker. *Improving Search In Genetic Algorithm*. New York: Pitman, 1987. 3.4.2

[10] Ernesto Bribiesca. Measuring 3-d shape similarity using progressive transformations. volume vol. 29, No. 7, 1996. 5.1

[11] R. A. Buser and N. F. de Rooij. ASEP: A CAD program for silicon anisotropic etching. *Sensors and Actuators*, 28:71–78, 1991. 1.1

[12] R. Cheng and M. Gen. Genetic algorithms for multi-row machine layout problem. *Edngineering Design and Automation*, 1996. 3.4.3

[13] J. Cohoon and W. Paris. Geneti placement. *IEEE Transactions on Computer-Aided Design*, 6:1272–1277, 1987. 3.2

[14] S. B. Crary and Y. Zhang. CAEMEMS: An intergrated computer-aided engineering workbench for micro-electro mechanical systems. In *MEMS 90*, pages 113–115, Institute of Electrical Engineers, 1990. 1.1

[15] C.Zhu, G.Sundaram, J.Snoeyink, and Joseph S.B.Mitchell. Generating random polygons with given vertices. *Computation Geometry Theory Application*, 6:277–290, 1996. 6.3.3, 6.3.3

[16] D. Dasgupta and Z. Michalewicz. *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, 1997. 3.4.3

[17] Yuval Davidor. Epistasis variance: A viewpoint on ga-hardness. *Foundation of Genetic Algorithms*, pages 23–35, July 1991. 3.7

[18] L. Davis. Applying adaptive algorithms to epistatic domains. 1985. 3.4.4

[19] L. Davis. Adapting operator probabilities in genetic algorithms. *Proceedings of the third International Conference on Genetic Algorithms*, 1989. 3.9

[20] L. Davis. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991. 3.4.3

[21] Kenneth A. DeJong. The analysis and behavior of a class of genetic adaptive systems. *Ph.D. dissertation*, 1975. 3.4.2

[22] E.M.Arkin, L.P.Chew, D.P.Huttenlocher, K.Kedem, and J.S.B.Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis And Machine Intelligence*, vol. 13, No. 3:209–215, 1991. 5.2, 5.2, 7.2

[23] L. J. Eshelman, R. Caruna, and J.D. Schaffer. Biases in the crossover landscape. *The Third International Conference on Genetic Algorithms*, pages 10–19, 1989. 3.4.2

[24] Larry J. Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombinition. *Foundation of Genetic Algorithms*, pages 265–283, July 1991. 3.3, 3.4.2, 3.8.3, 3.8.3, 3.9

[25] Larry J. Eshelman and J.David Schaffer. Real-coded genetic algorithms and interval-schemata. *Foundation of Genetic Algorithms*, pages 187–202, July 1991. 3.4.3, 3.4.3

[26] B. Fox and M. McMahon. Genetic operators for sequencing problems. *Foundations of Genetic Algorithms*, pages 284–009, July 1991. 3.2, 3.2, 3.4.4

[27] M. Gen, B. Liu, and K. Ida. Evolution program for constrained nonlinear optimization. *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, 1994. 3.5

[28] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms And Engineering Design*. John Wiley & Sons, 1996. 3.4.3, 3.4.4, 3.5, 3.8.1, 3.8.2

[29] B. Gogoi, R. Yeun, and C. H. Mastrangelo. The automatic synthesis of planar fabrication process flows for surface micromachined devices. In *Proceedings of the IEEE Micro Electro Mechanical Systems Workshop*, pages 153–157, Oiso, Japan, January 1994. 1.1, 2.2

[30] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. 3.1, 3.2, 3.2, 3.4.4, 3.8.2

[31] David E. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. *IlliGAL Report 90001, University of Illinios at Urbana-Champaign, Urbana, IL*, 1990. 3.2

[32] John . Grefenstette. Deception considered harmful. *Foundations of Genetic Algorithm*, pages 75–91, July 1993. 3.7

[33] John J. Grefenstette. Incorporating problem specific knowledge in genetic algorithms. In *Genetic Algorithms and Simulated Annealing*, 1987. 3.3, 3.4.4

[34] H.Alt and M.Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. 5.2, 5.2

[35] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975. 3.1, 3.2, 3.4.2, 3.6, 3.6, 6.1

[36] R. B. Hollstien. *Artificial genetic adaptation in computer control systems*. Ph.D. dissertation, The University of Michigan Press, 1971. 3.2

[37] Ted J. Hubbard. *MEMS Design: The Geometry of Silicon Micromachining*. Ph.D. thesis, California Institute of Technology, Pasadena, CA 91125, July 1994. 4.1

[38] Ted J. Hubbard and Erik K. Antonsson. Design of MEMS via Efficient Simulation of Fabrication. In *Design for Manufacturing Conference*. ASME, August 1996. 1.1, 1.3.1, 4.1, 4.2, 7.2

[39] Ted J. Hubbard and Erik K. Antonsson. Cellular Automata in MEMS Design. *Sensors and Materials*, 9(7):437–448, 1997. 1.1

[40] Cezary Z. Janilow and Zbigniew Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. *The Forth International Conference on Genetic Algorithms*, 1991. 3.2, 3.5

[41] J.W.Boyse. Data structure for a solid modeller. 1979. 5.1

[42] Liang kai Huang and Mao jiun J. Wang. Efficient shape matching through model-based shape recognition. *Pattern Recognition*, 29:207–215, 1996. 5.1

[43] K.C.Ng, Y.Li, D.J.Murray-Smith, and K.C.Sharman. Genetic algorithms applied to fuzzy sliding mode controller design. *Genetic Algorithms In Engineering Systems: Innovations And Applications*, 1995. 3.9

[44] J. Koza. Evolution and co-evolution of computer programs to control independently-acting agents. *Animals to Animals*, 1991. 3.2

[45] Erwin Kreyszig. *Introductory Mathematical Statistics*. John Wiley and Sons, New York, 1970. 6.3.3

[46] Per B. Ljung and Martin Bachtold. Automatic model generation and analysis for mems. *Micro-electro-mechanical Systems (MEMS)*, 66:545–551, November 1998. 1.1

[47] Mark K. Long, Joel W. Burdick, and Erik K. Antonsson. Design of Compensation Structures for Anisotropic Etching. In *MSM'99, Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuators*, San Juan, Puerto Rico, April 1999. IEEE. 2.3

[48] Marc Madou. *Fundamentals of Microfabrication*. CRC Press, New York, 1997. 1.1

[49] Bernard Manderick, Mark de Weger, and Piet Spiessens. The genetic algorithm and the structure of the fitness landscape. In *The Fourth International Conference on Genetic Algorithms*, 1991. 3.3

[50] Fariborz Maseeh. Intellicad MEMS computer-aided design. In Jan G. Korvink, editor, *1997 CAD for MEMS Workshop Digest*, page 18, Switzerland, March 1997. Physical Electronics Laboratory, ETH Zurich. 1.1

[51] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1980. 1.2

[52] Z. Michalewicz, T. Logan, and S. Swaminathan. Evolutionary operations for continuous convex parameter spaces. *Proceedings of the Third Annual Conference on Evolutionary Programming*, 1994. 3.4.3, 3.5

[53] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Program.* Springer-Verlag, 1994. 3.2, 3.2, 3.4.2, 3.4.3, 3.8.2, 3.8.2

[54] M.J.Atallah. A linear time algorithm for the hausdorff distance between convex polygons. *Inf. Process. Letter*, 17:207–209, 1983. 5.2

[55] M.J.Atallah, C.C.Ribeiro, and S.Lifschitz. Computing some distance functions between polygons. *Pattern Recognition*, vol. 24, No. 8:775–781, 1991. 5.2

[56] Tamal Mukherjee and Gary K. Fedder. Structured design of microelectromechanical systems. In *Proceedings of the 1997 Design Automation Conference*, pages 680–685, New York, NY, June 1997. ACM SIGDA. Paper 42.3. 1.2, 2.2

[57] D. Mumford. The problem of robust shape descriptors. *Proceedings of First International Conference on Computer Vision*, 1987. 5.2

[58] W.G. Oldham, S.N. Nandgaonkar, A.R. Neureurather, and M.M. O'Toole. A general simulator for vlsi lithography and etching process: Part i - application to projection lithography. *IEEE Transaction Electron Device*, 26:717–722, 1979. 1.1

[59] W.G. Oldham, A.R. Neureurather, C. Sung, J.L. Reynolds, S.N. Nandgaonkar, and M.M. O'Toole. A general simulator for vlsi lithography and etching process: Part ii - application to deposition and etching. *IEEE Transaction Electron Device*, 27:1455–1559, 1980. 1.1

[60] I. M. Oliver, D. J. Smith, and J. Holland. A study of permutation crossover operators on the traveling salesman problem. *Proceedings of the Second International Conference*, 1987. 3.4.4

[61] P.Cox, H.Maitre, M.Minoux, and C.C.Ribeiro. Optimal matching of convex polygons. *Pattern Recognition Letter*, 9:327–334, 1989. 5.2

[62] Kurt E. Petersen. Silicon as a mechanical material. *Proceedings of the IEEE*, 70(5):420–457, May 1982. 1.1, 1.1

[63] B. Puers and W. Sansen. Compensation structures for convex corner micromachining in silicon. *Sensors and Actuators*, A21-23:1036–1041, 1990. 7.3.1

[64] N. Radcliffe. Genetic neural networks on mimd computers. *Ph.D. Thesis, University of Edinburgh, UK*, 1990. 3.4.3

[65] Kazuo Sato, Mitsuhiro Shikida, Yoshihiro Matsushima, Takashi Yamashiro, Kazuo Asaumi, Yasuroh Iriye, and Masaharu Yamamoto. Orientation-dependent silicon etching database allowing process-CAD for bulk micromachining. In Jan G. Korvink, editor, *1997 CAD for*

MEMS Workshop Digest, page 20, Switzerland, March 1997. Physical Electronics Laboratory, ETH Zurich. 3.3

[66] H. Schwefel. *Numerical Optimizaion of Computer Models*. John Wiley & Sons, Chichester, 1981. 3.4.3

[67] S. D. Senturia, N. Aluru, and J. White. Simulating the bahavior of mems devices: Computational methods and needs. *IEEE Computational Science and Engineering*, 4:30–43, January 1997. 1.1

[68] S. D. Senturia, R. M. Harris, B. P. Johnson, S. Kim, M. A. Shulman, and J. K. White. A computer-aided design system for microelectromechanical systems (MEMCAD). *Journal of Microelectomechanical Systems*, 1:3–13, March 1992. 1.1

[69] C. H. Sequin. Computer simulation of anisotropic crystal etching. *Sensors and Actuators A Physical*, 34(3):225–241, September 1992. 1.1, 2.3

[70] Mitsuhiro Shikida, Kazuo Sato, Kenji Tokoro, and Daisuke Uchikawa. Comparison of anisotropic etching properties between koh and tmah solutions. *Journal of Microelectromechanicanical Systems*, 1999. 7.3.1

[71] William M. Spears and Kenneth De Jong. An analysis of multi-point crossover. *Foundations of Genetic Algorithms*, pages 301–315, July 1991. 3.4.2

[72] William M. Spears and Kenneth De Jong. On the virtues of parameterized uniform crossover. *The Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991. 3.4.3

[73] Gilbert Syswerda. Uniform crossover in genetic algorithms. *The Fourth International Conference on Genetic Algorithms*, pages 2–9, 1989. 3.4.2, 3.4.3

[74] Gilbert Syswerda. A study of reproduction in generational and steady-state genetic algorithms. *Foundations of Genetic Algorithms*, pages 94–101, July 1991. 3.8.3

[75] O. Than and S. Buttgenbach. Simulation of anisotropic chemical etching of crytalline silicon using a cellular automata model. *Sensors and Actuators*, A45:85–89, 1994. 1.1

[76] T.P.Pavlidis. Polygonal approximation by newton's method. *IEEE Transaction Computation*, vol. C-26 No. 8:800–807, 1977. 5.1

[77] Darrell Whitley, T. Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. *The Fourth International Conference on Genetic Algorithms*, 1989. 3.3, 3.4.4

[78] L. Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Proceedings of Third International Conference Genetic Algorithms*, pages 116–221, 1989. 3.8.2, 3.8.2

[79] Alden H. Wright. Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms*, pages 205–217, July 1991. 3.2, 3.2, 3.4.3

[80] QingXin Zhang, Litian Liu, and Zhijian Li. A new approach to convex corner compensation for anisotropic etching of (100) si in koh. *Sensors and Actuators*, 56:251–254, 1996. 7.3.1

[81] Zhenjun Zhu and Chang Liu. Anisotropic crystalline etching simulation using a continuous cellular automata algorithm. *Micro-electro-mechanical Systems (MEMS)*, 66:577–582, November 1998. 1.1