A GENETIC ALGORITHM APPLIED TO THE OPTIMIZATION OF AIRFOIL DESIGN

By

PETER JAHNS

Bachelor of Science
United States Air Force Academy
Colorado Springs, Colorado
1985

Master of Education
Northwestern Oklahoma State University
Alva, Oklahoma
1993

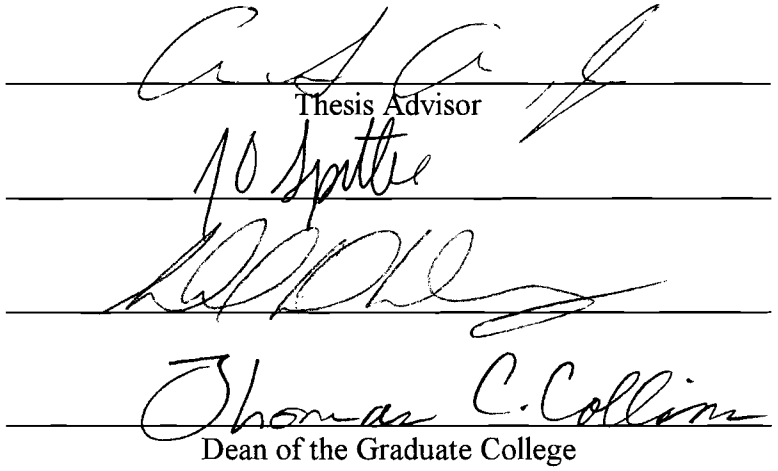
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1995

OKLAHOMA STATE UNIVERSITY

A GENETIC ALGORITHM APPLIED TO THE

OPTIMIZATION OF AIRFOIL DESIGN

Thesis Approved:



Thesis Advisor

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Andrew S. Arena, for his supervision, guidance, and inspiration. His support was invaluable not only with regard to the completion of this research, but throughout my graduate studies as well. Many thanks to the other members of my committee, Dr. R. D. Delahoussaye and Dr. D. G. Lilley, whose assistance was important to this project.

I am deeply grateful to my parents, Dr. Hans. O. Jahns and Mrs. Suse Jahns, for their ongoing support and encouragement.

I would especially like to thank my wife, Rhonda, for her understanding in times of difficulty, her optimism in the face of frustration, and her endless patience and love. I will be eternally grateful.

TABLE OF CONTENTS

Section Page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

c	airfoil chord
c_f	local skin friction coefficient
c_l	sectional lift coefficient (corrected for separation)
$c_{l,u}$	sectional lift coefficient (uncorrected for separation)
c_m	coefficient of pitching moment about the quarter-chord
c_p	coefficient of pressure
H	shape factor
H_{tr}	shape factor at the start of the transition region
MAXMUT	maximum allowable mutation per gene
n_{ind}	number of individuals per generation
n_{surv}	total number of survivors per generation
n_{best}	number of primary survivors (based on fitness alone)
n_{niche}	number of additional niche survivors (based on fitness and diversity)
n_{mut}	number of mutated offspring per survivor
n_{cross}	number of cross-breedings between survivors
P	probability factor
Re	Reynolds number*
s_{sep}	separation point (measured from trailing edge)*
V_i	local freestream velocity

V_{max}	maximum velocity along airfoil
V_∞	global (far-field) freestream velocity
x	chordwise location*
x_{cp}	center of pressure*
x_{sep}	separation point*
x_{tr}	transition point*
z	height, normal to chord*
α	angle of attack
δ	boundary layer thickness
δ^*	displacement thickness
Φ_i	velocity potential of individual singularities
Φ_∞	velocity potential of global freestream
Φ^*	total velocity potential
γ	circulation
η_u	upper airfoil surface
η_l	lower airfoil surface
η_t	thickness distribution
η_c	camber distribution
λ	Thwaites' dimensionless pressure gradient parameter
θ	momentum thickness

NOTE: * indicates that the parameter has been nondimensionalized by chord (c)

CHAPTER 1

INTRODUCTION

1.1. Research Objective

Optimization of airfoil design is a field whose roots extend back virtually to the dawn of aviation. Even marginal improvements in airfoil design may translate into tremendous gains in terms of aircraft speed, payload, loiter time, fuel costs, etc. Although optimization methods have changed enormously in the past century, the quest for improved performance continues unabated.

In the early days of aviation, airfoil optimization consisted largely of trial and error. Aerodynamic theory was in its infancy and designers had very little corporate knowledge to rely upon. Validation of aircraft performance consisted of human pilots flying new and often unproven designs, a rather imprecise and risky affair. With the advent of the wind tunnel, accurate data could be obtained under controlled conditions wherein failure was far less costly. Computer technology further revolutionized the airfoil design industry, as designs could quickly and accurately be evaluated before construction of any actual hardware. In recent years, computational fluid dynamics (CFD), with its ability to provide insight into complex flow phenomena, has become an indispensable tool in aircraft design.

Regardless of the methodology employed, it is possible that the performance of many airfoils could be improved through the implementation of an optimization scheme. The purpose of the current research effort was to determine the feasibility of using a genetic algorithm as an optimization tool for aerodynamic design. Specifically, a genetic algorithm was used to design a high lift airfoil, subject to constraints in angle of attack, thickness, and pitching moment coefficient. The effectiveness of the algorithm was evaluated and its range of applicability explored. Finally, some of the limitations of the technique, as well as its potential for further research, were revealed.

1.2. Literature Review

1.2.1. Direct Airfoil Design

In the direct approach to airfoil design, the geometry of the airfoil is specified first; thereafter, the resulting flowfield is evaluated. Flowfield parameters (velocity, pressure, temperature, density, Mach number, etc.) are used to calculate airfoil performance, from which a determination is made as to how well the design meets the desired criteria. Based upon the results, the airfoil geometry is altered and again evaluated. The direct method involves some subjectivity since the designer often relies upon his intuition and past experience to some degree. Due to the lack of any generally accepted airfoil theory, most airfoils designed before the 1930's were developed using the direct approach, including the common NACA 4- and 5-digit series airfoils which are still in use today [Abbott & Von

Doenhoff, 1959]. While this trial and error approach of repetitive analysis and intuition is not wholly without merit, the fact remains that it is quite inefficient.

1.2.2. Inverse Airfoil Design

The inverse approach, logically enough, works in reverse order. The desired features of the flowfield are specified first; an airfoil is then designed which will generate those flowfield parameters. Advances in boundary layer theory have made the inverse approach more tenable in recent years since aerodynamicists have a better understanding of the interaction between flowfield features, airfoil performance, and airfoil shape. For example, it is possible to calculate a freestream pressure distribution which maximizes the length of the laminar boundary layer along an airfoil. Once the requisite pressure distribution has been calculated, the airfoil which produces that distribution can be designed. Inverse design methods have been used extensively in the development of high lift airfoils [Liebeck, 1978, Eppler, 1990, and Selig & Guglielmo, 1994], transonic airfoils [Lee & Eyi, 1992], and a variety of fluid flow problems [Dulikravich, 1992].

However, the quality of the optimized shape still depends upon the ability of the designer to establish a desired optimum. Relying on experience and intuition, the designer may well specify flow parameters which do *not* correspond to a truly optimal condition. The resulting airfoil may meet the design specifications, but there is no assurance that it represents the best possible design.

1.2.3. Shape Perturbation Method

In contrast to the direct and inverse design methods described in the preceding sections, shape perturbation involves true mathematical optimization. This method consists of coupling an analysis code with a numerical optimizer to optimize an objective function. The analysis code can take the form of any suitable flowfield solver (Navier-Stokes, Euler, panel method, etc.). Cheung, Aaronson, & Edwards [1995] used a parabolized Navier-Stokes solver in conjunction with a nonlinear constrained optimizer to minimize pressure drag of a theoretical minimum-drag body at supersonic speeds.

The method has also been utilized to develop high lift, low drag transonic airfoils [Vanderplaats, 1984]. He represented a generic airfoil as a linear superposition of basis shapes, each one of which represented an existing airfoil. Mathematically:

$$\mathbf{Y} = a_1\mathbf{Y}_1 + a_2\mathbf{Y}_2 + \ldots + a_n\mathbf{Y}_n$$

where $\mathbf{Y}_1$ through $\mathbf{Y}_n$ are vectors containing the surface coordinates of the basis shapes, and a_i are the design variables or weighting factors.

He selected four existing good airfoil designs as basis shapes, thereby reducing the number of design variables and improving the numerical conditioning of the optimization problem. Design variable gradients were computed from finite difference approximations and an inviscid flow solver was used to evaluate the resulting aerodynamic performance.

His findings indicated that the shape perturbation method provided a significant improvement in transonic performance with relatively modest computational effort.

However, depending upon the nature of the function, shape perturbation may be rather intensive computationally because of the gradient evaluations involved; it may fail altogether for highly complex, non-linear functions.

1.2.4. Simulated Annealing

Simulated annealing is a combinatorial optimization technique based upon annealing of solids. In physics, annealing denotes a process in which a solid is heated to a liquid phase and then slowly cooled, such that the energy state of the substance decreases very gradually. Provided that the cooling is carried sufficiently slowly, the material particles arrange themselves in the lowest possible energy state at the completion of the process [Ohm, 1994]. If cooling takes place too rapidly, particles can become 'frozen' prematurely, resulting in crystal lattice defects and a final state which does *not* correspond to the lowest possible energy level.

In a simulated annealing process, the energy state is modeled by an objective cost function while temperature is a control parameter in the same units as the cost function [Kirkpatrick, Gelatt, & Vecchi, 1983]. The cooling schedule is simulated by decrementing the control parameter for each iteration by a small factor, α, slightly less than unity (typically between 0.80 and 0.99) i.e.:

$$T_{n+1} = \alpha T_n$$

For example, given a function $f(x, y, z)$ to be minimized and an initial solution (x_0, y_0, z_0), one or more of the variables are perturbed by a small amount. The function is re-evaluated at the new point (x_1, y_1, z_1) and a comparison is made between the two function evaluations. If f_1 is less than f_0, the new point is 'accepted'; if it is greater than f_0, it is accepted with probability $\exp(-\Delta f/T)$, the so-called Metropolis Monte Carlo function [Wang, 1991]. As the iterations progress, T decreases and uphill moves are accepted less and less frequently until, at zero 'pseudo-temperature', no more moves are possible.

This willingness to accept uphill moves, especially in the initial phases of the process, make simulated annealing much less susceptible to local minima than other gradient-type methods. However, if the solution domain is highly multi-modal and/or the minima are very deep, this technique may still become trapped in a local minimum. This shortcoming can be circumvented by carrying out the process several times, starting from a different random point for each test, and retaining the best results.

1.2.5. Genetic Algorithm Background

The mere fact that human beings exist on Earth is testimony to another optimization scheme which began four billion years ago. Starting with little more than basic elements and simple chemical compounds, the process of evolution has produced the

incredible diversity of life forms existing on Earth today. This sublime process serves as the inspiration for an optimization technique known as a genetic algorithm.

Plants and animals reproduce by transmitting chromosomes to their offspring. The chromosomes are comprised of a large number of genes, each of which determines a particular trait. During reproduction, the genetic information of two parents is combined to produce offspring with slightly different characteristics. Additionally, random mutations in one or more of the genes occur from time to time. While these mutations normally have a limited impact on the survivability of the offspring, they occasionally affect a crucial gene, and the mutated individual is markedly different from either of its parents. In the vast majority of cases, these mutations are detrimental and the affected individual, being less suited to its environment, dies. But once in a great while, the combination of cross-breeding and mutation produces an offspring that is better adapted to its environment, and the mutant reproduces and flourishes. Over time, only the most fit species survive, either by replacing similar species or by evolving to fill different ecological niches.

In the late 1950's, John Holland began experimenting with artificial life and machine learning. He was heavily influenced by *The Genetical Theory of Natural Selection*, written by the esteemed evolutionary biologist R. A. Fisher, which described the mechanism of evolution in mathematical terms. Holland realized that evolution, like learning, was a tool for adapting to an ever-changing environment. On the time scale of thousands of generations, evolution could be considered a form of optimization. He concluded that, if the salient features of evolution could be translated into the realm of mathematics, logic, and probability, then evolution itself could be modeled. Thus was born the genetic algorithm (GA), a design process that espoused design 'from the bottom up' [Levy, 1992].

A genetic algorithm begins with a population of potential solutions to a problem. Each solution is assigned a fitness value, reflecting the degree to which it satisfies the desired design criteria, and those solutions with the highest fitness values are selected for survival and reproduction. Genetic material (encoded representation of the solutions) is manipulated through cross-over and mutation. Aspects of the more promising solutions are retained, recombined, and mutated, while inferior solutions are discarded. Eventually, better solutions to the problem emerge.

To use a GA, the designer only defines what is to be maximized, imposes certain constraints, and lets the algorithm find solutions which best satisfy the design requirements. Human-imposed prejudices, while not entirely eliminated, are certainly reduced significantly. The genetic algorithm does not know or care what its human counterpart might find appealing or consider to be a 'good' design. Quite possibly, completely different and potentially superior solutions can emerge, unencumbered by the limitations of the designer's experience, intuition, or bias towards a particular type of design.

1.2.6. Previous Genetic Algorithm Applications

A particularly appealing aspect of genetic algorithms is their almost universal applicability. Provided that suitable representations for genes, individuals, mutations, and so on can be formulated, the algorithm can be applied to optimize virtually any design. The algorithm can efficiently search complex, multi-dimensional solution landscapes and the precision of the solutions is limited only by the accuracy of the fitness function. In con-

trast to calculus-based optimization schemes, the existence or continuity of derivatives is irrelevant, and the algorithm requires little or no mathematical manipulation beyond rudimentary sorting and selecting.

Genetic algorithms have been applied to a wide variety of optimization problems. Holland applied a GA to optimize strategies in a simplified version of chess, and one of his graduate students at MIT used an algorithm to simulate the functions of single-celled organisms [Levy, 1992]. Genetic algorithms have been utilized to minimize the weight of bridge truss structures, reduce the pressure loss in natural gas pipelines, and enhance the quality of X-ray photographs [Goldberg, 1989]. Charles Karr used a GA to obtain a better model for predicting the performance of a hydrocyclone, a device used to separate particles in the mineral processing industry [1995]. Other applications include the design of gas turbines and fiber-optic networks [Dumitrache, 1995].

Genetic algorithms have also been used in several inverse airfoil design methods. Obayashi & Takanashi implemented a GA to design an optimal pressure distribution which minimized transonic drag [1995]. Once the target distribution had been obtained, an inverse design code was used in conjunction with a Navier-Stokes solver to compute the corresponding airfoil geometry. Yamamoto & Inoue applied similar methodology in a L/D maximization problem [1995].

A particularly relevant application of the genetic algorithm was the development of a transonic airfoil by Quagliarella and Della Cioppa [1994], using the *direct* design approach. They modified the shape of a basis function (airfoil) by superimposing linear combinations of various modification functions. Each of the modification functions was

continuous and derivable, ensuring that any linear combination would also be continuous and derivable. All possible shapes were then given by:

$$y = y_b + \sum_{k=1}^{n} w_k f_k$$

where y_b represents the basis function,

f_k are the modification functions,

and w_k represent the related weighting factors (genes)

The genetic algorithm was implemented to minimize the wave drag of an airfoil, subject to constraints on c_l, Mach number, and airfoil thickness, by finding the optimum combination of weighting factors, w_k. Using a symmetric airfoil as the basis shape, the algorithm was able to reduce the wave drag by 30% in 354 generations. When the experiment was conducted with a 'pre-optimized' basis shape (cambered airfoil) a similar reduction in wave drag was achieved after only 68 generations. By selecting an initial shape which was already higher up the evolutionary chain, computer run time was reduced by 80%, indicating the effect of initial design selection on algorithm efficiency.

1.2.7. General Considerations

While the genetic algorithm concept may be rather intuitive, the implementation of a GA requires careful consideration. The success of the algorithm depends upon a number

of factors such as the nature of the solution landscape (including the number and type of design variables), design constraints, selection criteria, implementation of mutation and/or crossover, maximum allowable mutation size, and population size. Moreover, the specification of these parameters may have a significant impact on the final 'optimal' solution and can be quite problem-dependent. Research by Srinivas and Patnaik [1994] indicates that " . . . the choice of the control parameters itself can be a complex nonlinear optimization problem. Further, it is becoming evident that the optimal control parameters critically depend on the nature of the objective function."

Due to the high degree of randomness inherent in genetic algorithms, they do not lend themselves to rigorous mathematical proof. Mutation, mate selection, and survival selection are all accomplished in a random or pseudo-random fashion; this 'directed randomness' is the very heart of genetic algorithms. However, it should be emphasized that randomness is only one *component* of a genetic algorithm; Section 2.3 will illustrate some of the key distinctions between a GA and a purely random search. Because of the inherently random element, it is impossible to verify whether or not a particular design truly represents an optimum solution; the viability of individual designs can only be verified by comparing them to existing, known solutions.

These limitations notwithstanding, several observations can be made regarding the behavior of genetic algorithms in general.

1.2.7.1. Solution Landscape

If the nature of the solution landscape is well known in advance, a genetic algorithm may not be the most effective optimization technique in the first place. For functions which are simple and continuous, calculus-based techniques such as those of Newton and Fibonacci can be just as effective and far simpler to implement than a GA. Although purely random searches or enumerative techniques (which conduct a point-by-point search) are not especially elegant, they may be preferable to more sophisticated methods if the domain space is small and low-dimensional [Filho, Treleaven, & Alippi, 1994].

For more complex optimization problems in which the application of a GA is indicated, it is usually desirable to minimize the number of genes so as to reduce the epistaticity of the system. Highly epistatic relationships, in which the genes interact in a complex, extremely non-linear fashion, are not well suited to optimization by genetic algorithms since so many favorable mutations must occur simultaneously to produce any improvement to the proposed solution [Srinivas & Patnaik, 1994]. Furthermore, algorithm performance is enhanced if all of the design variables are of the same *type*, although this requirement is relaxed somewhat if the total number of genes is small. When optimizing a bridge truss structure, for example, the algorithm would be more effective if all of the genes represented individual beam sizes (with number of beams, truss geometry, and construction materials all being constant) than would be the case if all of these parameters were allowed to vary simultaneously.

1.2.7.2. Design Constraints

Like most other optimization methods, genetic algorithms usually operate in a constrained solution space. For some problems, the constraints may be as straightforward as placing upper and lower bounds on the range of allowable values of each gene. Even with such restrictions in place, however, certain combinations of genes may result in designs which are infeasible from a practical standpoint. A sound understanding of the engineering principles involved in a particular application is therefore required if the results are to be physically meaningful.

The function of a genetic algorithm is merely to propose alternate solutions to a particular problem; it does not, by itself, determine the quality of those solutions. In order to calculate the fitness of a given solution, a 'fitness solver', which is normally called as a subroutine in a GA computer code, is required. In other words, the GA plays by the rules specified by the fitness solver, whether or not those rules are physically realistic. Furthermore, the fitness solver may be more accurate in some regimes than in others and the algorithm can cleverly exploit these limitations by populating regions in the solution domain where the fitness solver overestimates design performance. Consequently, it may be necessary to constrain the search space in order to circumvent the limitations of the fitness solver itself.

1.2.7.3. Selection Criteria

The simplest selection criterion is the standard selection method; the most fit individuals of each generation are designated the parents of the subsequent generation. However, this approach invariably leads to population uniformity and premature convergence to local optima [Winston, 1992].

An alternate method, the rank method, involves rank-ordering the individuals from the most to the least fit. Survivors are selected according to the following probabilistic method. The highest-ranking candidate is selected with probability P (generally in the range of 0.5 to 0.9). If the top-ranked candidate is not selected, the one ranked number two is picked, again with probability P. This selection process continues until either an individual is selected or only one individual (the lowest-ranking one) remains, in which case the lowest-ranking candidate is selected. While this method does reduce overcrowding somewhat, the population still tends to coalesce about a single 'niche' [Winston, 1992].

The efficiency of the algorithm can be improved markedly by using the rank-space method, which incorporates diversity in the selection process. In other words, the probability of an individual's survival is determined not only by its fitness, but also its diversity from the other candidates. By incorporating diversity, the algorithm simulates the selection process evident in the biological realm, wherein diverse organisms can utilize resources for which the competition is less fierce than in more crowded niches. In his book Artificial Intelligence, Patrick Winston demonstrated the importance of diversity in a two-dimensional non-linear optimization problem. With diversity incorporated in the selection

process, the algorithm required only one fifth as many generations to find the global maximum as when fitness alone was used to select survivors [1992].

1.2.7.4. Mutation and Crossover

The two primary operators which drive any genetic algorithm are mutation and crossover. A mutation is a small random variation in one or more of the genes, accomplished through the use of a computer-generated random number, while crossover consists of combining genes from two previously selected survivors. The relative importance of the two operators is determined to some degree by the solution landscape itself.

In the case of smooth, well-behaved functions, mutations alone can successfully 'hill-climb' local peaks, but are often unable to traverse local minima (moats). Any individuals which mutate away from the local maximum receive a lower fitness ranking and are unlikely to survive to the next generation. Without crossover, the only way an individual can cross a moat is if extremely large mutations are allowed. Excessive mutation size, however, results in a more random search in which one generation bears little resemblance to the previous one. Moreover, one would have to know in advance the size of mutations required to traverse the local minima, which in turn requires that the extent of the minima themselves be known. If one had such detailed information about the solution landscape, there would be no reason to resort to the genetic algorithm in the first place.

Conceivably, there might be conditions under which mutations alone could be adequate. If a designer wished to fine-tune a particular design, he might be uninterested in

finding solutions corresponding to other local maxima. For this type of optimization, mutations alone may produce satisfactory results.

In contrast, crossover enables the algorithm to search multi-modal landscapes much more efficiently by 'skipping over' local minima. In a simple two-dimensional problem, for example, a good 'x' gene of one survivor could combine with a favorable 'y' gene of another to produce a child whose genes placed him on the other side of the moat in a single generation. Without crossover, however, the algorithm may well become trapped in a local peak and never seek out other, potentially superior solutions. Therefore, unless one is quite certain of the objective nature of the problem to be optimized and/or is willing to consider only local optima, crossover should always be selected - it may be essential to the success of the algorithm.

1.2.7.5. Maximum Allowable Mutation Size

Most of the early work with genetic algorithms focused on strings of binary digits. When operating on these strings, mutations merely involved toggling a bit from 0 to 1 or vice versa [Levy, 1992]. In the case of real-values genes, however, maximum mutation size (MAXMUT) must be restricted to some reasonable value if the algorithm is to function properly. If MAXMUT is too large, the GA degenerates into a random search; if it is too small the algorithm cannot effectively explore the solution space. Maximum allowable mutation size must therefore be tailored to suit the needs of the particular optimization problem under scrutiny.

1.2.7.6. Population Size

The number of individuals per generation, or population size, can also influence the efficiency of the GA. Increasing the population size increases the diversity of the gene pool and reduces the possibility that the GA will prematurely converge to a local maximum. On the other hand, larger populations increase the computer time required for the population to converge to the optimal regions in the search space [Srinivas & Patnaik, 1994].

1.2.8. Comparison Between the Genetic Algorithm and Simulated Annealing

Of all the design methods described in this chapter, simulated annealing bears the most resemblance to the genetic algorithm. Both techniques are effective combinatorial optimization schemes characterized, in part, by an inherently random component. Furthermore, each method has a mechanism to avoid getting trapped in local optima; GAs incorporate diversity in the selection process while simulated annealing relies upon the cooling schedule. The crucial difference between the two methods is the presence or absence of a crossover mechanism. With crossover, genetic algorithms can recombine favorable aspects of previously discovered solutions, thereby constantly introducing new and potentially beneficial genetic material into the population, a capability which is absent

from the simulated annealing process. Very broadly speaking, one could liken simulated annealing to a GA without crossover, whose maximum allowable mutation size decreases with every generation.

The observations in the preceding sections are generally applicable to most genetic algorithm applications. Before applying a GA to an airfoil optimization problem, however, a more detailed *quantitative* understanding of the GA control parameters was necessary. The development and validation of the genetic algorithm code used in this study will be discussed in Chapter 2.

CHAPTER 2

ALGORITHM DEVELOPMENT AND VALIDATION

In order to illustrate the operation of the genetic algorithm, an algorithm was developed to optimize two-dimensional mathematical functions. This phase of the design process represented an attempt to optimize the optimizer under conditions where the solution to the problem was known in advance. The well-known 'sombrero' function shown below ($z = \sin(r) / r$) represented an ideal test case; it has one global maximum located at the origin and an infinite number of local maxima separated by an infinite number of local minima, yet is relatively well-behaved and simple to visualize.

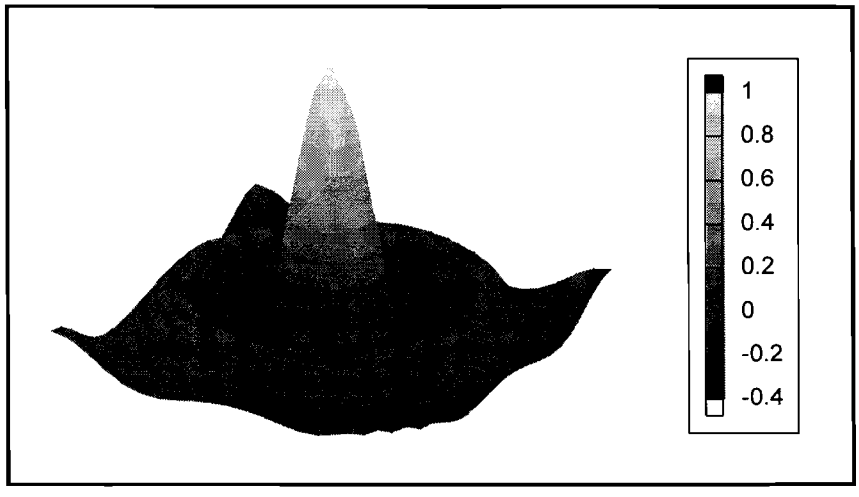


Figure 2.1. Simple Solution Landscape

2.1. Definition of Parameters

In order to apply the genetic algorithm to the aforementioned function, the following parameters were modeled in the computer-simulated environment:

1. Gene: an x or y value, each constrained to a min/max value of ± 10.
2. Chromosome (Individual): an (x,y) pair.
3. Mutation: a random variation in a gene, supplied by a computer-generated random number.
4. Crossover: a process in which the x gene of one survivor is combined with the y gene of another.
5. Environment (Solution Domain): two-dimensional Cartesian space, centered on the origin, and extending ± 10 in each dimension.
6. Fitness: $\frac{\sin(r)}{r}$, where $r = \sqrt{x^2 + y^2}$
7. Constraints: the boundaries of the solution domain, i.e. $(-10 \leq x \leq 10)$ and $(-10 \leq y \leq 10)$.

2.2. Implementation

Once the parameters had been quantified, the genetic algorithm was executed in the following sequence:

1. Given a population of (x,y) chromosomes, the fitness value was determined for each one in a FITNESS SOLVER subroutine; subsequently, they were rank-ordered from the most to the least fit. The most fit individual was automatically selected for survival and was designated as the primary survivor.

2. The remaining individuals were rank-ordered by diversity according to the following relation:

$$DIV_i = \frac{1}{d_i^2} = \frac{1}{(x_i - x_{ps})^2 + (y_i - y_{ps})^2}$$

where subscripts i and ps refer to the particular individual and the primary survivor, respectively. The combined rank was determined by ranking each individual according to the sum of its fitness rank and diversity rank, with one of the two rankings designated as the tiebreaker. Additional survivors were selected based upon their combined rank, according to the probabilistic method outlined in Chapter 1. The probability factor, P, controlled the amount of bias towards the selection of the most fit/diverse individuals. An example of the rank-space method, with diversity designated as TIEBREAK, is tabulated below:

(x,y)	Fitness	Fitness Rank	Diversity	Diversity Rank	Rank Sum	Combined Rank	Probability of Selection (P = 0.7)
(0, -8)	0.1237	1	*	*	*	*	*
(5,5)	0.1002	3	.0052	1	4	1	0.700
(-7,0)	0.0934	4	.0088	2	6	2	0.210
(1,8)	0.1214	2	1.0	4	6	3	0.063
(0, -4)	-0.1892	5	.0625	3	8	4	0.027

Table 2.1. Rank-Space Selection Method

Probabilistic selection was accomplished by assigning each candidate (excluding the primary survivor) a segment of the interval [0.0, 1.0]; the size of each candidate's segment determined its probability of selection. In the example above, the first candidate was assigned the interval [0.0, 0.7], the second one, (0.7, 0.91], and so on. A random number in the interval [0.0, 1.0] was generated, and its value determined which individual was chosen to accompany the primary survivor into the next generation. The individual thus selected was labeled the 'first niche survivor'.

Subsequently, remaining candidates were rank-ordered according to their diversity from *both* the primary survivor and the first niche survivor, i.e., $DIV = \Sigma(1/d_i^2)$. This particular specification ensured that DIV reflected a candidate's diversity from *all* previously selected survivors. The rank-space method was again utilized to select the second niche survivor, and so on, until a full complement of $1+n_{niche}$ survivors had been selected. These survivors became the parents of the next generation.

3. After the selection process was completed, a new population was created by mutating and/or breeding the parents. Mutated offspring were created by applying small random alterations to the genes of each survivor; the maximum allowable mutation per gene was restricted by the user-defined variable MAXMUT. In addition, each parent was paired with another (randomly selected) parent and their genes were crossed, producing two more 'children'. The complete population of any generation, therefore, consisted of:

a. n_{surv} unmated parents (survivors from the previous generation)

b. n_{surv} mutated offspring

c. 2 x n_{surv} cross-bred offspring

In order to allow greater flexibility to the scheme outlined above, several modifications were incorporated. The user could specify the number of primary survivors (n_{best}) as well as the number of additional niches (n_{niche}) to be filled. A high value of n_{best} would presumably lead to a more uniform, relatively fit population in which the individuals coalesced about a few local maxima and ignored other potential solutions. In contrast, higher values of n_{niche} would explore the solution space more efficiently, albeit at the cost of greater computational effort per generation.

Two other variables, n_{mut} and n_{cross}, were introduced to determine the effects of mutation and crossover individually. After these modifications were incorporated, the total number of individuals per generation, n_{ind}, was given by:

$$n_{ind} = (n_{best} + n_{niche}) \times (1 + n_{mut} + 2(n_{cross}))$$

An initial 'seed' population was generated by randomly populating a portion of the solution domain. The size of the initially populated region was varied to determine the effect of seed population diversity. The seed domain was a square region of varying size located at the lower left corner of the solution domain (-10,-10), placing it as far as possible from the global maximum at the origin. Therefore, a 1x1 seed domain covered only 0.25% of the total solution domain, while a 20x20 region effectively covered the domain in its entirety. This seed domain was used *only* to generate the initial population; subsequent generations were free to explore the entire 20x20 region.

The algorithm was executed hundreds of times using different combinations of control parameters. The effect of individual parameters was isolated by varying a single parameter while holding all others constant. Table 2.2 lists the control variables, the standard values about which variations were applied, and the range of values considered.

Parameter	Standard Value	Range of Values
n_{best}	1	1 - 10
n_{niche}	3	1 - 10
n_{mut}	1	1 - 10
n_{cross}	1	0 - 5
TIEBREAK	Diversity Rank	Diversity/Fitness Rank
MAXMUT	1.0	0.1 - 10.0
Seed Domain Size	10x10	(1x1) - (20x20)
P	0.7	0.5 - 1.0

Table 2.2. Genetic Algorithm Control Parameter Values

Figure 2.2 depicts a flowchart which summarizes the solution methodology of the genetic algorithm computer code used in this study.

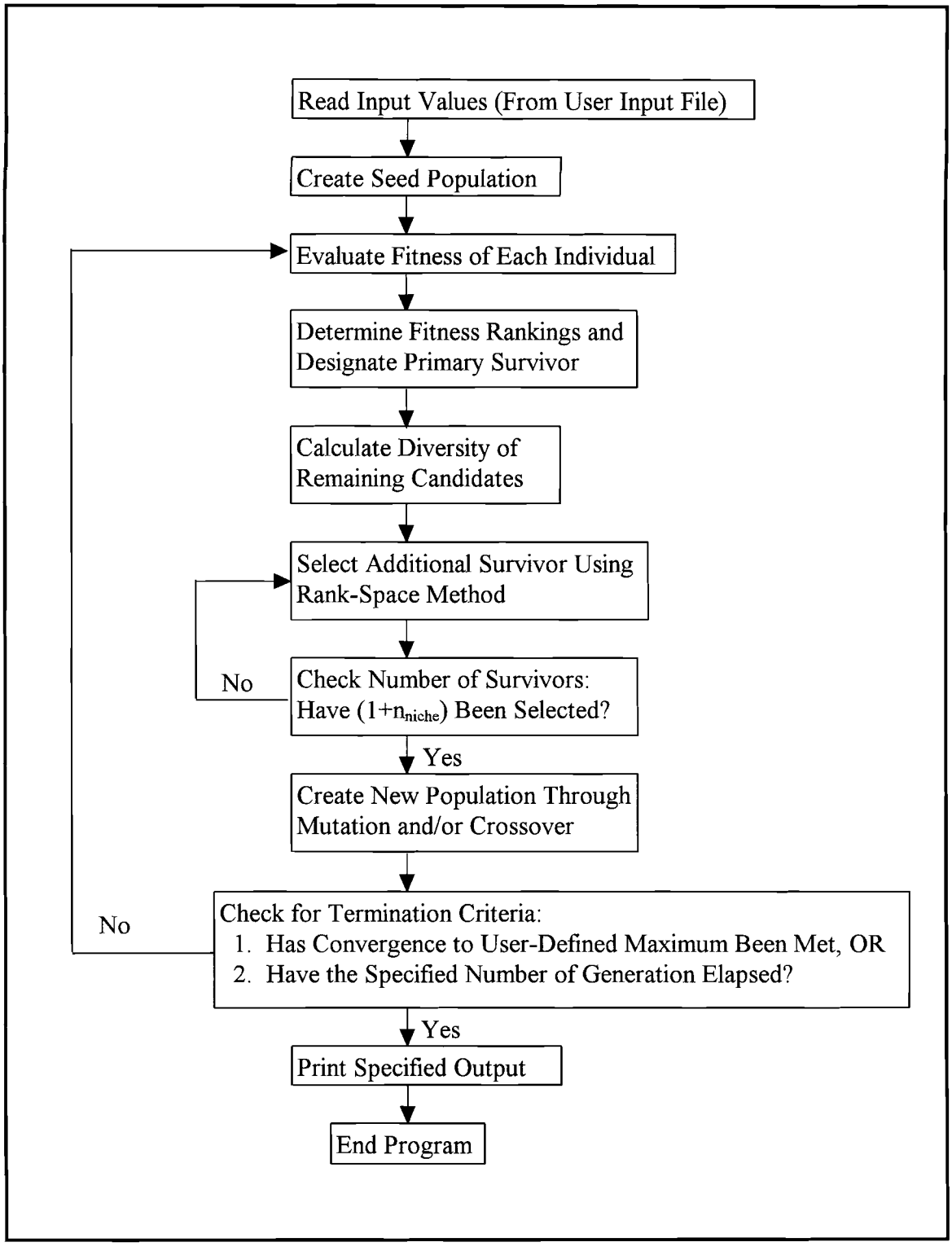


Figure 2.2. Genetic Algorithm Flow Chart

With any given set of parameters, the algorithm was allowed to run for 1000 generations or until the global maximum was located. If the algorithm converged to a solution with fitness $z \geq 0.999$ (99.9% of the mathematical optimum), the test was considered a success and the number of generations recorded. In the event that the global maximum was not found in 1000 generations, the test was considered unsuccessful, although, for averaging purposes, 1000 generations was recorded. For a given set of parameters, the experiment was conducted ten times and the average number of generations required for convergence was calculated. Thereafter, one of the control parameters was altered, and the algorithm was again applied for ten test runs.

2.3. Results

The results revealed that some of the control parameters were vital to the success of the algorithm while others were relatively unimportant. Two factors were considered when evaluating the efficiency of the algorithm: average number of generations and total computational effort required for convergence. In some cases, the algorithm converged in fewer generations, yet required significantly more computational effort due to a larger population size. The results are tabulated below:

Parameter	Value	n_{ind}	Success (out of 10)	Avg. # of Generations (n_{gen})	Computational Effort (n_{ind} x n_{gen})
n_{best}	1*	16	10	87.3	1396.8
	3	24	10	87.6	2102.4
	5	32	10	75.5	2416.0
	10	52	10	80.0	4160.0
n_{niche}	1	8	2	900.7	7205.6
	3	16	10	87.3	1396.8
	5*	24	10	56.6	1358.4
	10	44	10	39.8	1751.2
n_{mut}	1*	16	10	87.3	1396.8
	3	24	10	79.3	1903.2
	5	32	10	73.4	2348.8
	10	52	10	72.1	3749.2
n_{cross}	0	8	1	925.8	7406.4
	1*	16	10	87.3	1396.8
	3	32	10	74.1	2371.2
	5	48	10	57.9	2779.2
TIEBREAK	Diversity*	16	10	87.3	1396.8
	Fitness	16	10	103.7	1659.2
MAXMUT	0.1	16	8	431.6	6905.6
	1.0*	16	10	87.3	1396.8
	10.0	16	9	178.2	2851.2
Seed Domain Size	1x1	16	9	319.1	5105.6
	5x5	16	10	163.3	2612.8
	10x10	16	10	87.3	1396.8
	20x20*	16	10	65.9	1054.4
P	0.5	16	10	141.0	2256.0
	0.7*	16	10	87.3	1396.8
	0.9	16	10	96.6	1545.6
	1.0	16	10	95.0	1520.0

Table 2.3. Effect of Control Parameters on GA Convergence
(* indicates least computational effort required)

The data gathered here is, by definition, empirical in nature; it does not *prove* that a particular set of control parameters truly represents an optimal combination. The selec-

tion of parameters, as well as the actual values of those parameters, involved a good deal of subjectivity. The control parameters may interact in a complex, non-linear fashion; quite possibly, the effects of individual parameters can *not* be determined by varying parameters independently. Unexplored combinations might have yielded faster convergence and the intrinsic randomness of the algorithm introduced an additional element of uncertainty. Finally, this experimentation was conducted for only one particular optimization problem; as mentioned before, the optimal combination of parameters is highly problem-dependent.

These limitations notwithstanding, several conclusions were drawn from the results. An increase in n_{best} or n_{mut} yielded only a marginal improvement in algorithm performance, yet required considerably more computational effort. The principal effect of these parameters was to overpopulate the primary niche; they contributed very little to the richness of the gene pool. Since their contribution to evolution was negligible, n_{best} and n_{mut} were retained at their minimum value so as to reduce the total number of calculations required.

On the other hand, convergence to the global optimum was substantially enhanced as seed domain size, n_{niche}, or n_{cross} increased. These parameters significantly augmented the diversity of the genetic material available for recombination. A large initial domain populated the solution landscape relatively evenly and enhanced the efficacy of the algorithm, especially during the early stage of evolution. In contrast, smaller starting regions translated into a comparatively homogeneous original gene pool from which the GA required numerous generations to really become effective.

Without crossover, the power of recombination was lost, and better chromosomes were only achievable through mutations. Table 2.3 shows that the algorithm was successful only one of ten times when $n_{cross} = 0$, as compared to a 10 of 10 success rate when crossover was selected. As mentioned before, mutations alone are rather ineffective in traversing local minima and this effect was clearly demonstrated here.

Similarly, when only one additional niche was specified (two survivors total), the algorithm converged only twice. Even with crossover selected, the limited amount of genetic material available for recombination seriously hampered algorithm effectiveness. Although an increase in either operator did require more computer time, the increase was less marked for n_{niche} than for n_{cross}, while the marginal improvement was more significant. Therefore, it was decided that increasing the number of niches was warranted, but an increase in crossover frequency was not.

The algorithm seemed to be relatively unaffected by the specification of the tie-breaker, although favoring diversity yielded somewhat faster convergence. Similarly, the results were quite insensitive to changes in the probability factor at or above 0.7. Presumably, with a P value of 0.5, selection of survivors was insufficiently biased towards the most fit/diverse individuals, and resulted in a more random search. Finally, it was observed that a MAXMUT value of 1.0 accelerated convergence; a smaller value hampered efficient exploration of the domain, while a larger one resulted in a more random search.

Based upon these results, the only changes made to the standard parameter set were to populate the entire solution domain with the initial population (seed domain = 20x20) and to increase the number of niches from 3 to 5. Using this new optimal parameter set, which had 6 survivors and 24 individuals per generation, the average number of

generations required to locate the global maximum was 35.1 at a total computational effort of 842.4, a 40% improvement over any previously obtained results.

For the sake of comparison, a purely random search of the solution domain was conducted, using 24 random points at a time. In this random search, no information was transmitted from one set of points to the next, so the concept of a generation was meaningless; each set of 24 points was considered a 'generation' only for comparison's sake. A comparison between the two methods is tabulated below:

Search Method	n_{ind}	Success (out of 10)	Avg. # of Generations (n_{gen})	Computational Effort (n_{ind} x n_{gen})
GA (Optimal Parameter Set)	24	10	35.1	842.4
Random	24	7	622.5	14,940.0

Table 2.4. Comparison Between Genetic Algorithm and Random Search Methods (Simple Solution Landscape)

Table 2.4 reveals that the genetic algorithm was much more efficient in that, on average, it sampled only 5% as many points as the random search method before converging to the global maximum. The maximum and average population fitness using each search method is depicted in Figure 2.3.

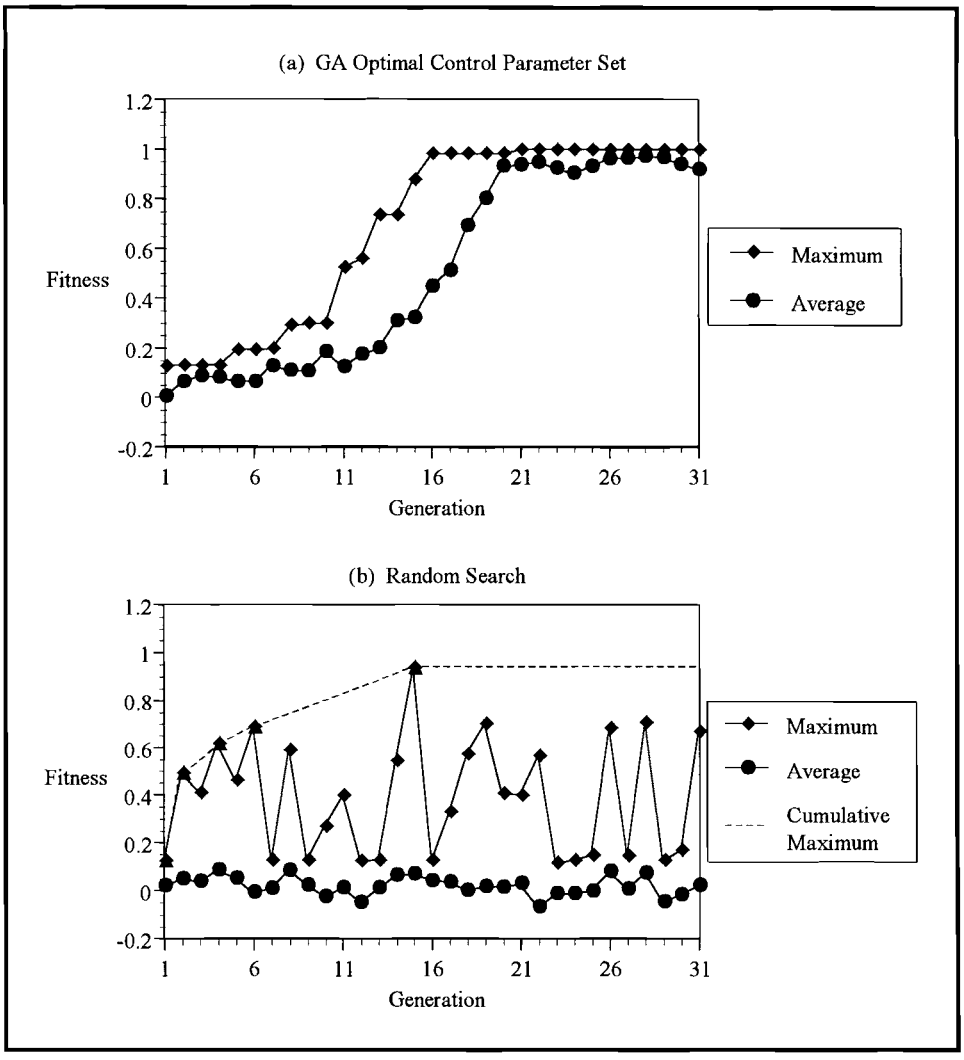


Figure 2.3. Maximum and Average Population Fitness (Simple Solution Landscape)

Interestingly, this figure simultaneously illustrates the major similarity as well as the crucial difference between the two search schemes. In Figure 2.3(a), the step-wise improvement in maximum fitness is due to the discovery of a particularly fortunate combination of genes, revealing the random aspect of the GA. If one considers the cumulative maximum of the random technique, one could point out that it also increases in irregular,

discrete increments. In fact, the random search located a fitness value *near* the global maximum (0.963) in generation 15, although it never found the global peak in this test run. The key distinction between the two methods is revealed by a comparison of the *average* fitness graphs. The random search produces an average fitness value which is relatively constant over time. However, in the GA application, each jump in maximum fitness is followed by a very definite, although more gradual, increase in the fitness of the population *as a whole*. This retention of knowledge and the subsequent dissemination of genetic information into the population at large is what distinguishes the genetic algorithm from the purely random search.

The qualitative nature of the fitness curves in Figure 2.3(a) is very characteristic of genetic algorithms. Biologists have noted the same trend in natural evolution: sudden spurts of change followed by long periods of relative quiescence. Although the gene pool is still seething with activity during the dormant phases, the jumps in maximum fitness occur only when an especially advantageous gene combination is found, corresponding to the discovery of a new evolutionary niche. The resulting super-organism then spreads its genes into the population, and average fitness level of the entire population is enhanced correspondingly.

The cause of this stairstep evolution is revealed in Figure 2.4. In each diagram, the survivors of that particular generation are denoted by '**x**' symbols. The survivors of the seed population (Figure 2.4(a)) tended to lie near the ring-like niche at $r \approx 7.5$, where fitness was roughly 0.125. After 11 generations, a chromosome with fitness 0.57 had emerged in the central niche, although average population fitness remained relatively unchanged. However, within the ensuing several generations, both maximum and average

population fitness improved dramatically as superior genetic information diffused into the population. By generation 16, the *average* fitness level was nearly as high as the maximum fitness present only five generations previously. In the final few generations, the fitness levels had stabilized near 1.0, indicating another period of evolutionary dormancy.

In contrast, the evolutionary pattern evident in Figure 2.4 is completely absent from the purely random search, as indicated in Figure 2.5.

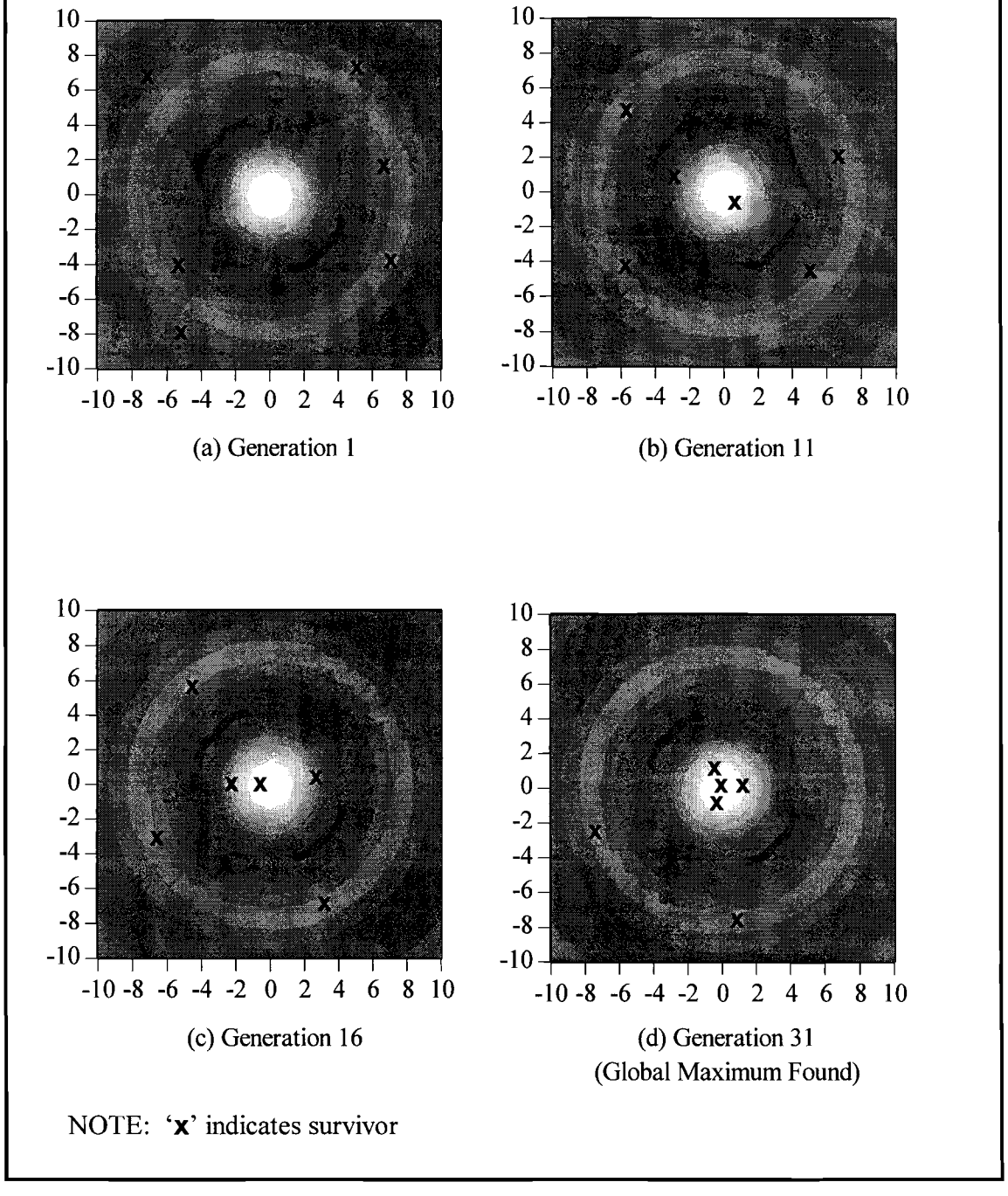


Figure 2.4. Evolution of Genetic Algorithm Solutions
(Simple Solution Landscape)

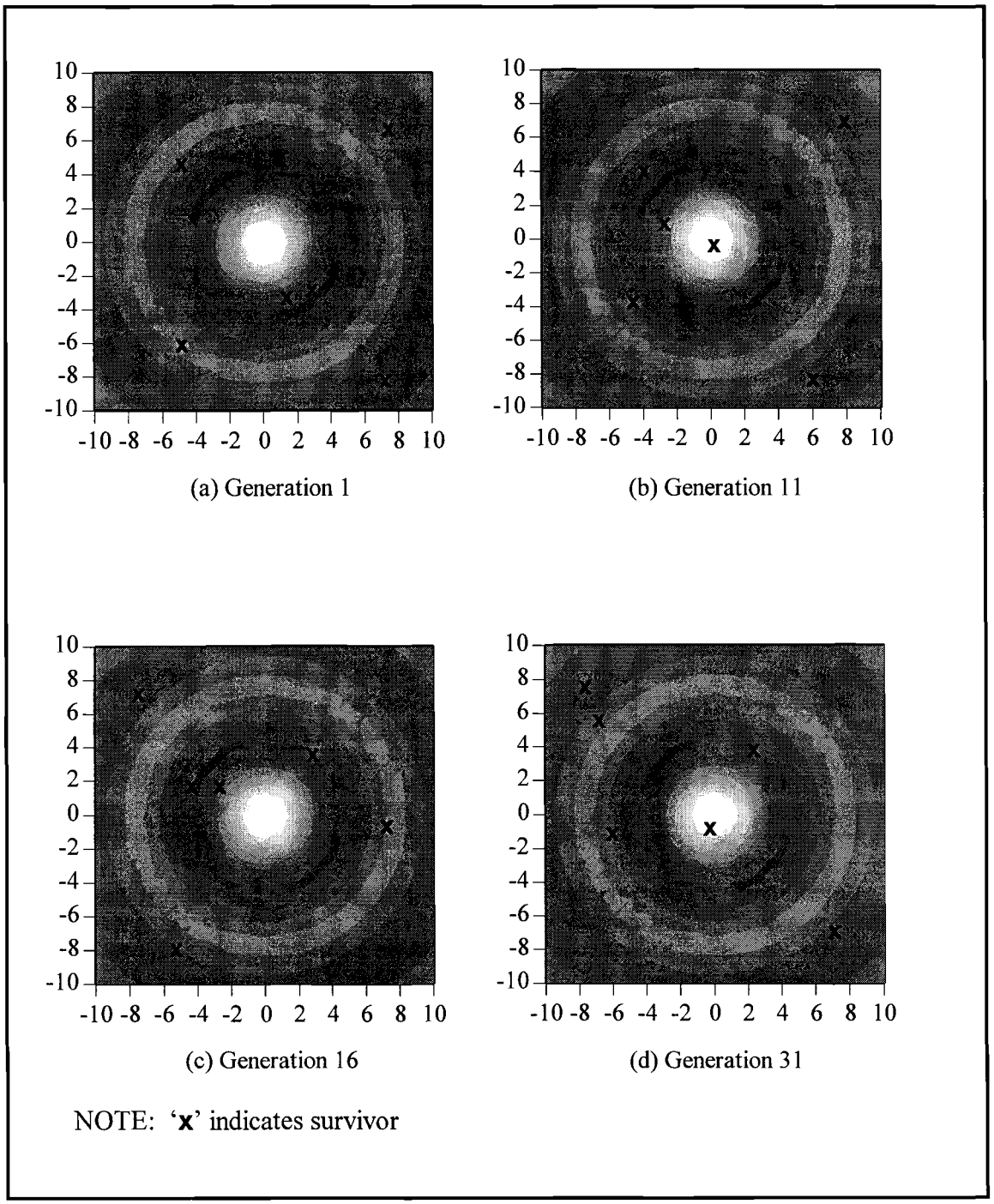


Figure 2.5. Evolution of Random Search Solutions
(Simple Solution Landscape)

As mentioned previously, the global maximum of this sample fitness function was defined as $z \geq 0.999$, corresponding to 99.9% of the true mathematical maximum. In most engineering applications, however, mathematical optima are rarely attainable due to modeling inaccuracies and physical limitations. Considering that a simple, intuitive random search identified a solution which was 96.3% as fit as the theoretical optimum after evaluating only 360 points, one might well ask if the marginally better fitness of the GA solutions was worth the effort. As indicated in Section 1.2.7.1, a random search or a brute force enumerative (point by point) technique may be preferable to the genetic algorithm in cases where the domain space is small and low-dimensional, as was certainly the case for this application. Moreover, since the solution landscape had a broad, flat region of highly fit solutions near the global peak, it was easy to find a solution that was almost as good as the true optimum.

The true potential of a genetic algorithm is not realized until the solution domain becomes larger and/or more complex, domains for which random or enumerative searches become prohibitively inefficient. This characteristic was demonstrated by considering a more complex two-dimensional solution landscape, namely, the graph of the function:

$$z = \left[\frac{1}{\left(\frac{x}{2}+8\right)^2+\left(\frac{y}{2}-4\right)^2+\frac{1}{2}}\right]+\left[\frac{1}{\left(\frac{x}{2}-6\right)^2+\left(\frac{y}{2}+7\right)^2+\frac{1}{2}}\right]+\left[\frac{1}{\left(\frac{x}{2}-6\right)^2+\left(\frac{y}{2}-4\right)^2+\frac{1}{2}}\right]^2$$

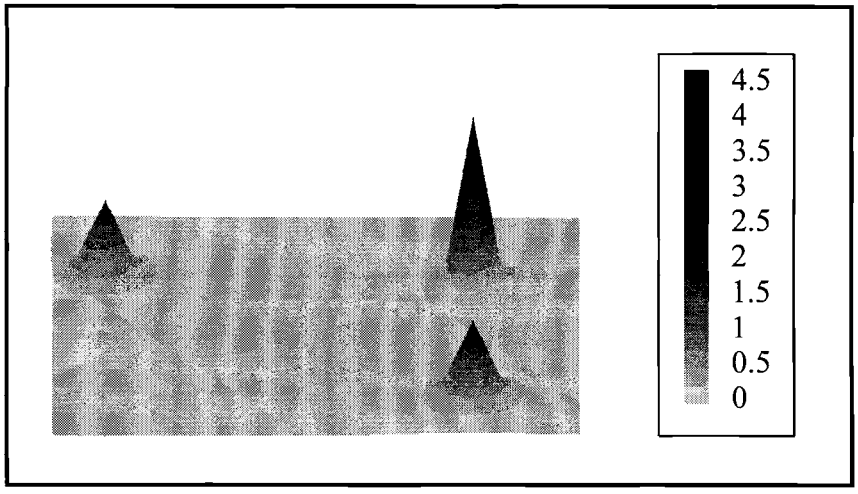


Figure 2.6. Complex Solution Landscape

This function has two local maxima of $z \approx 2.0$ and a global maximum of $z \approx 4.0$. Using the same methodology as before, a comparison was made between the effectiveness of the two optimization schemes when applied to this more difficult problem, using the same procedures and parameters as in the simpler test case. For the purpose of comparing convergence rates, either method was considered successful if a solution with fitness 3.8 (95% of the mathematical optimum) was identified. Table 2.5 indicates that the GA required far less computational time to locate the maximum than the random search; indeed, the latter method failed 8 of 10 times in this experiment. Furthermore, Figure 2.7 indicates that, in the number of generations required for the GA to converge to the peak, the random search had located a solution that was only about 50% as fit. Finally, Figures 2.8 and 2.9 illustrate evolution of solutions for each search method. Once again, the evolutionary nature of the genetic algorithm, which is completely absent from the random search, is revealed.

Search Method	n_{ind}	Success (out of 10)	Avg. # of Generations (n_{gen})	Computational Effort (n_{ind} x n_{gen})
GA (Optimal Parameter Set)	24	10	24.3	583.2
Random	24	2	918.9	22,053.0

Table 2.5. Comparison Between Genetic Algorithm and Random Search Methods (Complex Solution Landscape)

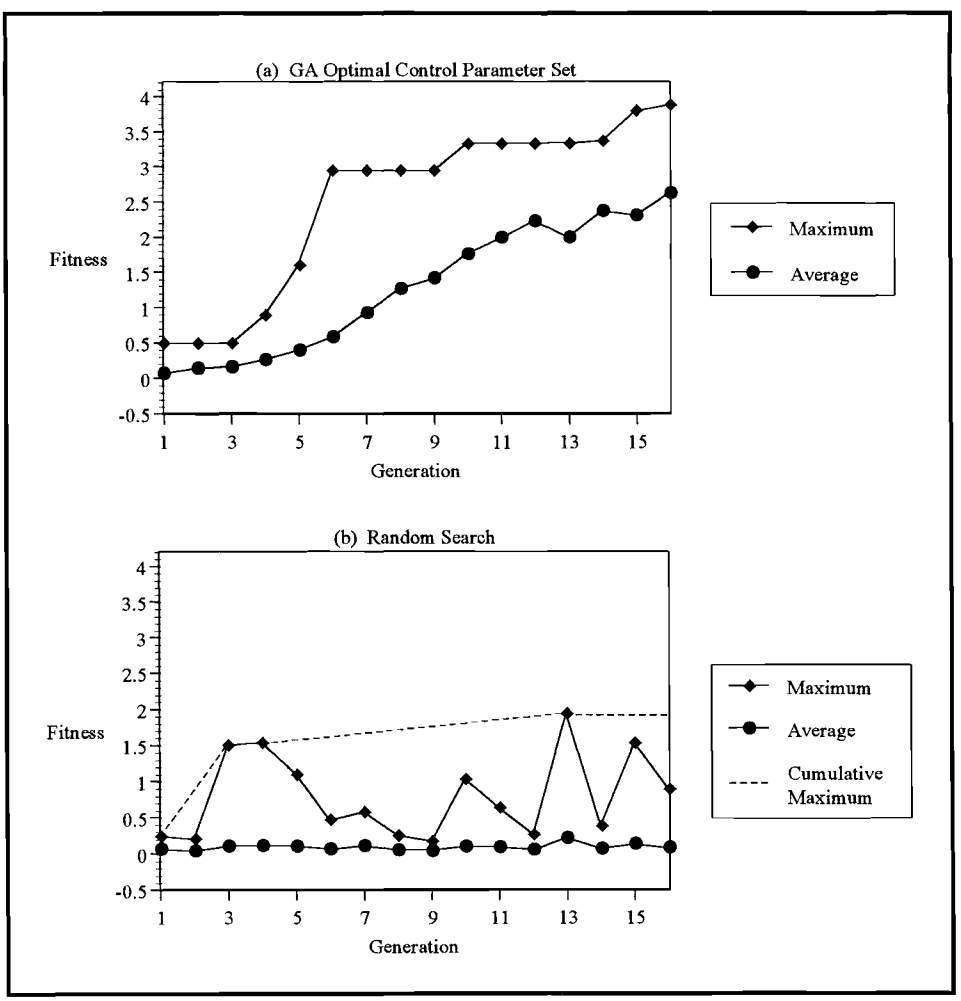


Figure 2.7. Maximum and Average Population Fitness (Complex Solution Landscape)

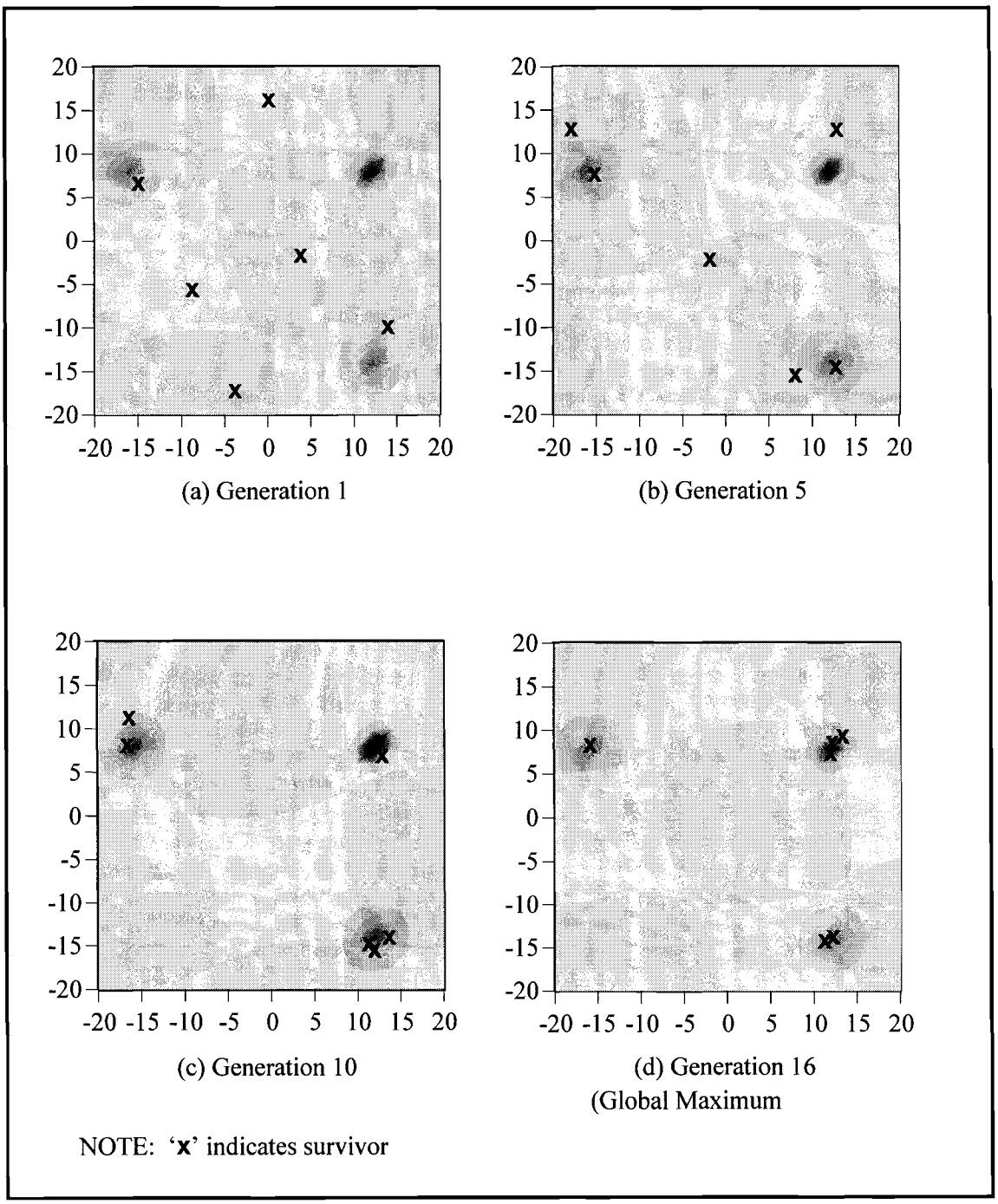


Figure 2.8. Evolution of Genetic Algorithm Solutions
(Complex Solution Landscape)

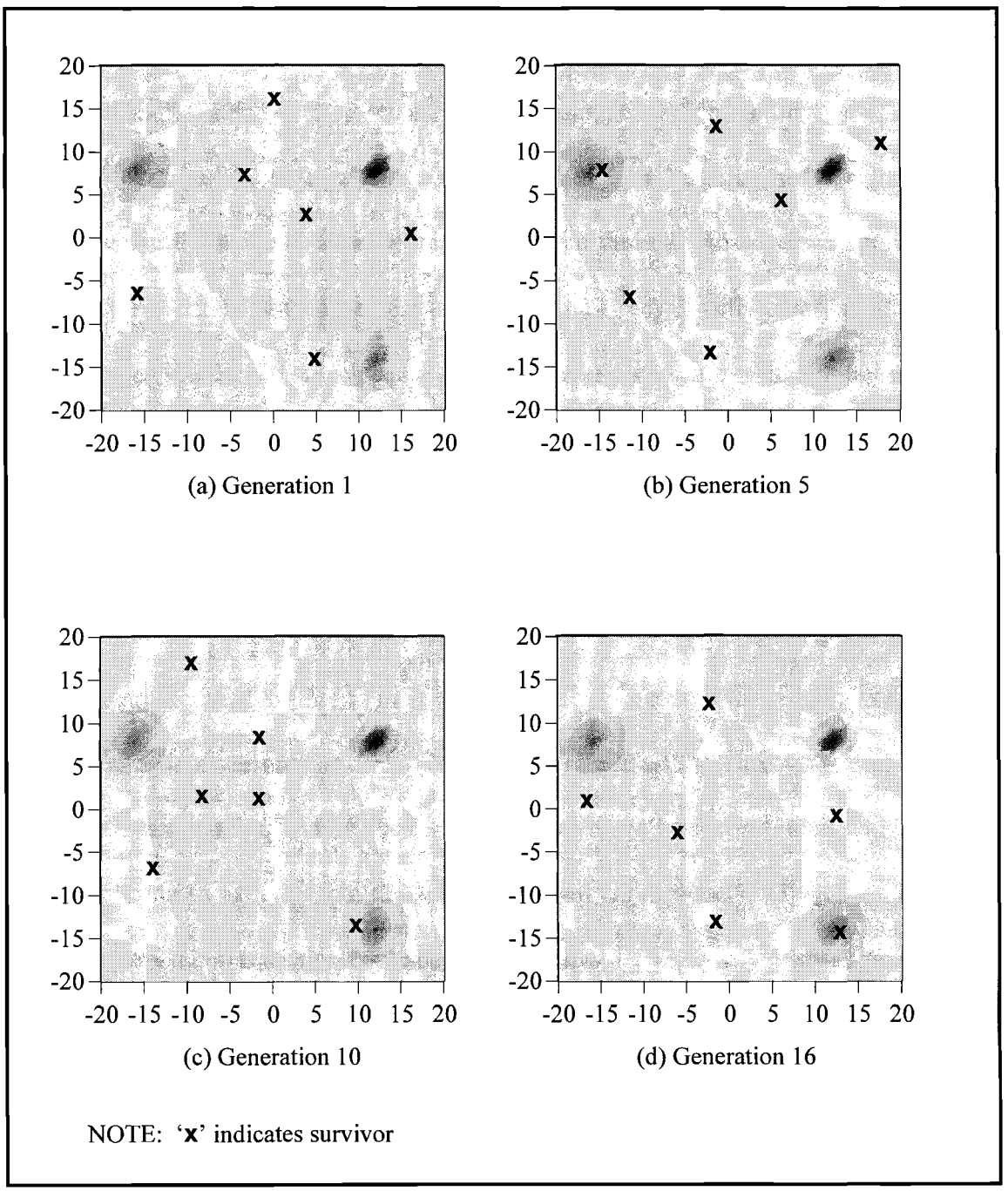


Figure 2.9. Evolution of Random Search Solutions
(Complex Solution Landscape)

2.4. Conclusions

The data gathered in this phase of research demonstrated that the genetic algorithm was highly effective in searching two-dimensional solution landscapes. The results indicated that two factors were crucial to the efficiency of the algorithm: a highly diverse gene pool and the recombination of genetic material through crossover. Population size alone was less critical, except insofar as larger populations directly contributed to diversity. In other words, a relatively small, highly diverse population yielded more promising results than a larger, homogeneous one. Finally, the genetic algorithm performed better than a purely random search, especially in the case of the more complex solution landscape.

All of the variables discussed in this section were supplied in an input file, so the user could tailor the algorithm to meet his/her particular needs. Furthermore, since fitness was evaluated in a subroutine within the main program, the GA computer code was highly modular. Mathematical and/or physical constraints could be incorporated within the fitness subroutine so as to restrict the solution domain appropriately. Therefore, the algorithm developed in this section could be applied to a wide variety of optimization problems simply by substituting a suitable fitness solver.

CHAPTER 3

AERODYNAMIC FITNESS SOLVER

The very nature of a genetic algorithm requires that fitness be evaluated thousands of times, so computational speed of the fitness solver was a major consideration. In the case of airfoils, numerous methods exist for evaluating the flowfield about an airfoil. Navier-Stokes solvers, which account for the effects of viscosity and compressibility, are the most accurate but require the most computational effort. For inviscid, compressible fluids, Euler solvers offer an improvement in speed, but are still rather computer-intensive. In the case of inviscid, *incompressible* flow (potential flow), much faster panel methods are available, although their range of applicability is more limited than the more sophisticated techniques.

In most aerodynamic applications, the effects of viscosity are confined to a very thin boundary layer, beyond which the flow may be assumed to be inviscid. Provided that compressibility is not an issue (Mach number less than 0.3 or so), the outer region can be analyzed with a potential flow solver like a panel method. Subsequently, viscous effects are analyzed with boundary layer equations; the superposition of the two regions yields the overall solution. Although two regions must be solved separately in this approach, the computational effort required is far less than in either of the other two methods.

For this research effort, airfoil performance prediction was accomplished through a combination of a panel method potential flow solver and an integral method for the prediction of boundary layer performance. Specifically, the Smith-Hess panel method, a simple, fast, yet relatively accurate potential flow solver, was implemented to calculate the velocity and pressure distribution about an airfoil. Subsequently, the laminar and turbulent regions of the boundary layer were evaluated using Thwaites' and Head's methods, respectively. This particular solution methodology, henceforth referred to as the AERO-SOLVER, was incorporated as a subroutine within the main GA computer program.

3.1. Inviscid Solution

The Smith-Hess panel method models the airfoil as a number of singularities (constant-strength sources and vortices) distributed on straight-line panels. Panel endpoints are distributed using cosine spacing so as to ensure more dense paneling near the leading and trailing edges. Each panel has an outward unit normal and a collocation point (panel midpoint), as indicated in Figure 3.1.

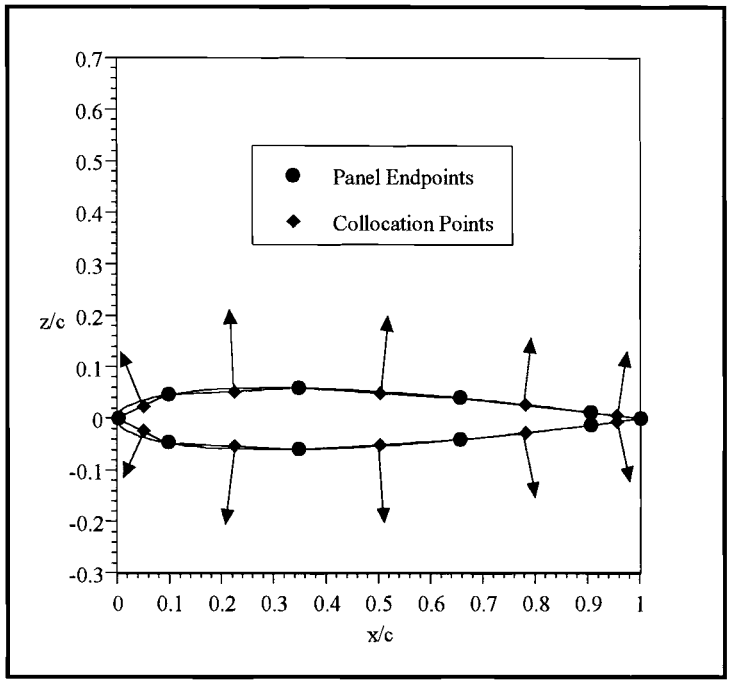


Figure 3.1. Airfoil Discretization Method

The velocity potential of each singularity element, Φ_i, is combined with the freestream potential, Φ_∞, to yield the total velocity potential, Φ^*. In algebraic terms:

$$\Phi^* = \Phi_\infty + \Sigma \Phi_i$$

The continuity equation for this incompressible, irrotational flow, in the airfoil's frame of reference, is given by Laplace's equation:

$$\nabla^2 \Phi^* = 0$$

The solution, however, is not unique without the imposition of boundary conditions. Physical constraints require that two conditions be satisfied. First, flow tangency at all collocation points must be enforced to prevent flow through solid boundaries. Second, the total amount of circulation introduced into the field must be sufficient to produce a stagnation point at the trailing edge. These two conditions are satisfied, respectively, by the following Neumann boundary conditions:

$$(\nabla \Phi^*) \cdot \mathbf{n}_i = 0$$

where $\mathbf{n}_i$ represents the outward unit normal of each panel, and

$$\gamma_{T.E.} = 0$$

The continuity equation, subject to the boundary conditions, can then be solved to obtain a unique solution for Φ^*. The x and y velocities at each collocation point are determined by taking the gradient of the potential, i.e.:

$$(v_x, v_y) = \nabla \Phi^*$$

Once the velocities are known, the pressure coefficients are obtained from:

$$c_{p,i} = 1 - \frac{V_i^2}{V_\infty^2}$$

where $V_i = \sqrt{v_x^2 + v_y^2}$

and V_∞ = freestream velocity

Finally, the pressure coefficients are numerically integrated over the length of the chord to obtain the lift and moment of the airfoil. As mentioned previously, the inviscid solution obtained in this manner is surprisingly accurate for thin bodies at low α. However, as thickness and/or α increases, the accuracy of the inviscid solution begins to deteriorate. In these more general cases, the effects of viscosity must be considered to obtain physically meaningful results.

3.2. Viscous effects

The inviscid solution method described in the previous section is predicated on one crucial assumption; namely, that viscous effects are confined to a very thin boundary layer adjacent to the solid surface. However, in the case of non-streamlined bodies where boundary layer separation is likely to occur, this assumption breaks down since the boundary layer is no longer vanishingly small compared to the characteristic length of the flow (usually taken to be the airfoil chord length, c). Viscous effects begin to exert an influence

not only on drag, but on lift and moment as well, and the potential solution becomes totally inadequate. In order to account for the effects of viscosity, the boundary layer was divided into three regions of interest: the laminar region, transition, and the fully turbulent region.

3.2.1. Laminar Region

Thwaites' method was used to calculate the growth of the boundary layer from the stagnation point to the point of transition. This method, which relies on Von Karman's momentum integral equation, makes no specific assumption on the form of the velocity profile and was therefore admirably suited to evaluation of the boundary layer on various airfoil shapes. Starting from the stagnation point, θ, λ, H, and c_f were calculated up to the point of transition. A detailed explanation of Thwaites' method may be found in Moran [1984] or White [1991].

3.2.2. Transition to Turbulence

Boundary layer transition is a complex phenomenon which is not yet completely understood. Transition is strongly affected by numerous parameters including freestream pressure gradient, compressibility, temperature, surface roughness, etc. [Kuethe & Chow, 1976]. Studies by Schlichting, Michel, Granville, Jaffee et al., and Wazzan et al. all sought

to predict the effect of pressure gradient on transition [White, 1991]. Generally speaking, these methods involve curve-fitting formulas to experimental data obtained from smooth shapes, and yield satisfactory results in the regimes for which they were developed. However, they are quite susceptible to relatively minor scattering in the data, and the actual position of transition may vary considerably from case to case [Bradshaw, Cebeci, & Whitelaw, 1981, and Moran, 1984]. Moreover, these methods were applied to experimental data, not to data obtained from an inviscid flow solver. Even minor inaccuracies in the potential flow solution could therefore lead to significant errors in calculated transition point. Finally, these studies considered only the effect of pressure gradient; other factors such as temperature and surface roughness were not included.

In addition, it is imprecise to speak of a transition 'point'. Transition is a phenomenon which occurs over a region, the length of which may exceed the total length of the laminar boundary layer [Cebeci & Smith, 1974]. Unfortunately, there is no comprehensive theory for calculating performance in the transition region.

Considering all of the aforementioned difficulties in accurately determining the transition point, it was decided to fix transition at the point of maximum velocity. This assumption certainly seemed reasonable considering the fact that the adverse pressure gradient aft of V_{max} substantially decreases the stability of the laminar boundary layer. The literature supports the validity of this approach [Schlichting, 1968, Hughes, 1979, and Moran, 1984], and in many cases this judiciously imposed transition point works at least as well as more complex prediction methods with virtually no computational effort [Kai, 1995].

This approach contained one potentially fatal flaw: it assumed that transition to turbulent flow would *always* occur at the V_{max} point and made no allowance for possible laminar separation bubbles. This shortcoming was circumvented by restricting analysis to Reynolds numbers greater than 200,000 and realizing that turbulators could be employed to ensure transition in wind tunnel or flight tests.

3.2.3. Turbulent Region

Integral techniques are superior to single parameter differential methods (such as Stratford's method) when separation prediction is based upon an inviscid solution as opposed to experimental data. Inviscid solutions characteristically exhibit an unrealistically sharp leading-edge pressure spike, the chordwise extent of which is normally confined to a very small region on the order of a few percent of the chord, as illustrated in Figure 3.2.

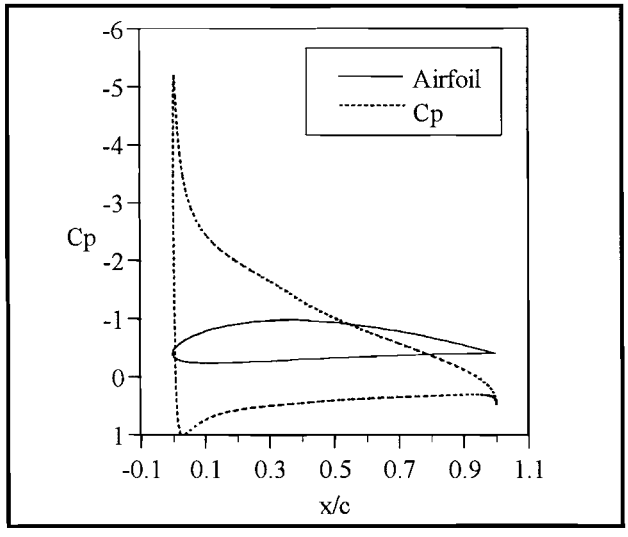


Figure 3.2. Leading Edge Pressure Spike
(NACA 4412 Airfoil, $\alpha = 10^{o}$)

While this spike has a limited effect upon the overall (integrated) pressure distribution, it is characterized by a very high pressure *gradient*. Integral methods are largely unaffected by the spike since they tend to wash out its effects, but single parameter differential methods are rendered ineffective by the large gradients. Since all of the data used in this analysis was obtained from a potential flow solver, an integral method was deemed the more appropriate of the two.

Starting at the V_{max} transition point, Head's integral method was employed to evaluate the turbulent boundary layer. This method includes two first-order ordinary differential equations, so two initial conditions were required. Although Thwaites' method gave values for θ, H, and c_f up to the start of transition, only the initial turbulent value of θ could be taken from the laminar calculations since H and c_f change so radically during transition [Moran, 1984].

However, for many applications a judiciously chosen value of H_{tr} can yield fairly reliable results. Kai's research, which examined the correlation between shape factor and separation point, indicates that a starting guess of 1.5 yields results which agree quite well with experimental data for a variety of airfoils and over a broad range of Reynolds numbers [1995]. Therefore, H_{tr} was set at 1.5 for the purposes of the current study. This approach is more reasonable than it may seem at first glance, since the shape factor is relatively constant during the initial portions of the turbulent boundary layer, yet increases very rapidly near the separation point. Provided that a reasonable value for H_{tr} is selected, the downstream calculations soon forget about the initial guess. Therefore, an error in the assumed value of H_{tr} leads to only minor deviations in the predicted separation point.

3.3. Corrected Solution

The purpose of the boundary layer analysis performed in the preceding section was, in short, to estimate the effects of viscosity on lift and moment. Drag was not a consideration. Once the separation point had been determined, the potential flow solution was adjusted using two rather simple relations [Eppler, 1990]. Specifically, c_l was corrected by:

$$\Delta c_l = \pi \Delta\alpha; \qquad \Delta\alpha = -s_{sep}(\delta_{us} + \alpha_c)$$

where s_{sep} = arc length along the airfoil surface from the separation point to the trailing edge

δ_{us} = slope of the upper airfoil surface near the trailing edge

α_c = angle of attack relative to the mean chord line

The moment coefficient about the quarter chord, c_m, was adjusted accordingly:

$$\Delta c_m = -\tfrac{1}{4}\Delta c_l (1 - s_{sep})^{1.5}$$

Using these adjustments, which were derived from Helmholtz theory, the potential flow solution agrees quite well with experimental data [Kai, 1995].

3.4. Panel Number Sensitivity Analysis

Obviously, the accuracy of the panel method is dependent upon the number of panels (N) used to discretize the airfoil. Since panels endpoints were connected by straight-line segments, the true airfoil shape was more closely approximated as N increased. Furthermore, boundary layer parameters were numerically evaluated only at discrete intervals (collocation points); the accuracy of these evaluations would certainly be enhanced by more refined panel spacing. On the other hand, computational effort was proportional to N^2, so a compromise between accuracy and speed was necessary.

Figure 3.3 indicates that c_l and c_m were relatively constant as long as 150 or more panels were used. Although slight variations in both parameters were noted for higher N, the marginal improvement in accuracy was deemed insufficient to warrant the greater computational effort required. Therefore, 150 panels were used for the remainder of this study.

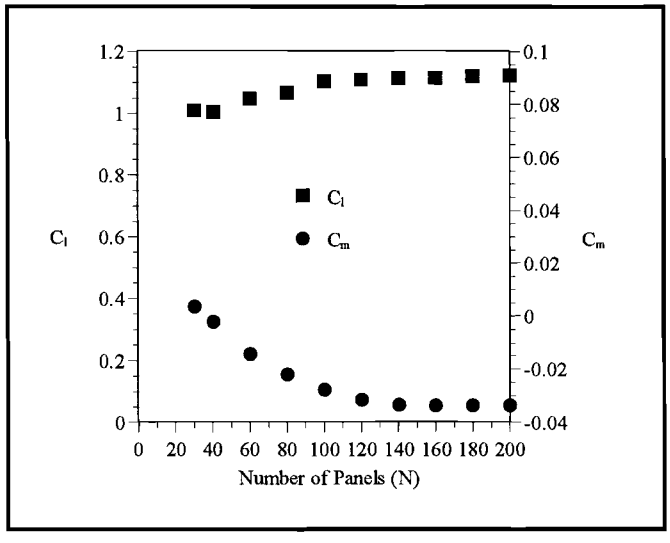


Figure 3.3. Panel Number Sensitivity for Liebeck LA2566 Airfoil
($\alpha = 10^o$, Re = 250,000)

3.5. Summary

The sole function of the aerodynamic fitness solver described in this chapter was to estimate the lift and moment coefficients of a given airfoil. It was implemented as a sub-routine within the main computer program and evaluated a single airfoil at a time. All other GA tasks (mutation, crossover, survivor selection, etc.) were accomplished in the main program, which required encoded representations of the airfoil population to carry out said tasks. Chapter 4 will discuss how the transformation between airfoil shape and genetic material was accomplished, and the means by which the genetic algorithm was implemented in the airfoil optimization problem.

CHAPTER 4

RESULTS

In order to test the genetic algorithm in an airfoil application, the GA was tasked with maximizing the lift coefficient (fitness) of an airfoil, subject to certain constraints which will be addressed in Section 4.2.5. Before incorporating the AEROSOLVER into an optimization scheme, however, a proper representation of airfoil shape was required. Two different approaches were considered. In the first, each collocation point of the airfoil was cast as a design variable. Section 4.1. will present some of the difficulties encountered in this approach. Much better results were obtained in the second formulation, discussed in Section 4.2, in which airfoils were represented using linear combinations of thickness and camber distributions.

4.1. Collocation Points as Genes

Initially, the genetic algorithm was incorporated by defining the genes as the individual collocation points of the airfoil. From one generation to the next, each point was free to mutate up or down independently of its neighbors. Crossover was accomplished

by combining the top surface of one surviving airfoil with the bottom surface of another. This method represented the most general approach since the airfoil could, theoretically, evolve into any arbitrary shape and was not restricted to a particular family of airfoils.

However, several factors hampered the efficacy of this approach. First, since each point could mutate independently of its neighbors, the algorithm produced airfoils with jagged, irregular surfaces. The AEROSOLVER was ill-suited for predicting separation along these uneven surfaces. As discussed in Section 3.2, most boundary layer methods were developed by curve-fitting formulas to experimental data obtained from airfoils with smooth, continuous surfaces and are not universally applicable to arbitrary shapes. Furthermore, integral methods such as Head's tend to integrate through the pressure curve discontinuities, further limiting accuracy of the results. For instance, in the case of the irregular shape depicted in Figure 4.1, the boundary layer would almost certainly separate upon encountering one of the surface discontinuities, yet the AEROSOLVER predicted separation at 0.92c. Furthermore, even if such a design did produce exceptionally high lift, it would be wholly impractical from the standpoint of drag and structural difficulties.

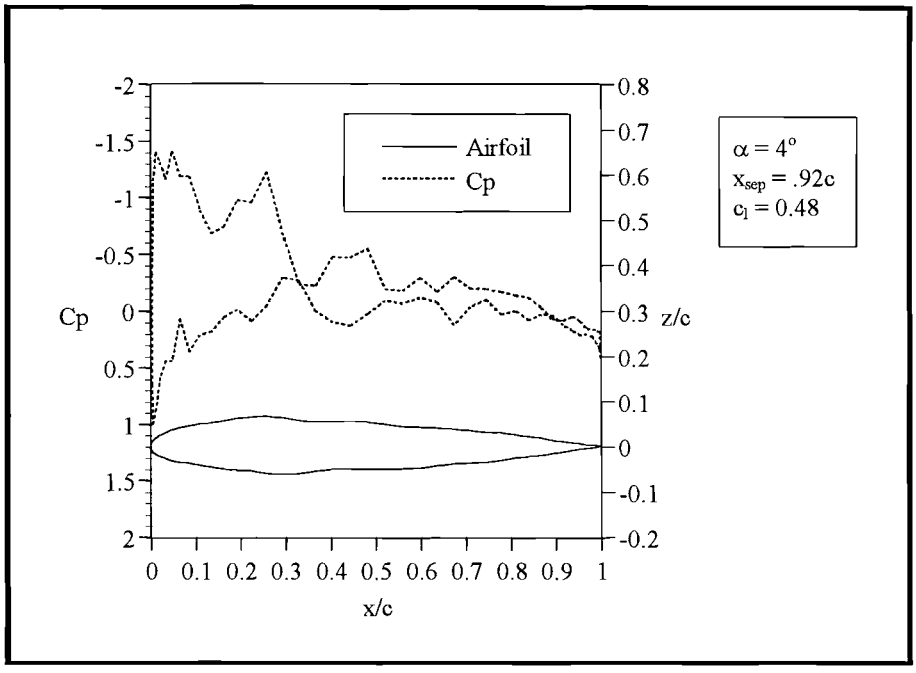


Figure 4.1. 'Collocation Points as Genes' Sample Airfoil

Another limitation of this approach arose from the GA itself. By defining 150 design variables (collocation points) the system was extremely epistatic and ill-suited to optimization by GA, as discussed in Section 1.2.7.1. The interaction between genes was so complex that it would be impossible to isolate, even qualitatively, the effect of any individual gene. In order to produce a viable airfoil shape, most or all of the genes had to simultaneously mutate in a specific fashion; even one or two unfavorable mutations could destroy an otherwise excellent design. In theory, provided that a more accurate AEROSOLVER was available, the algorithm should work even in this inefficient approach, but the computational time required would be prohibitive. In order to circumvent these limitations, an alternate formulation of the design variables was required.

4.2. Superposition of Basis Shapes

A much more successful approach was inspired by a synthesis of optimization schemes utilized by Vanderplaats [1984] and later by Quaglirella & Della Cioppa [1994], as discussed in Sections 1.2.3 and 1.2.6, respectively. In this approach, various airfoils were created through linear superposition of basis functions, thereby ensuring smooth, continuous shapes for which the AEROSOLVER was much more accurate. Moreover, the number of design variables was easily controlled by limiting the number of basis shapes considered.

For the purposes of this study, the basis shapes consisted of thickness and camber distributions. For small perturbations, the effects of thickness and camber are somewhat independent; therefore, a qualitative relationship between fitness and the individual genes could be established and the epistaticity of the design problem was reduced significantly.

Any airfoil can be uniquely represented by a combination of thickness and camber functions. In the present application, thickness, η_t, was defined as the distance between the upper and lower surfaces, while the camber function, η_c, was a mathematical representation of the locus of points halfway between the two airfoil surfaces. Given an analytical or numerical formulation of the upper and lower airfoil surfaces (η_u and η_l, respectively), thickness and camber are given by:

$$\eta_t = \eta_u - \eta_l$$

$$\eta_c = \frac{1}{2}\left(\eta_u + \eta_l\right)$$

Given *n* basis airfoils, different shapes were obtained through linear superposition of *2n* basis shapes (*n* thickness functions and *n* camber functions), i.e.:

$$\eta_{t\,GA} = a_1\eta_{t1} + a_2\eta_{t2} + \ldots + a_n\eta_{tn}$$

$$\eta_{c\,GA} = b_1\eta_{c1} + b_2\eta_{c2} + \ldots + b_n\eta_{cn}$$

where a_i and b_i represented the genes of the genetic algorithm.

Once the a_i and b_i values were specified, the resulting combination of η_{tGA} and η_{cGA} was sufficient to uniquely define the GA airfoil.

4.2.1. Selection of Basis Airfoils

The four basis airfoils selected for this research were the Selig S1223, Wortmann FX63-137, Liebeck LA2566, and NACA 4412. The first two airfoils were specifically developed for high lift at low Reynolds numbers while the Liebeck LA2566 was designed to provide maximum lift in unseparated flow. Although not particularly a high-lift airfoil, the NACA was added in the interest of diversification since it has thickness and camber distributions which are noticeably different from those of the others.

In addition, two qualitatively different design philosophies were manifested in the four basis shapes. The first two airfoils have impressive $c_{l,max}$ values of 2.18 and 1.60, respectively, but their aft loading results in undesirably large nose-down pitching moments.

(All c_m values presented in this report refer to moments about the airfoil quarter chord, 0.25c). In order to counteract the large airfoil moment, any conventionally configured aircraft using the Wortmann or Selig wing would require a significant downward force on the horizontal stabilizer, reducing the total effective lift considerably [Nelson, 1989]. In contrast, the Liebeck and NACA airfoils generate most of their lift over the leading half of the chord and produce much lower pitching moments. While their $c_{l,max}$ values are somewhat lower (1.5) than their high lift counterparts, the reduced pitching moments are beneficial when considering overall aircraft performance. Figure 4.2 depicts the c_p curves for the four basis airfoils, illustrating the different qualitative nature of their pressure distributions.

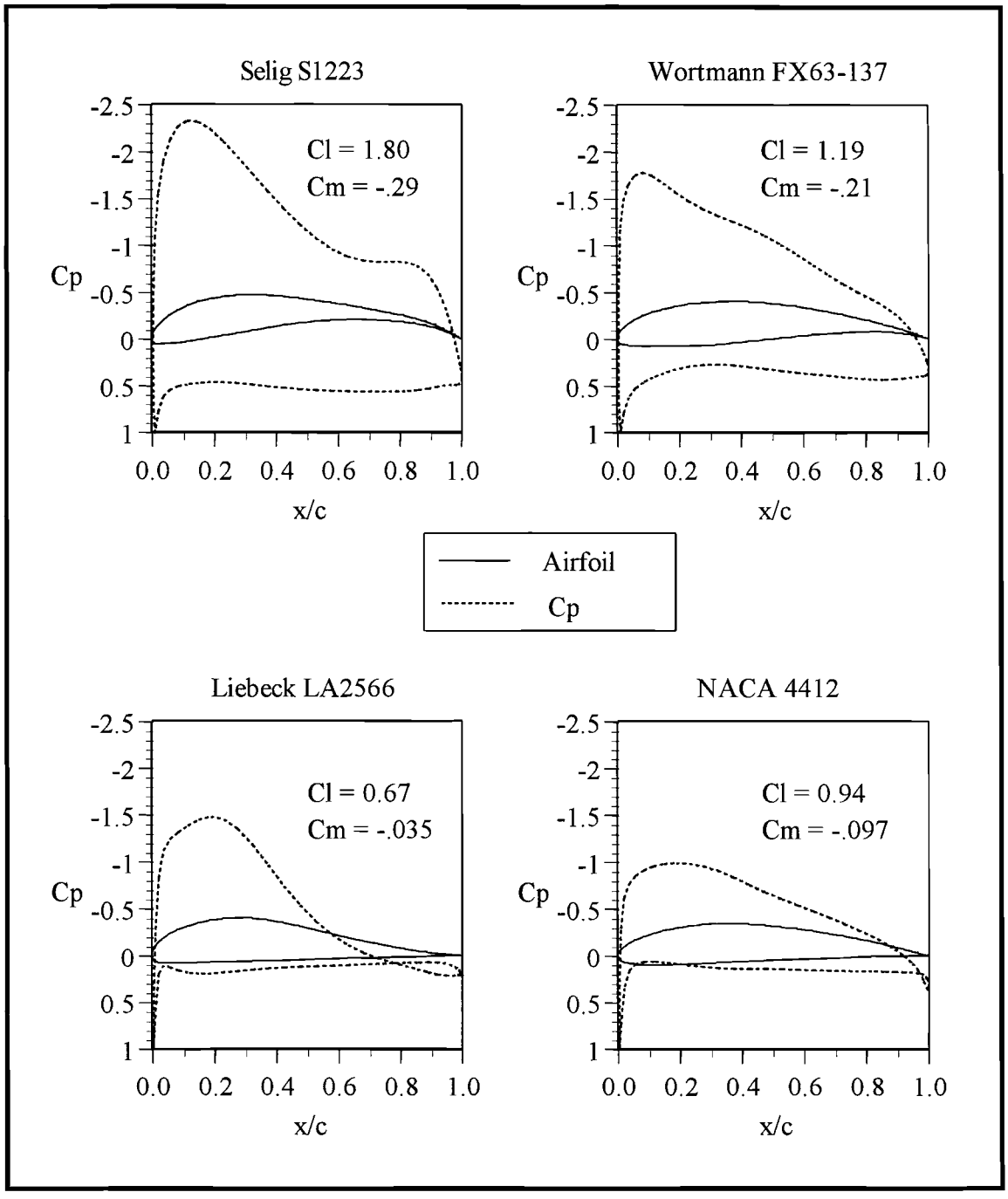


Figure 4.2. C_p Distributions of Basis Airfoils
($\alpha = 4^o$, Re = 250,000)

4.2.2. Mathematical Representations of Basis Functions

The thickness and camber functions for the Selig, Wortmann, and Liebeck airfoils were obtained by curve-fitting formulas to airfoil data. Curve-fitting was not required for the NACA 4412, since, like all NACA 4-digit series airfoils, it was *defined* by mathematical thickness and camber distributions. Unfortunately, the other three did not lend themselves particularly well to curve fits; although correlation coefficients of 0.996 or higher were noted, an exact fit was simply not possible. While the use of additional terms in the curve fit formula did yield higher correlation coefficients, a point of diminishing returns was soon reached since the higher order curves had a greater number of inflection points. In other words, a higher order curve fit correlated better *mathematically*, but physically it didn't reflect the true shape of the airfoil, and the accuracy of the predicted performance solution deteriorated correspondingly.

This limitation was most critical near the leading and trailing edges. Minor changes in geometry near the edges had a much greater effect on overall pressure distribution than similar changes in the mid-chord region. Furthermore, in order to ensure a closed body, the leading and trailing edges had to have zero thickness. At all other points, positive thickness was required not only physically, but mathematically as well. In the extreme case of zero thickness, the superposition of the top and bottom singularities resulted in infinite velocities. Near the leading and trailing edges, even minor inaccuracies in the curve fit, while seemingly insignificant in an absolute sense, often translated into large *relative* errors. Therefore, minor modifications in η_t and η_c were made in order to im-

prove the fit near the edges, even though the accuracy of the fit was degraded somewhat in the mid-chord region.

4.2.3. Validation of Aerodynamic Fitness Solver

In order to evaluate the accuracy of the curve-fit approach in conjunction with the AEROSOLVER, c_l and c_m calculations were conducted for each basis airfoil through a range of α. As indicated in Figures 4.3 through 4.6, lift and moment predictions agreed quite well with existing data for low angles of attack. At higher α, the predicted performance of the Wortmann, Selig, and NACA airfoils was qualitatively valid, although some error was noted. Considering the fact that elements such as surface roughness, freestream turbulence, and the like were not taken into consideration, the algorithm was considered to be sufficiently accurate for these three airfoils.

However, the algorithm failed to predict stall for the Liebeck airfoil until very high angles of attack (32°, not shown in figure), indicating a more severe limitation in the accuracy of the curve fit, the flow solver, or a combination of the two. Liebeck airfoils were designed to produce a Stratford type pressure recovery aft of the V_{max} point. Such a recovery is characterized by a flow in which the boundary layer remains attached along the entire upper surface while avoiding separation by a constant specified margin; consequently, it is very susceptible to minor deviations in the shape of the c_p curve [Liebeck, 1978].

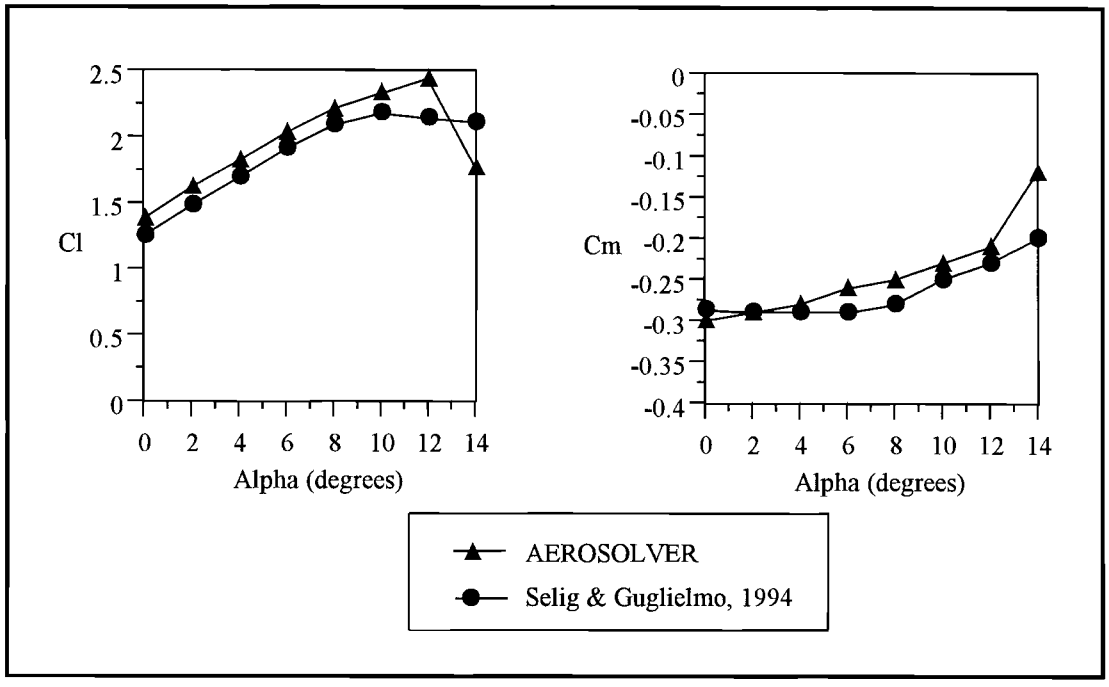


Figure 4.3. c_l, c_m Curves for Selig S1223 Airfoil
(Re = 200,000)

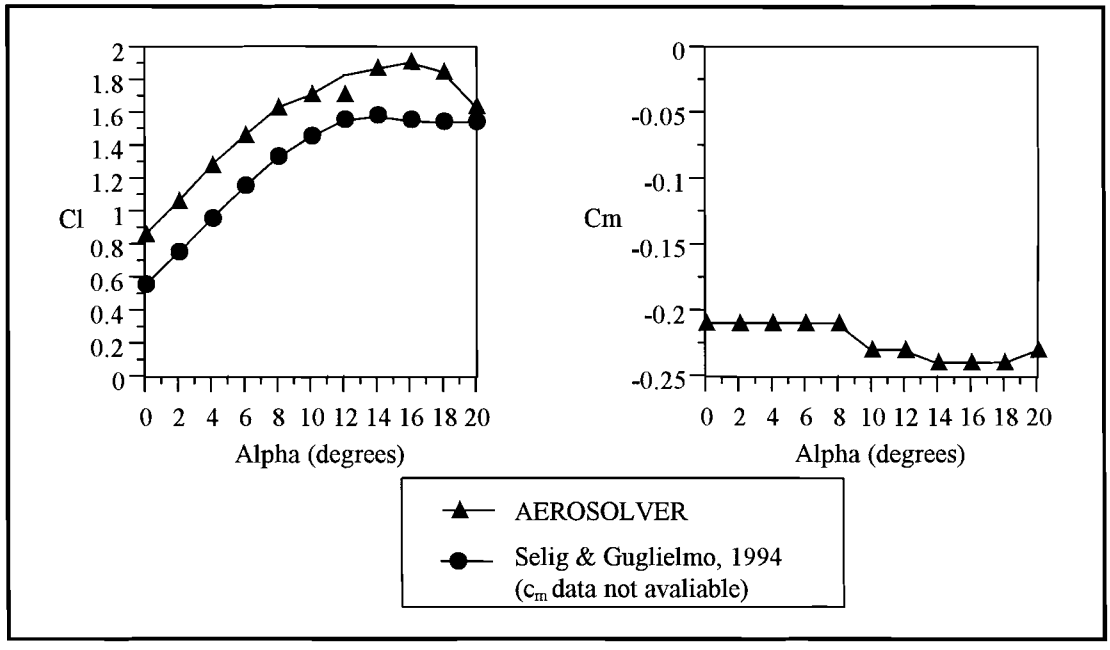


Figure 4.4. c_l, c_m Curves for Wortmann FX63-137 Airfoil
(Re = 200,000)

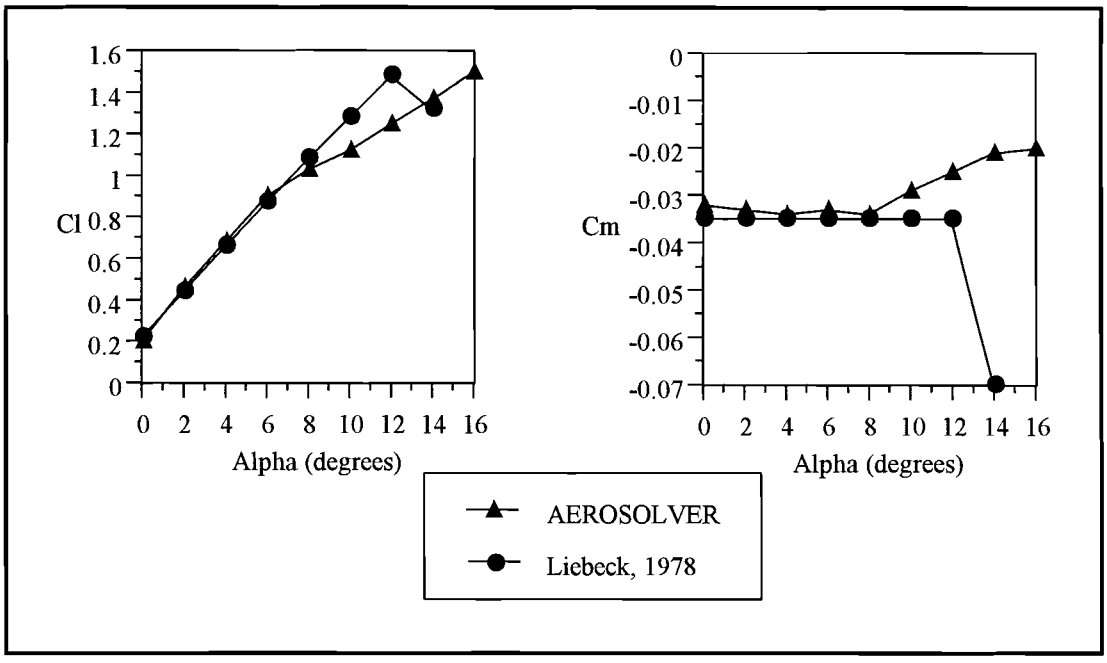


Figure 4.6. c_l, c_m Curves for Liebeck LA2566 Airfoil
(Re = 250,000)

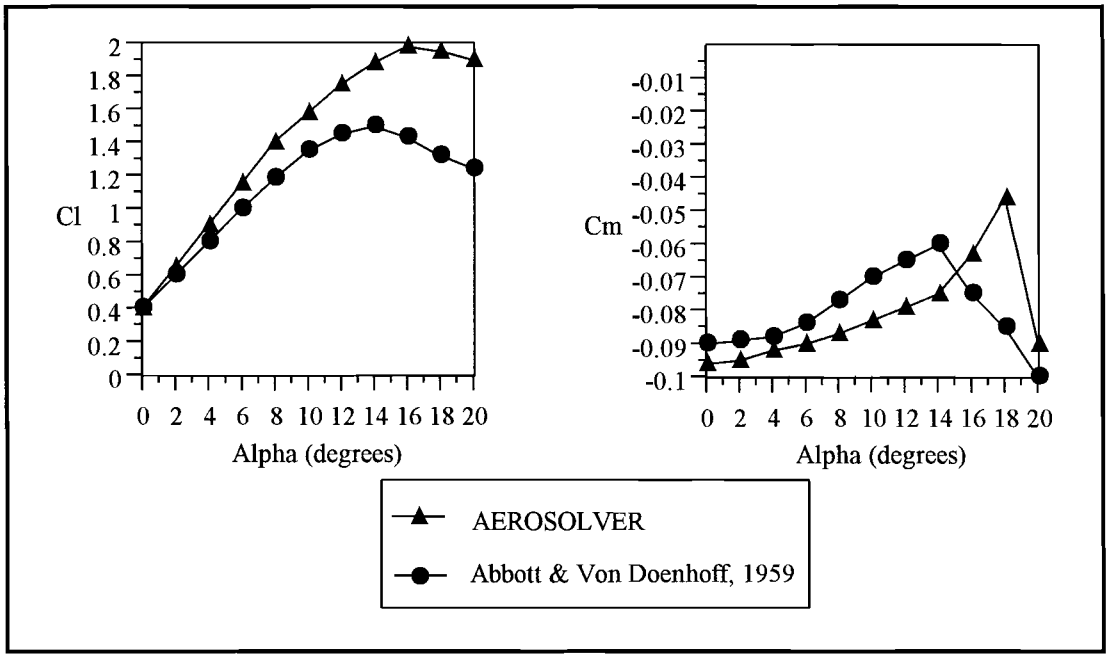


Figure 4.5. c_l, c_m Curves for NACA 4412 Airfoil
(Re = 3,000,000)

Additionally, the c_p curve of the Liebeck has a comparatively long, flat section in the vicinity of the maximum velocity point. This region is characterized by very small velocity gradients, $\partial V_i/\partial x$, so a minor error in the inviscid velocity distribution could lead to appreciable variations in the calculated V_{max} point. Consequently, boundary layer transition, which was assumed to occur at V_{max}, could have been significantly in error as well, thereby impacting the entire boundary layer analysis and performance prediction of the airfoil.

In contrast, the maximum velocity point for the other three airfoils occurs in a region of relatively high velocity gradients. Therefore, an inaccuracy in velocity distribution for these airfoils would yield a smaller error in V_{max} point prediction, and have less impact on c_l estimation.

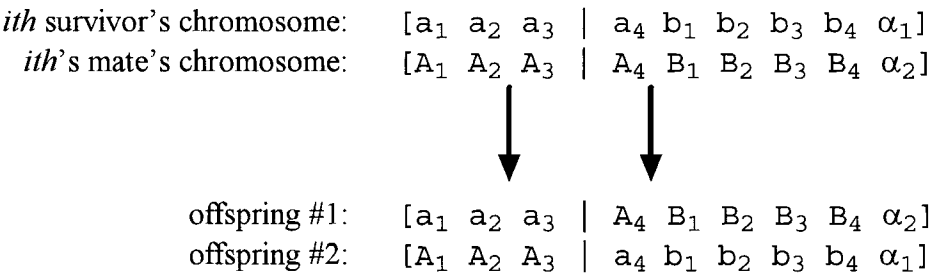
Kai [1995] examined the precision of various separation prediction methods when applied to both experimental data and potential flow solutions. Using identical methodology as in this study, the AEROSOLVER was extremely accurate in predicting separation based upon *experimental* data, yet was significantly degraded when applied to potential flow data. Furthermore, he didn't employ any curve-fitting approximations whatsoever; he used arrays of collocation points directly to define various airfoil shapes. Based upon these results, it was concluded that the primary limitation of the AEROSOLVER arose *not* from curve-fit inaccuracies, but from the difficulties involved with predicting transition based upon potential flow data. This limitation in the flow solver was partially circumvented by restricting the solution domain to a space in which the AEROSOLVER was more accurate. The specific constraints incorporated will be presented in Section 4.2.5.

4.2.4. Genetic Algorithm Implementation

With four basis airfoils selected, the algorithm had eight basis functions with which to create new airfoil shapes. The weighting coefficients, a_i and b_i, represented the eight genes of the algorithm. At this point, however, the entire analysis would have been restricted to one particular angle of attack. While the results might have been interesting, the goal of high lift airfoils is to maximize $c_{l,max}$, *not* to maximize c_l at any specific angle of attack. Therefore, a ninth gene, representing α, was incorporated. On the other hand, if design at a particular α was desired, the user could tailor the input file accordingly simply by specifying α and setting its maximum mutation size to zero.

The chromosome of each airfoil was defined as a nine-element string of genes, i.e., $[a_1\ a_2\ a_3\ a_4\ b_1\ b_2\ b_3\ b_4\ \alpha]$. Given a population of chromosomes, each one was converted into a thickness distribution, a camber function, and an angle of attack. Thereafter, fitness was evaluated and survivors selected in accordance with the rank-space method discussed in Chapter 2. A new population was created, as before, through a combination of mutation and crossover, with one variation. In the two-gene case, the chromosome was split in the middle during crossover, while in the nine-gene case, the crossover point was selected at random. By allowing a random crossover point (rather than a fixed, central one), the algorithm was able to recombine genetic material more freely, possibly resulting in gene combinations which would have been more difficult to locate otherwise. The chromosome

of the *ith* survivor was cut at the crossover point and reattached to the chromosome of the *ith* survivor's mate, as indicated below ('|' indicates the crossover point):



Initially, the optimal control parameter set of Section 2.3 was adopted for the airfoil application. Preliminary results indicated, however, that several modifications were required for this more complex, nine-dimensional solution domain:

1. MAXMUT: In the two-dimensional case, maximum mutation size was set at 1.0, representing 5% of the total allowable range of values for each gene (± 10). During initial tests in the airfoil application, a_i and b_i values fell into the nominal range of -0.5 to +1.5, so MAXMUT was reduced to 0.1, similarly restricting the maximum mutation per gene to 5% of the 'range' of values allowed. The genes were *not* restricted per se; the nominal range was only utilized to determine an appropriate value for MAXMUT.

2. n_{niche}: When only six total niches were used, as in the two-dimensional case, the survivors appeared to inadequately populate the solution domain and seemed sus-

ceptible to local maxima. Unfortunately, the nine-dimensional solution domain was impossible to visualize, so this characteristic could not be determined directly. Instead, this conclusion was reached by observing the 'final' survivors at the end of different evolutionary test runs, where each test run consisted of several hundred generations. The final survivors of one test run often represented only one or two qualitatively different designs, corresponding to one or two different evolutionary niches. A subsequent test run, however, might produce designs which were very similar *to each other*, but markedly different from the final survivors of the first test run. In essence, it was deduced that various niches were widely separated, in a chromosomal sense, and the population was insufficiently diverse to locate and exploit these distant niches. In order to overcome this shortcoming, n_{niche} was increased to 49, which, in addition to the primary survivor, gave a total of 50 survivors per generation and a population size of 200 individuals.

Other than these modifications, all of the control parameters discussed in Chapter 2 were implemented in the same way as for the two-dimensional case. While the algorithm itself appeared to perform effectively, it became apparent that several *physical* constraints were necessary to ensure meaningful results.

4.2.5. Constraints

4.2.5.1. Angle of Attack

Ideally, no restrictions on angle of attack would have been required. If the AEROSOLVER had been more accurate in predicting stall, especially for the Liebeck airfoil, excessive α values would have been penalized by a lower fitness value. More sophisticated flow solvers would have been more accurate at high α, but would have required prohibitively long computational times. For example, the AEROSOLVER used in this study required about one minute to evaluate a complete generation of 200 airfoils. Conservatively estimating that a Navier-Stokes' solver could evaluate a *single* airfoil per hour, it would take over a week to complete the calculations for a single generation. Since evolutionary test runs often consisted of hundreds or thousands of generations, computer time for a Navier-Stokes' solver would have been measured in months or years. Therefore, a potential flow solver was deemed the only practical solution method in spite of its limitations at high α. In order to compensate for this deficiency, α was limited to a maximum of 10°, thereby restricting the solution domain to a regime in which the flow solver was comparatively accurate.

4.2.5.2. Thickness

Drag and weight of the airfoil were not specifically addressed in the problem formulation. Interestingly, the algorithm managed to exploit this limitation by producing extremely thick airfoils, up to maximum thickness values of 50% or more. It is true that enormously thick airfoils can produce impressive lift coefficients, but the associated drag and weight render them utterly impractical for aerodynamic applications. Consequently, future airfoils were limited to a maximum thickness no greater than 20% of the chord.

On the other extreme, the evolution towards unrealistically thin airfoils revealed a limitation of the panel method itself. Since the thickness weighting coefficients, a_i, were not restricted in any way, the superposition of thickness functions could easily produce a (theoretical) airfoil with zero or negligible thickness. When this occurred, the near superposition of top and bottom collocation points produced erroneously high velocities which had a major effect upon lift and moment predictions. Similarly, inverted or 'negative thickness' airfoils, which resulted when $\Sigma a_i < 0$, were wholly unrealistic from a practical standpoint. Obviously, a constraint on minimum thickness was required to weed out mathematically superb designs which were physically unattainable.

At the point of maximum thickness, airfoil thickness must be at least 5% (0.05c) to ensure adequate structural strength. Near the trailing edge, however, where nominal thickness values on the order of 0.0001 were common, the 'thinness' restriction had to be significantly relaxed to allow the airfoil to taper off to the trailing edge point. It was therefore impossible to enforce a *single* fixed minimum thickness value along the entire

chord length. Instead, a variable restriction, based upon the thickness distribution of the Selig airfoil, was implemented.

Past experience has indicated that the trailing edge of a 12" Selig is so thin as to pose manufacturing difficulties [Arena, 1995]. This problem could be partially overcome by using a larger scale, but eventually a point would be reached where the trailing edge would become structurally unsound. However, detailed structural analysis was deemed beyond the scope of the current study, especially since this entire project was performed using non-dimensional airfoils. Therefore, it seemed reasonable to limit the minimum thickness at any collocation point to one-half the thickness of the Selig airfoil at the corresponding point.

4.2.5.3. Leading Edge Pressure Spikes

The leading edge pressure spike had an impact upon the inviscid solution (prior to the incorporation of viscous effects) which was not immediately apparent. Since the inviscid lift coefficient was determined by integrating the c_p curve, the large pressure spike actually increased the calculated $c_{l,u}$ value slightly. For each of the basis airfoils, the magnitude of these c_p values was no greater than 5.0, and since the spike extended only a very short chordwise distance, its effect upon overall lift was negligible. The GA, however, often produced airfoils with a much sharper leading edge, and these airfoils had utterly unrealistic c_p values of -100 or more. Although these c_p values were only present at one or two collocation points, the magnitude of these spikes was such that the overall lift

coefficient *was* substantially affected. Research by Valarezo and Chin indicates that, for low Re/Mach number combinations, the absolute limit on c_p is in the range of -5.0 to -7.0 [1994]. This physical limitation arises from the fact that local pressure can never be less than zero, regardless of the theoretical velocity distribution. Consequently, any airfoil shapes which produced pressure coefficients less than -6.0 at any collocation point were discarded from further consideration.

4.2.5.4. Pitching Moment Coefficient

The flow solver also demonstrated limited ability to predict the performance of extremely cambered airfoils since boundary layer calculations were conducted only for the upper surface. For common airfoil shapes at positive α, separation is normally associated with the top surface where the boundary layer must overcome adverse pressure gradients. On the other hand, the lower surface flow is almost always characterized by a favorable pressure gradient and is unlikely to separate. Consequently, most separation prediction methods deal primarily with the upper surface boundary layer, even though highly cambered airfoils may encounter separation along the lower surface as well.

In general, camber not only contributes to lift, but also has an adverse effect on pitching moment, as evidenced by the performance of the Wortmann and Selig airfoils. Considering the correlation between camber and pitching moment, it seemed reasonable to restrict c_m rather than camber directly. Constraining the pitching moment not only circumvented this limitation of the flow solver, but also encouraged evolution towards a re-

gime which was advantageous from a practical standpoint, as discussed in Section 4.2.1. Therefore, c_m was restricted to -0.1; airfoils whose nose-down pitching moments exceeded this limit were given a zero fitness value, effectively removing them from further consideration.

It should be emphasized that some subjectivity was involved in the selection of constraints, which is true for any constrained optimization problem. A different set of restrictions would certainly have led to the evolution of different airfoils. Nonetheless, the goal of this project was to demonstrate the GA technique itself, not necessarily to design an airfoil for a particular flight regime. In summary, the task of the genetic algorithm was to:

Maximize: c_l (fitness)

Subject to:

1. Angle of attack $\leq 10^o$
2. Maximum airfoil thickness $\leq$ 20% of chord
3. Minimum airfoil thickness $\geq$ 50% as thick as the Selig at corresponding collocation points
4. $c_{p,min} \geq -6.0$
5. $c_m \geq -0.1$

4.2.6. Seed Population

In the two-dimensional optimization problems of Chapter 2, the dimensions of the solution domain were known, i.e., each gene was constrained to a range of allowable values between fixed upper and lower boundaries. For example, in the simple solution landscape, the genes of each chromosome in the seed population were determined by randomly selecting values in the interval between ±10. With these mathematical constraints, it was simple to create a seed population which effectively covered the entire solution space.

In this airfoil application, however, the constraints were physical rather than mathematical in nature. In other words, the weighting factors of the basis shapes were not restricted *directly*; rather, the resulting airfoil was constrained by limitations in thickness, c_m, etc., which were applicable only to airfoils resulting from *combinations* of genes. It was impossible to determine realistic upper and lower bounds on the individual genes in advance, so there was no range from which to randomly select values for the seed population.

In many engineering applications, the designer begins with an existing design as a starting point. In order to evaluate GA sensitivity to variations in starting design, three different seed populations were considered. The use of different starting points also provided an opportunity to evaluate whether or not the search converged to a global maximum. A globally optimum solution (subject to the design constraints and the limitations of the flow solver) would be indicated if the algorithm converged to the same shape regardless of the initial design selection. On the other hand, if different starting points yielded

different 'optimal' airfoils, one would be forced to the conclusion that the algorithm had become stuck in a local maximum.

In the first test, the entire seed population consisted of mutated versions of the NACA 4412 airfoil (in addition to the unmutated 4412 itself). Similarly, the second cycle was initiated with the Selig airfoil as the starting point. These two airfoils were chosen since they had very different performance characteristics: the 4412 has the lowest lift and moment coefficients of the four basis shapes while the Selig airfoil has the highest.

Preliminary test runs indicated that all viable airfoils had genes which were nominally in the range of -0.5 to +1.5. This range was empirically established *only* after the completion of several evolutionary cycles and served solely as a guideline. Therefore, the seed population of the third test was created by randomly selecting genes for each individual within this range. Finally, for the sake of comparison, a purely random test was conducted in which the genes were randomly generated, again in the range of -0.5 to 1.5, for *each* generation, as described in Chapter 2.

4.2.7. Airfoil Evolution

Initial tests revealed that very little evolution occurred after 200 generations or so. In order to be reasonably assured of convergence without excessive computational effort, subsequent experiments were terminated after 300 generations. The results of three evolutionary cycles corresponding to the three seed populations, in addition to a purely random search, are indicated in Figure 4.7.

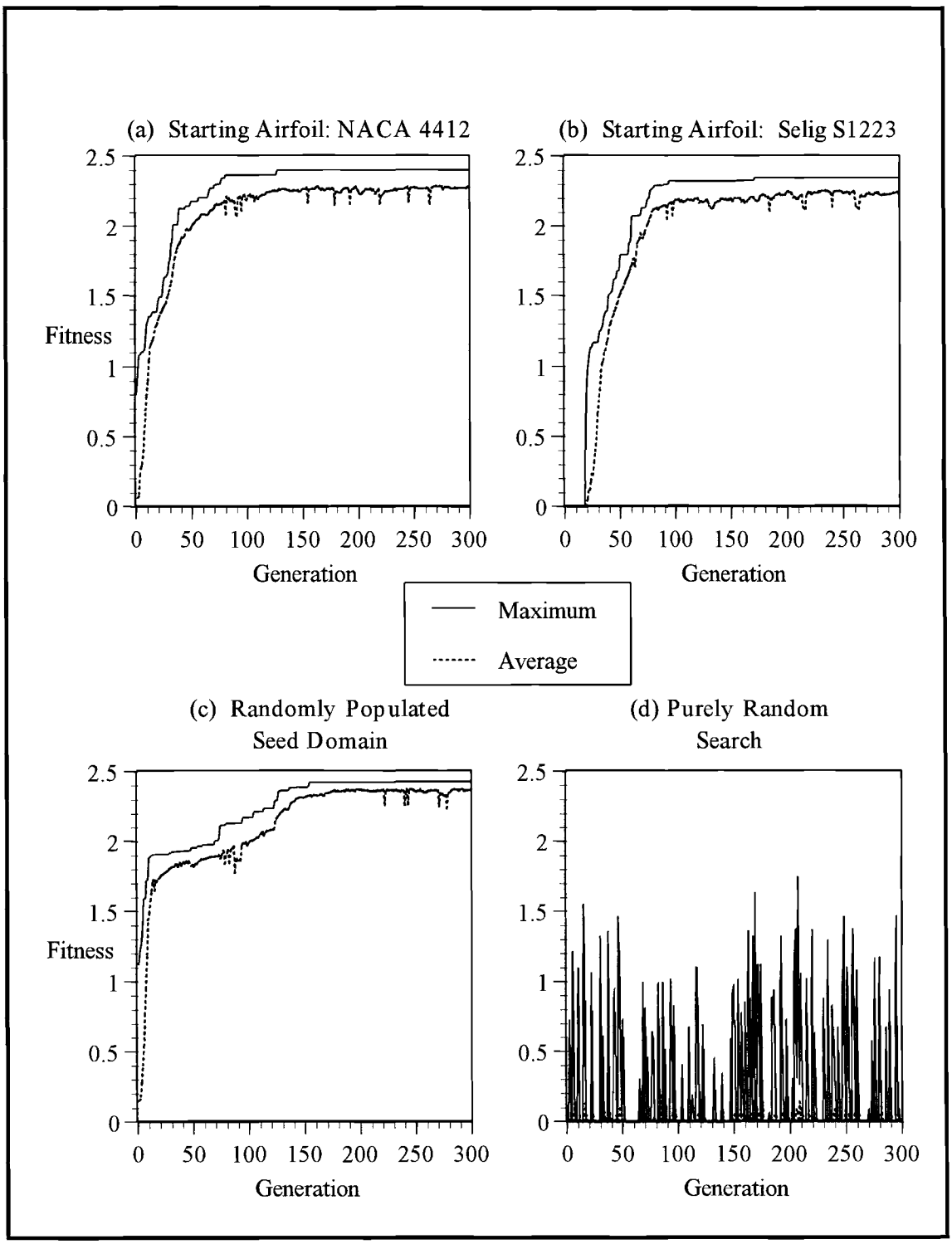


Figure 4.7. Maximum and Average Airfoil Population Fitness

While the random search did identify an airfoil with a $c_{l,max}$ of 1.61, the vast majority of the randomly-generated airfoils violated one or more of the physical constraints and had zero fitness. On the other hand, it was interesting to note that the fitness curves for all three GA tests exhibited the same qualitative behavior and terminated in the same maximum fitness (approximately 2.4). These results indicate that the algorithm was quite insensitive to the particular starting point selected, provided that the population was sufficiently large and diverse.

Several trends in the fitness curves are particularly noteworthy. The rapid rise in initial fitness levels was due, in part, to increasing angles of attack. Regardless of thickness or camber distributions, virtually all airfoils exhibit a linear relation between c_l and α at low angles of attack. During the initial phases of evolution, survivor selection was heavily biased towards those candidates with high α, while the other genes were relatively unimportant. This trend continued until α had stabilized near 10^o, corresponding roughly to c_l values of 1.5 to 1.6. Thereafter, angle of attack was essentially constant and subsequent evolution resulted from alterations in airfoil geometry.

Figure 4.7(b) indicates that the Selig airfoil required about 20 generations before *any* evolution took place. This apparent anomaly arose from the constraints incorporated in the optimization problem. As discussed previously, the aft-loaded Selig airfoil has a large pitching moment, on the order of -0.3, and any airfoil with $c_m < -0.1$ was automatically given a zero fitness value. Numerous generations were required to overcome this c_m barrier; however, once an individual with a suitable c_m was identified, evolution progressed normally.

Finally, each of the three GA cycles indicates a step-wise improvement in maximum and average population fitness values. The steps are smaller than in the two-dimensional case, presumably because the larger number of genes required several simultaneous mutations for an 'evolutionary breakthrough'.

Figures 4.8 to 4.10 illustrate the evolutionary process; each diagram corresponds to the primary survivor of the indicated generation. Although the airfoils in the beginning stages of evolution are markedly different from one another, a definite trend of increasing camber and forward loading is common to all three. Regardless of the starting point, each cycle converged to virtually the same design, as depicted in diagram (f) of each figure. Because of this trend, it certainly appeared that this design represented the globally optimum solution to this particular optimization problem, although impossible to prove mathematically.

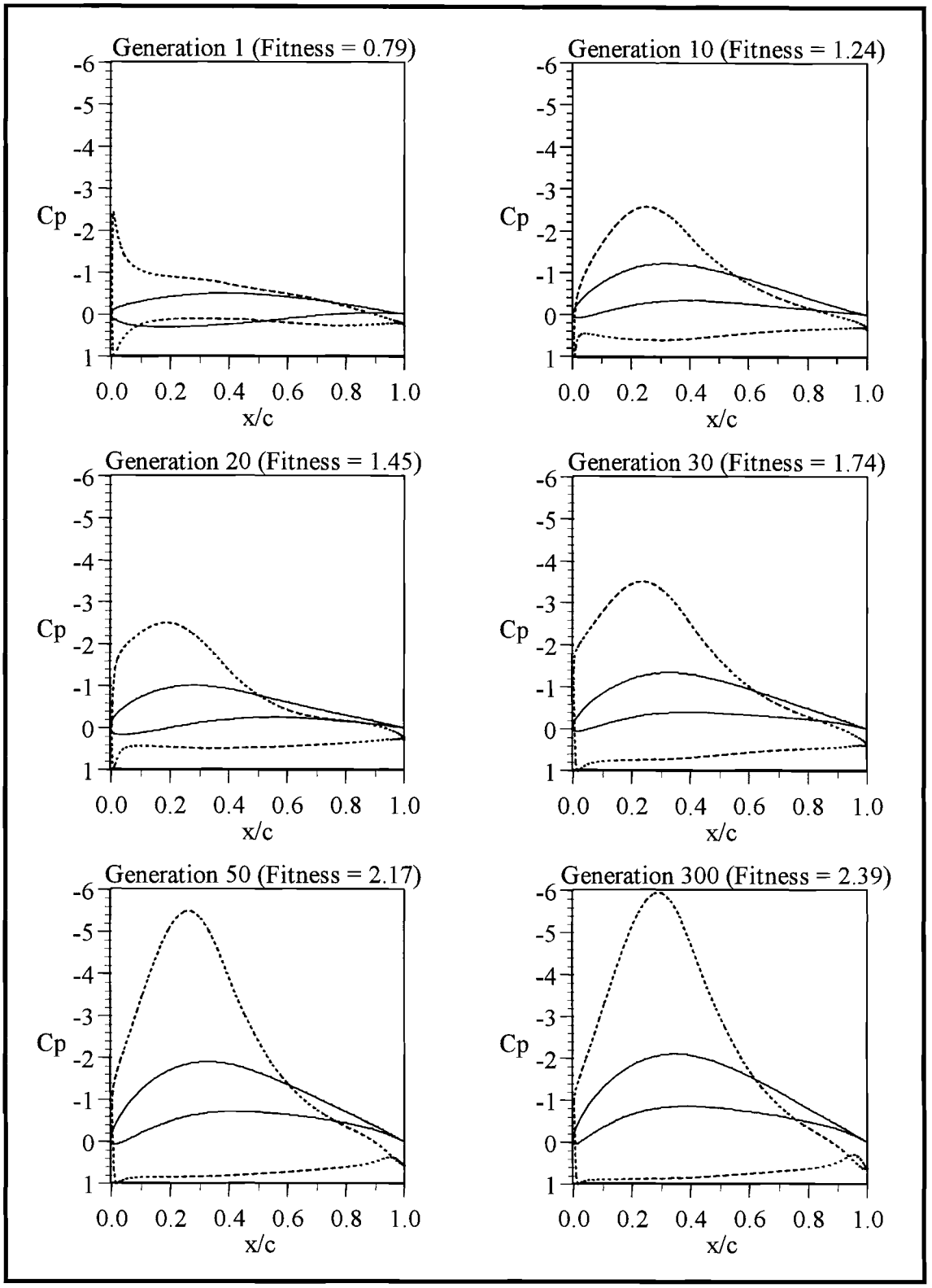


Figure 4.8. Evolution of Airfoils
(Starting Airfoil: NACA 4412)

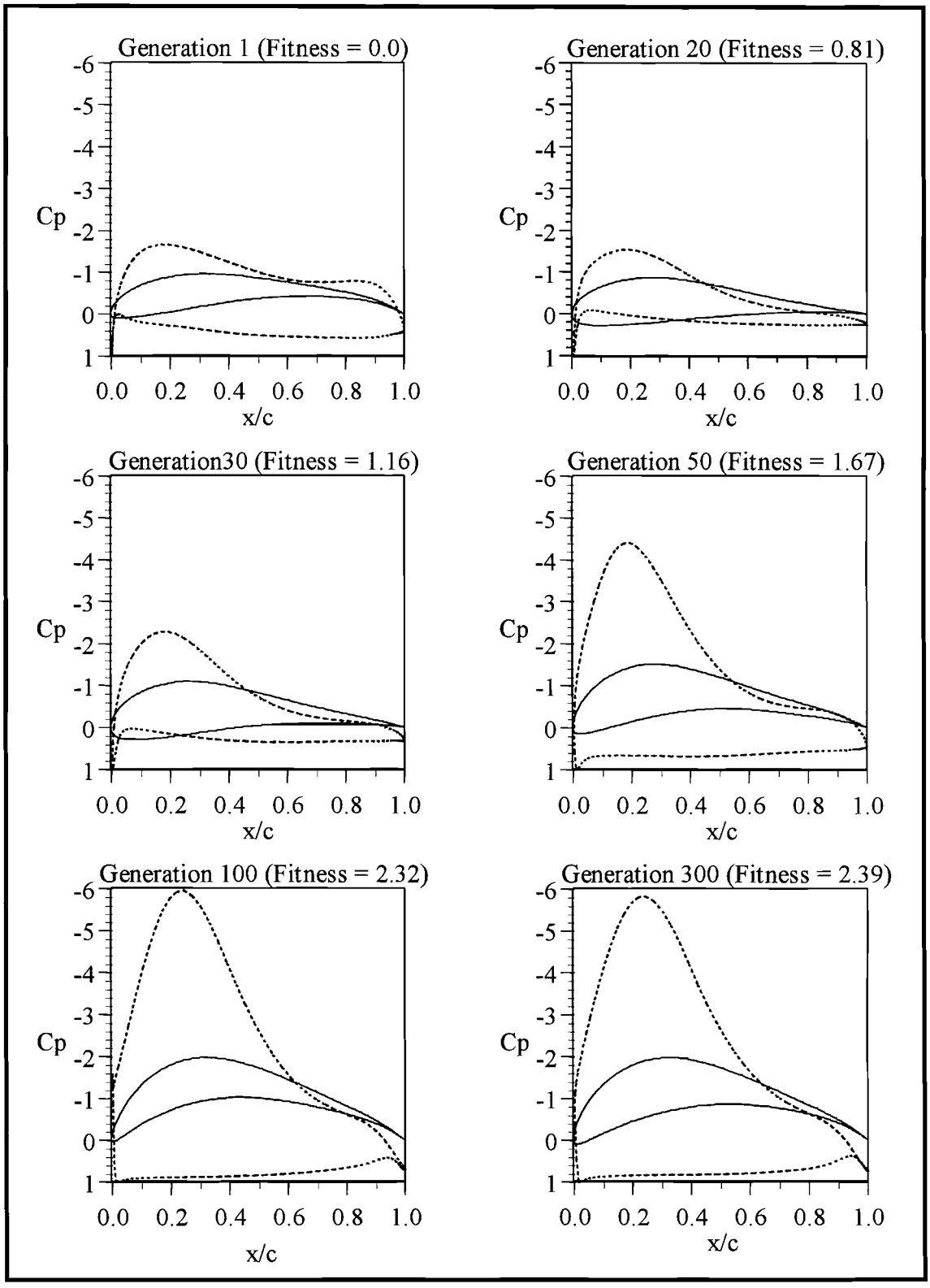


Figure 4.9. Evolution of Airfoils
(Starting Airfoil: Selig S1223)

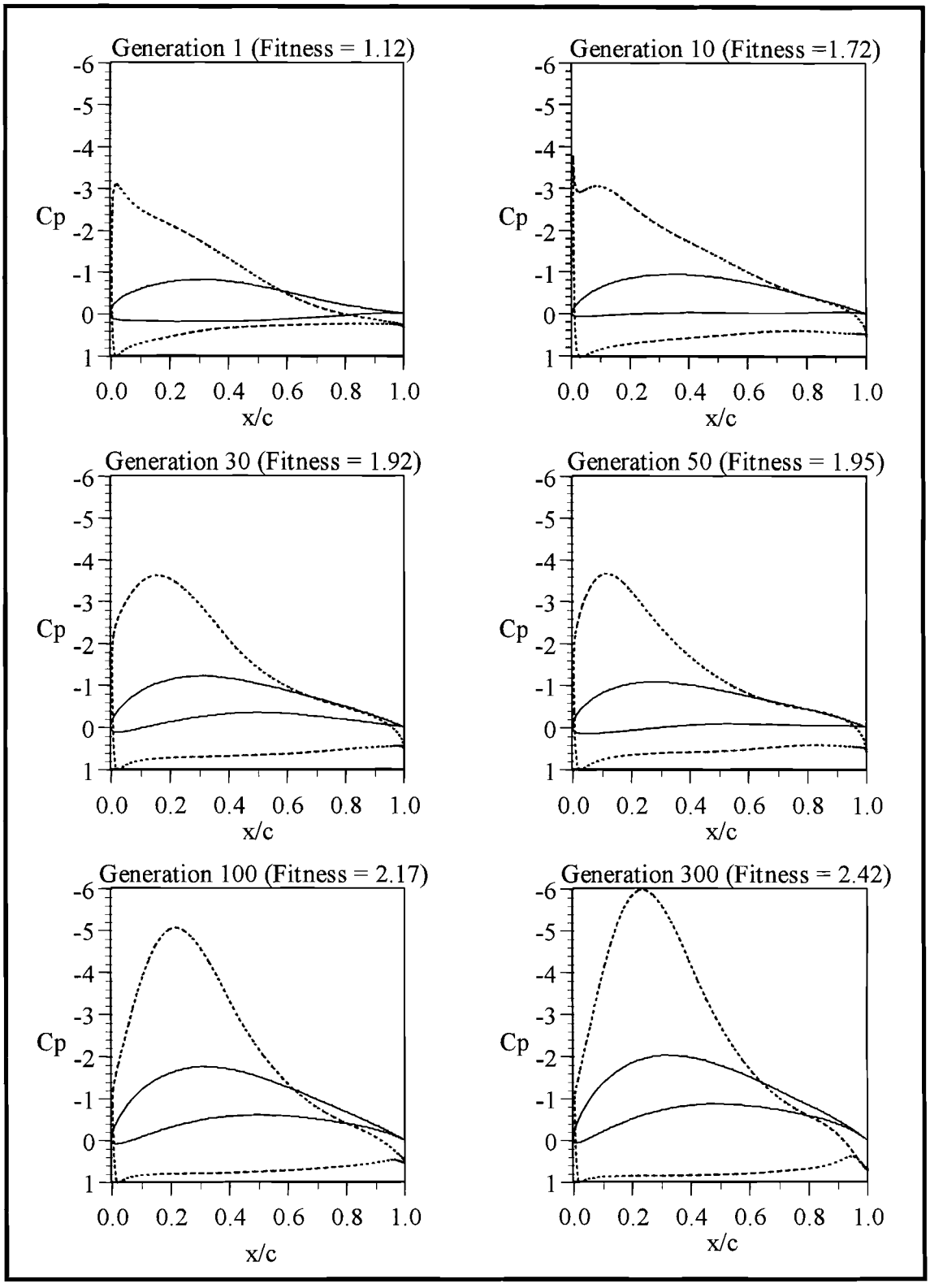


Figure 4.10. Evolution of Airfoils
(Starting With Randomly Populated Seed Domain)

4.2.8. Optimal Airfoil Performance

The GA optimal airfoil was a highly cambered design with a forward-loaded pressure distribution and an estimated boundary layer separation point located at 0.64c. Figure 4.11 shows the geometry and pressure distribution of the GA optimal airfoil along with the four basis airfoils for the sake of comparison. The figure illustrates that the optimal pressure distribution is one in which the V_{max} point is shifted aft significantly, delaying boundary layer transition and separation.

The performance of the five airfoils is summarized in Table 4.1. These results indicate that the genetic algorithm successfully identified a design with significantly higher fitness than any of the basis airfoils. Essentially, the GA optimal design combined the favorable characteristics of each of the basis airfoils: the high c_l of the Selig and Wortmann airfoils with the low c_m of the Liebeck and NACA.

Airfoil	c_l	c_m	Fitness
Selig S1223	2.33	-0.28	0.0 (c_m constraint violated)
Wortmann FX63-137	1.67	-0.26	0.0 (c_m constraint violated)
Liebeck LA2566	1.08	-0.08	1.08
NACA 4412	1.63	-0.09	1.63
GA Optimal	2.42	-0.09	2.42

Table 4.1. Comparison Between Basis Airfoils and GA Optimal Airfoil ($\alpha = 10^o$, Re = 250,000)

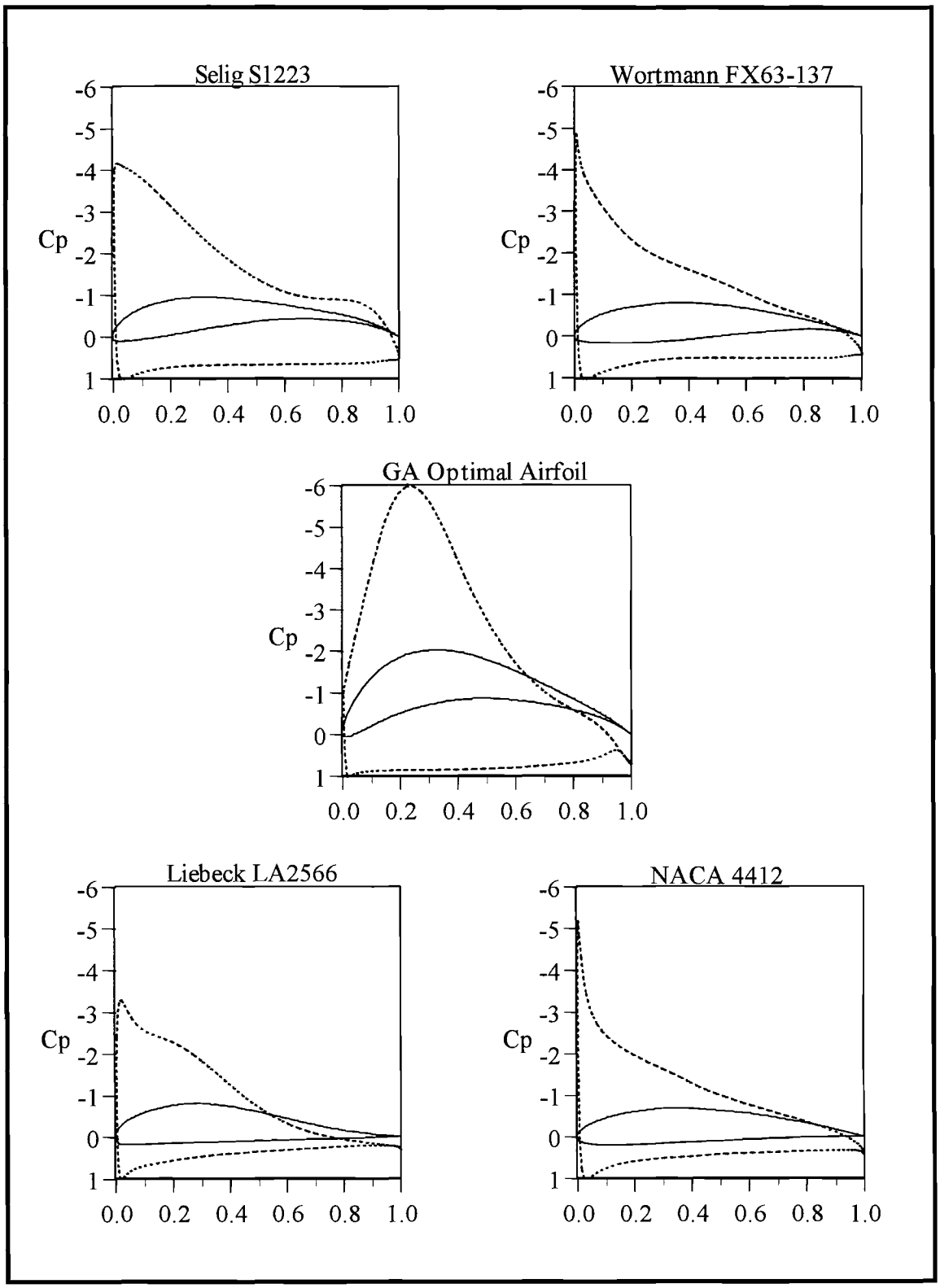


Figure 4.11. Airfoil Geometry and Pressure Distributions for Basis Airfoils and GA Optimal Airfoil ($\alpha = 10^{o}$, Re = 250,000)

The lift and moment performance of the GA design was evaluated at angles of attack from 0° to 16°. The $c_{l\alpha}$ curve in Figure 4.12 illustrates a linear relation between c_l and α up to the stall angle of attack (approximately 12°). As α increases beyond stall, c_l drops off precipitously, corresponding to a sudden forward jump in the boundary layer separation point.

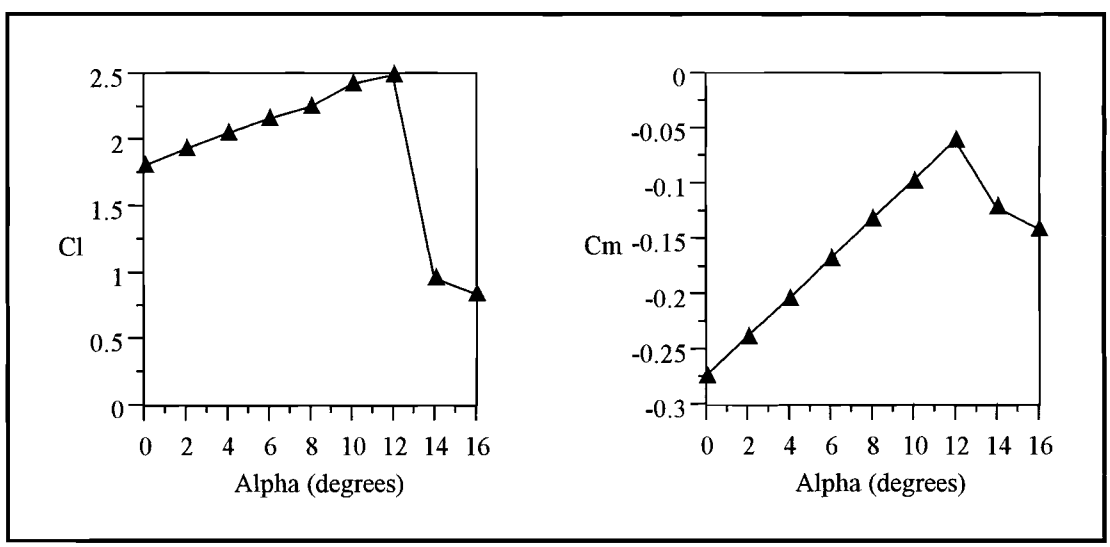


Figure 4.12. c_l, c_m Curves for GA Optimal Airfoil
(Re = 250,000)

The $c_{m\alpha}$ curve also depicts a linear relation between c_m and α below stall angle of attack. Interestingly, the GA design has a zero fitness value for any α outside the narrow range of 9.8° to 10°. Below 9.8°, the c_m restriction is violated, while above 10° the angle of attack limit is exceeded. Of course, this does not indicate that the *physical* performance of the GA airfoil would undergo a drastic change as α passed through 9.8° or 10°; the sharp discontinuity in fitness is purely a result of the way the mathematical constraints were incorporated.

At low angles of attack, the c_m curve indicates that the GA airfoil has a rather large pitching moment (-0.275 @ 0°). While undesirable, this c_m is no greater than that of the Selig airfoil throughout its angle of attack range below stall α. At 10° α, the GA airfoil has slightly higher lift than the Selig, but with a substantially lower pitching moment. Since drag estimation was not conducted in this study, the drag characteristics of the GA optimal airfoil were not considered.

4.2.9. Limitations

The selection of basis airfoils, control parameter values, and, particularly, the design constraints, all had an impact on the genetic algorithm's performance. Moreover, there is no guarantee that this GA optimal design truly corresponds to the global maximum. As discussed in Section 1.2.7, one cannot prove that any GA-based design represents the mathematically optimum solution. However, considering the fact that the solution converged to the same design regardless of the starting point, one could reasonably infer that the final GA airfoil does, in fact, represent the best possible design *subject to the particular constraints incorporated and the limitations of the AEROSOLVER.*

The performance of the GA airfoil was not independently validated. Unquestionably, the inaccuracies of the AEROSOLVER affected the outcome of the genetic algorithm search. In the author's opinion, this was the most significant limitation of this project. It would have been highly desirable to evaluate the performance of the GA airfoil, either with a Navier-Stokes flow solver, or, ideally, through wind tunnel experiments.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1. Conclusions

The primary objective of this research project was to examine the viability of a genetic algorithm as a mechanism for optimizing airfoil design. Numerous practical considerations and potential pitfalls were identified. Specifically, an algorithm was developed to be used in conjunction with existing aerodynamic performance prediction methods. This project demonstrated that genetic algorithms are potentially useful tools in the airfoil design optimization process. The following conclusions were reached during this investigation.

1. Genetic algorithms are robust schemes for optimizing complex, multi-dimensional systems. The concept is rather intuitive and requires little mathematical analysis beyond the evaluation of the objective function which is to be maximized. However, extremely epistatic relations are ill-suited to optimization with this technique, as revealed by the 'collocation points as genes' approach. Better results are ob-

tained if the problem is reformulated to reduce the number of design variables and/or the complexity of interactions between them.

2. The optimal combination of control parameters can be quite problem-dependent. Mutations on the order of 5% to 10% of the range of allowable gene values seem to provide a satisfactory search; smaller mutations can be used to fine-tune a particular design while larger ones yield a more random search of the entire solution domain. Simple, smooth functions can be optimized using smaller populations and relatively few survivors, and the algorithm can easily 'hill-climb' local maxima through mutations alone. More complex functions may require larger populations, additional 'niche' survivors, and the presence of crossover. In general, algorithm performance is enhanced by a highly diverse gene pool and the recombination of genetic material through crossover.

3. It was proven that the algorithm was highly effective in locating global maxima of two-dimensional functions. Although impossible to prove mathematically, the technique seemed to find the global maximum in the airfoil application as well, as evidenced by the fact that the algorithm repeatedly converged to the same design regardless of the starting point used. It significantly outperformed a purely random search, especially for larger and/or more complex solution domains.

4. The GA code developed in this study was highly modular. By incorporating a suitable fitness solver subroutine and imposing appropriate mathematical or physical

constraints, the algorithm could be used to optimize a wide variety of design problems.

5. The genetic algorithm is strictly a search technique. It abides by the rules, so to speak, which are dictated by the fitness solver. Any results obtained from this technique are valid to the degree that the fitness solver models the physical system it represents. Since the very nature of the GA requires that fitness be evaluated repeatedly, the accuracy of the fitness solver must be weighed against computational time requirements.

6. As with most optimization schemes, a sound understanding of the engineering principles involved is essential if the results are to be physically realistic. Furthermore, the algorithm can cleverly exploit limitations of the fitness solver, as demonstrated in this study by the evolution towards unrealistically thin airfoils. Appropriate constraints may be required to ensure physically meaningful results and to restrict the search to the portion of the solution space in which the fitness solver is sufficiently accurate.

7. The major shortcoming of this study was the limited accuracy of the aerodynamic flow solver. Although the AEROSOLVER was qualitatively valid, as evidenced by the c_l and c_m curves of the basis airfoils, it was simply inadequate from a quantitative viewpoint. This limitation was most severe at higher angles of attack, unfortunately corresponding to the regime in which maximum lift occurs.

8. Regardless of the limitations of the flow solver, the genetic algorithm successfully identified an airfoil with significantly better performance than any of the basis airfoils. Its robustness was demonstrated by the fact that it converged to virtually the same solution regardless of the initial design. Any errors in estimated performance resulted from the AEROSOLVER, not the algorithm itself.

5.2. Recommendations

Ideally, a more accurate flow solver would have been used in this study, although full Navier-Stokes solvers are too computationally intensive to be feasible with current computer capabilities. Alternatively, greater precision might have been possible with a more accurate *potential* flow solver, especially if boundary layer analysis were conducted using a method specifically developed for use with an inviscid flow distribution.

Another possibility would have been to use the AEROSOLVER iteratively. The boundary layer displacement thickness could have been added to the airfoil contour, and a modified velocity distribution calculated by enforcing flow tangency at the edge of the boundary layer rather than at the airfoil surface. This methodology would require twice as much computational effort, yet still be enormously faster than a Navier-Stokes or Euler solver.

The use of additional constraints might have circumvented some of the difficulties encountered in this investigation. Specifically, the design problem could have included a

penalty for drag, or have been reformulated to maximize L/D ratio, thereby encouraging the evolution of more streamlined shapes for which the potential flow solution was more accurate.

Finally, experimental validation of the results would have been very beneficial. Any future research should include full CFD evaluation or wind tunnel data of the final design in order to validate the results obtained in the study.

BIBLIOGRAPHY

Abbott, I. M., & Von Doenhoff, A. E. (1959). Theory of Wing Sections. New York: Dover Publications, Inc.

Arena, A. S. (1995). Personal Communications.

Beale, E. M. L. (1988). Introduction to Optimization. New York: John Wiley & Sons, Ltd.

Bradshaw, P., Cebeci, T., & Whitelaw, J. H. (1981). Engineering Calculation Methods for Turbulent Flow. New York: Academic Press, Inc.

Cebeci, T., & Smith, A. M. O. (1974). Analysis of Turbulent Boundary Layers. New York: Academic Press, Inc.

Cheung, M. (1993). A Survey and Comparison of Conjugate Gradient Methods for Optimization. M. S. Thesis. Oklahoma State University.

Cheung, S., Aaronson, P., & Edwards, T. (1995). CFD Optimization of a Theoretical Minimum-Drag Body. Journal of Aircraft, 32(1), 193-198.

Dulikravich, G. (1992). Aerodynamic Shape Design and Optimization. Journal of Aircraft, 29(6), 1020-1026.

Dumitrache, I. (1995). Special Control Seminar Series on Intelligent Control. Oklahoma State University, 25 July 1995.

Ellis, T. M. R. (1982). A Structured Approach to Fortran 77 Programming. London: Addison-Wesley.

Eppler, R. (1990). Airfoil Design and Data. Berlin: Springer-Verlag.

Eppler, R., & Somers, D. M. (1980). A Computer Program for the Design and Analysis of Low-Speed Airfoils. Hampton, VA: National Aeronautics and Space Administration.

Filho, J. L. R., Treleaven, P. C., & Alippi, C. (1994). Genetic Algorithm Programming Environments. Computer, 27(6), 28-43.

Fisher, P. A. (1930). The Genetical Theory of Natural Selection. Oxford: Oxford University Press.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. New York: Addison-Wesley.

Hughes, W. F. (1979). An Introduction to Viscous Flow. New York: McGraw-Hill, Inc.

Kai, T. (1995). A Survey of Separation Prediction Methods Applied to Potential Flow Solutions for an Airfoil. M. S. Thesis. Oklahoma State University.

Karr, C. L. (1995). Improved Computer Modeling of Separation Equipment Using Least Median Squares Curve Fitting and Genetic Algorithms. Advances in Filtration and Separation Technology, 9, 385-389.

Katz, J., & Plotkin, A. (1991). Low Speed Aerodynamics: From Wing Theory to Panel Methods. New York: McGraw-Hill, Inc.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671-680.

Kuethe, A. M., & Chow, C. (1976). Foundations of Aerodynamics: Bases of Aerodynamic Design (3rd ed.). New York: John Wiley & Sons.

Lee, K. D., & Eyi, S. (1992). Aerodynamic Design via Optimization. Journal of Aircraft, 31(1), 1012-1019.

Levy, S. (1992). Artificial Life. New York: Pantheon Books.

Liebeck, R. H. (1970). Optimization of Airfoils for Maximum Lift. Journal of Aircraft, 7(5), 409-415.

Liebeck, R. H. (1973). A Class of Airfoils Designed for High Lift in Incompressible Flow. Journal of Aircraft, 10(10), 610-617.

Liebeck, R. H. (1978). Design of Subsonic Airfoils for High Lift. Journal of Aircraft, 15(9), 547-561.

Moran, J. (1984). An Introduction to Theoretical and Computational Aerodynamics. New York: John Wiley & Sons, Inc.

Nelson, R. C. (1989). Flight Stability and Automatic Control. New York: McGraw-Hill, Inc.

Obayashi, S. & Takanashi, S. (1995). Genetic Optimization of Target Pressure Distributions for Inverse Design Methods. AIAA Paper 95-1649, American Institute of Aeronautics and Astronautics.

Ohm, D. (1994). Generalized Simulated Annealing for Function Optimization Over Continuous Variables. M. S. Thesis. Oklahoma State University.

Quagliarella, D., & Della Cioppa, A. (1994). Genetic Algorithms Applied to the Aerodynamic Design of Transonic Airfoils. AIAA Paper 94-1896, Colorado Springs, CO: American Institute of Aeronautics and Astronautics.

Rice, M. S. (1971). Handbook of Airfoil Sections for Light Aircraft. Appleton, WI: Aviation Publications.

Schlichting, H. (1987). Boundary Layer Theory. New York: McGraw-Hill, Inc.

Selig, M. S., & Guglielmo, J. J. (1994). High-Lift Low Reynolds Number Airfoil Design. 12th AIAA Applied Aerodynamics Conference, Colorado Springs, CO.

Srinivas, M., & Patnaik, L. M. (1994). Genetic Algorithms: A Survey. Computer, 27(6), 17-26.

Valarezo, W. O., & Chin, V. D. (1994). Method for the Prediction of Wing Maximum Lift. Journal of Aircraft, 31(1), 103-109.

Vanderplaats, G. N. (1984). Numerical Optimization Techniques for Engineering Design: With Applications. New York: McGraw-Hill, Inc.

Wang, F. (1991). Computer Aided Optimal Design of Duct Systems. M. S. Thesis. Oklahoma State University.

White, F. M. (1991). Viscous Fluid Flow (2nd ed.). New York: McGraw-Hill, Inc.

Winston, P. H. (1992). Artificial Intelligence (3rd ed.). Reading, MA: Addison-Wesley.

Yamamoto, K. & Inoue, O. (1995). Applications of Genetic Algorithm to Aerodynamic Shape Optimization. AIAA Paper 95-1650, American Institute of Aeronautics and Astronautics.

APPENDICES

APPENDIX A: GA.CON CONTROL FILE

Reynolds number:
250000
Number of panels (m):
150
Number of genes (NGENE = 2 x #airfoils + 1):
9
Thickness weightings of component airfoils (Liebeck, Wortmann, Selig, NACA, in order):
0. 0. 0. 1.
Camber weightings of component airfoils (Liebeck, Wortmann, Selig, NACA, in order):
0. 0. 0. 1.
Alpha (degrees):
0
Max mutations of thickness weightings (separated by spaces):
.1 .1 .1 .1
Max mutations of camber weightings (separated by spaces):
.1 .1 .1 .1
Max mutation of alpha (degrees):
1.
Number of test runs (NTESTMAX):
1
Number of generations per test run (NGENMAX):
300
Number of fittest (w/o diversity) to be selected (NBEST):
1
Number of additional niches (NNICHE):
49
DIVWEIGHT (1.01: diversity is tiebreaker, 0.99: fitness is tiebreaker):
1.01
Number of mutated offspring per survivor (NMUT):
1
Number of cross-breedings (NCROSS); each crossbreeding yields two offspring:
1
HALT after each generation to view output (y/n)?:
N
Termination fitness (TERMFIT); program stops if this value is reached:

3.0
Probability of selection (P: 0.0 - 1.0):
0.7
Number of generations between printing (NPRINT):
10
Randomly generated seed initial population? (IRANINIT: 1=yes, 0=no)
0
Purely random search? (IRAND: 1=yes, 0=no):
0

APPENDIX B: GA.F COMPUTER PROGRAM

```
c=====================================================================
c
c     this program, ga.f, optimizes an airfoil by a genetic algorithm.
c
c     the program can be adapted to optimize any objective function
c     simply by replacing the aerosolver subroutine with a different
c     fitness function (including constraints)
c
c     input variables include:
c
c     re:         reynolds number
c     m:          number of panels
c     ngene:      number of genes (2 x #airfoils + 1)
c     chromo:     `ngene' elements; containing thickness and camber
c                    weightings of basis airfoils, plus alpha
      maxmut:     `ngene' elements; contains maximum allowable mutation
c                    size for each gene
c     ntestmax:  number of test runs
c     ngenmax:   number of generations per test run
c     nbest:      number of primary survivors to be selected
c     nniche:     number of `niche' survivors to be selected
c     divweight: determines the tiebreaker in rank-space selection.
c                   (either diversity rank (1.01) or fitness rank (0.99))
c     nmut:       number of mutated offspring per survivor
c     ncross:     number of cross-breedings per generation.  each cross-
c                    breeding produces two offspring
c     halt:       determines whether chromosomes, mutations, fitness,
c                    etc. are displayed for each generation (y/n)
c     termfit:    termination fitness.  program stops if this value is
c                              reached
c     p:          probability factor, normally between 0.5 and 1.0.
c                     controls amount of bias toward selection of most
c                       fit/diverse indiv.
c     nprint:     number of generation between printouts of primary
c                    survivor information
c     iraninit:  determines if seed population is to be randomly
c                    generated within domain space (1=yes, 0=no)
c     irand:      allows purely random search, if desired (1=yes, 0=no)
c
c=====================================================================

      integer  fitrank(200),divrank(200,200),combrank(200,200),
     +         temprank(200), surv(200), nptr(200), survctr, generat
      real     chromo(200,200), fitness(200), div(200,200), temp(200),
     +         maxmut(10), adam(200), maxfit(1000), avgfit(1000),
     +         initmut(10)

      character halt
      common /rey/ re
```

```
      character  (len = 10) date, time, zone
      integer date_time(8)

      pi = 4. * atan(1.)

      open(unit = 1, file = 'ga.con', status = 'old')
      open(unit = 2, file = 'ga.out', status = 'unknown')
      open(unit = 8, file = 'cps.dat', status = 'unknown')

      write(*,11)'generation', 'maxfit', 'avgfit'
      write(2,11)'generation', 'maxfit', 'avgfit'
11    format(a16, a10, a10)

c-----------------
c     get user input and initialize variables
c-----------------

      call user_input(m, ntestmax, ngenerat, ngene, nbest, nniche,
     +  divweight, nmut, maxmut, ncross, adam, halt, termfit, p, nprint,
     +  iraninit, irand)
      close(unit = 1)

      do 3000 ntest = 1,ntestmax

      nind = (nbest + nniche) * (1 + nmut + 2 * ncross)
      survctr = nbest + nniche

c-----------------
c     initialize the random number generator for each generation
c-----------------

      do 1000 generat = 1, ngenerat

        call date_and_time(date, time, zone, date_time)
        ranval = real(date_time(8))
        call srand(ranval)
        ranval = rand()

c-----------------
c     create the seed population, using a large initial mutation
c     (if desired) from a desired starting design, or by randomly
c     selecting genes within the solution space
c-----------------

        if (generat .eq. 1) then
          surv(1) = 1
          do 10 j = 1, ngene
            chromo(1,j) = adam(j)
10          initmut(j) = 1. * maxmut(j)
          call mutate(chromo,surv,1,nind-1,ngene,initmut,0,nind,halt)
          if (iraninit .eq. 1) then
            do 20 i = 1, nind
            do 20 j = 1, ngene
              ranval = rand()
              chromo(i,j) = 2.5 * (ranval - 0.2)
20          continue
          end if
        else
          call mutate(chromo, surv, survctr, nmut, ngene, maxmut,
     +                ncross, nind, halt)
        end if

        if (irand .eq. 1) then
          do 30 i = 1, nind
```

```
          do 30 j = 1, ngene
            ranval = rand()
            chromo(i,j) = 2.5 * (ranval - 0.2)
30        continue
        end if

c-----------------
c     calculate fitness and fitness rank for each individual
c-----------------

        do 40 i = 1, nind
          nshow = 0
          do 50 j = 1, ngene
50          temp(j) = chromo(i,j)
          call aerosolv(m, temp, nshow, fitlm, ngene)
          fitness(i) = fitlm
40      continue

        call rankem(fitness, fitrank, nptr, nind)
        do 60 i = 1, nbest
60        surv(i) = nptr(i)
        survctr = nbest

c-----------------
c     determine the diversity rank for each individual
c-----------------

        do 70 niche = 1, nniche
          call diversity(chromo, surv, survctr, div, nind, niche, ngene)
            do 80 i = 1, nind
80            temp(i) = -1. * div(niche,i)
            call rankem (temp, temprank, nptr, nind)
            do 90 i = 1, nind
90            divrank(niche,i) = temprank(i)

c-----------------
c     use fitness and diversity to select survivors (rank-space method)
c-----------------

            call rspace (fitrank, divrank, combrank, nptr, nind, surv,
     +                   niche, survctr, divweight, p, nbest, ctr)

70      continue

c-----------------
c     output results
c-----------------

        totfit = 0.
        do 100 i = 1,survctr
100       totfit = totfit + fitness(surv(i))
        avgfit(generat) = totfit / survctr
        maxfit(generat) = fitness(surv(1))
        write(2,101)  generat, maxfit(generat), avgfit(generat)
        write(*,101)  generat, maxfit(generat), avgfit(generat)
101      format(i16, f10.4, f10.4)

        if((generat.eq.1) .or. (generat/nprint*nprint.eq.generat)) then
          do 110 j = 1, ngene
110         temp(j) = chromo(surv(1),j)

          write(*,111)generat,(temp(j), j = 1, ngene-1), 10.*temp(ngene)
          write(8,111)generat,(temp(j), j = 1, ngene-1), 10.*temp(ngene)
111       format(//' generation:',i6/,' thickness weightings:',4f7.4/,
```

```
     +     '    camber weightings:',4f7.4/,5x,' alpha (degrees):',f7.4)

           nshow = 1
           call aerosolv(m, temp, nshow, fitlm, ngene)
           fitness(i) = fitlm
         end if


         if ((halt .eq. 'y') .or. (halt .eq. 'Y')) then
           pause
           write(*,121)'i','thickness weightings ','camber weightings  ',
     +      'alpha/10','fitness','fitrnk', 'div', 'divrnk','combrnk'
121        format(a3, a24, 4x, a24, 4x, a8, 2x, 5a10)

           do i = 1, nind
           write(*, 131) i, (chromo(i,j),j=1,ngene), fitness(i),
     +         fitrank(i), div(1,i), divrank(1,i), combrank(1,i)
131          format(i3, 4f6.3,4x,4f6.3,4x,f6.3,4x,f10.3,i10,f10.3,2i10
           end do

           write(*,*) `primary survivor(s):  '
           do i = 1, nbest
             write(*,*) surv(i)
           end do

           do niche = 1, nniche
             write(*,141)'survivor, niche',niche,': ',surv(niche+nbest)
141          format(a20,i3,a3,i3)
           end do
         end if

c-----------------
c     continue for prescribed number of generations or until termination
c     fitness has been reached.  print airfoil genes and performance
c     characteristics of all final survivors.
c-----------------

         if (fitness(surv(1)) .gt. termfit) go to 2000

1000   continue

2000   write(*,2001) generat
       write(8,2001) generat
2001   format(//,'  program terminated after', i6, ' generations')

       do i = 1, survctr

         do 120 j = 1, ngene
120        temp(j) = chromo(surv(i),j)

         write(*,151)generat,i,(temp(j), j = 1, ngene-1), 10.*temp(ngene)
         write(8,151)generat,i,(temp(j), j = 1, ngene-1), 10.*temp(ngene)
151       format(//' gen:',i5,'   surv #:',i3/,' thickness weightings:',
     +       4f7.4/,'    camber weightings:',4f7.4/,15x,' alpha:',f7.4)

         nshow = 1
         call aerosolv(m, temp, nshow, fitlm, ngene)
         fitness(i) = fitlm
       end do

3000   continue

       end
```

```
C=======================================================================
C
C     subroutine get_input:  read genetic algorithm parameters
C
C=======================================================================
      subroutine user_input(m, ntestmax, ngenerat, ngene, nbest, nniche,
     +  divweight, nmut, maxmut, ncross, adam, halt, termfit, p, nprint,
     +  iraninit, irand)

      real        maxmut(10), adam(200)
      character   halt, comment
      common /rey / re

        read(1,'(a)') comment
      read(1,*)  re
        read(1,'(a)') comment
      read(1,*)  m
        read(1,'(a)') comment
      read(1,*)  ngene
        read(1,'(a)') comment
      read(1,*)  (adam(i), i = 1, ngene/2)
        read(1,'(a)') comment
      read(1,*)  (adam(i), i = ngene/2+1, ngene-1)
        read(1,'(a)') comment
      read(1,*)  alphd
        read(1,'(a)') comment
      read(1,*)  (maxmut(i), i = 1, ngene/2)
        read(1,'(a)') comment
      read(1,*)  (maxmut(i), i = ngene/2+1, ngene-1)
        read(1,'(a)') comment
      read(1,*)  alphmut
        read(1,'(a)') comment
      read(1,*)  ntestmax
        read(1,'(a)') comment
      read(1,*)  ngenerat
        read(1,'(a)') comment
      read(1,*)  nbest
        read(1,'(a)') comment
      read(1,*)  nniche
        read(1,'(a)') comment
      read(1,*)  divweight
        read(1,'(a)') comment
      read(1,*)  nmut
        read(1,'(a)') comment
      read(1,*)  ncross
        read(1,'(a)') comment
      read(1,'(a)')  halt
        read(1,'(a)') comment
      read(1,*) termfit
        read(1,'(a)') comment
      read(1,*)  p
        read(1,'(a)') comment
      read(1,*)  nprint
        read(1,'(a)') comment
      read(1,*)  iraninit
        read(1,'(a)') comment
      read(1,*)  irand

C------------------
C     make alpha and alphmut the same magnitude as the other genes
C------------------
      adam(ngene) = alphd * .1
      maxmut(ngene) = alphmut * .1
      end
```

```
C=======================================================================
C
C     subroutine rankem:  rank orders the elements of an array (highest
C        to lowest).
C
C=======================================================================

      subroutine rankem(qual, qualrank, n, nind)

      integer    qualrank(200), n(200), best
      real       qual(200), qualbest

      do i = 1, nind
      n(i) = i
      end do

C------------------
C     sort pointer array in order of qual values
C------------------

      do 20 i = 1, nind-1
        best = i
        qualbest = qual(n(i))
        do 30 j = i+1, nind
          if (qual(n(j)) .gt. qualbest) then
            best = j
            qualbest = qual(n(j))
          end if
30      continue

C------------------
C     exchange pointers if necessary
C------------------

        if (best .ne. i) then
          ntemp = n(i)
          n(i) = n(best)
          n(best) = ntemp
        end if
20    continue

      do i = 1, nind
        qualrank(n(i)) = i
      end do

      end
```

```
C=======================================================================
C
C     subroutine diversity:  calculates the diversity factor for each
C     individual, based upon its diversity from all previously selected
C     survivors.
C
C=======================================================================

      subroutine diversity (chromo,surv,survctr,div,nind,niche,ngene)

      integer    surv(200), nind, survctr, niche, ngene
      real       chromo(200,200), div(200,200), suppdiv

      if (niche .eq. 1) then
      do 210 i = 1, nind
        div(1,i) = 0.
        do 210 j = 1, survctr
          dsqd = 0.0001
          do 211 k = 1, ngene
211          dsqd = dsqd + (chromo(i,k) - chromo(surv(j),k))**2.
210          div(1,i) = div(1,i) + 1./dsqd

      else
      do 220 i = 1, nind
        dsqd = 0.0001
        do 221 k = 1, ngene
221       dsqd = dsqd + (chromo(i,k) - chromo(surv(survctr),k))**2.
220     div(niche,i) = div(niche-1,i) + 1./dsqd

      end if

C------------------
C     set diversity of previously selected survivors to 0.0
C------------------

      do i = 1, survctr
        div(niche,surv(i)) = 10000.0
      end do

      end
```

```
c=======================================================================
c
c     subroutine rspace:  uses a rank-space method to calculate combined
c     rank for each individual, incorporating a user-supplied weighting
c     factor (divweight) and probabilistic selection.
c
c=======================================================================

      subroutine rspace (fitrank, divrank, combrank, nptr, nind, surv,
     +                   niche, survctr, divweight, p, nbest, ctr)

      integer    fitrank(200), divrank(200,200), combrank(200,200),
     +           totrank(200),surv(200),survctr, nptr(200), ctr(200)
      real       tot(200), prob(200), selectpt(200)

      do i = 1, nind
      tot(i) = -1. * (fitrank(i) + divweight * divrank(niche,i))
      end do

c------------------
c     remove previously selected individuals from consideration (hence,
c     the factor of 10.* nind) and rank the remaining individuals.  then
c     use probabilistic selection to determine additional survivors
c------------------

      do i = 1, survctr
        tot(surv(i)) = 10. * nind * tot(surv(i))
      end do

      call rankem (tot, totrank, nptr, nind)

      do i = 1, nind
        combrank(niche,i) = totrank(i)
      end do

      ncand = nind - survctr
      ncm1 = ncand - 1

      prob(nptr(1)) = p
      selectpt(nptr(1)) = p
      do i = 2, ncm1
        prob(nptr(i)) = p * ((1-p)**(i-1))
        selectpt(nptr(i)) = selectpt(nptr(i-1)) + prob(nptr(i))
      end do
      selectpt(nptr(ncand)) = 1.

      ranval = rand()

      do i = 1, ncand
        if(ranval .lt. selectpt(nptr(i))) then
          surv(nbest + niche) = nptr(i)
          goto 410
        end if
      end do

410   do i = 1, survctr
        divrank (niche,surv(i)) = 0
        combrank(niche,surv(i)) = 0
      end do

      survctr = survctr + 1

      end
```

```
c=======================================================================
c
c     subroutine mutate:  mutates and/or breeds the survivors.
c     maximum mutation size (maxmut) and number of mutated offspring
c     (nmut) is supplied by the user.  mates are selected at random and
c     each union produces two additional (unmutated) offspring.
c
c=======================================================================

      subroutine mutate(chromo, surv, survctr, nmut, ngene, maxmut,
     +                  ncross, nind, halt)

      integer      surv(200), survctr, nmut, ngene, cutpoint, mate(200)
      real         chromo(200,200), chrnext(200,200), mut, maxmut(10)
      character    halt

      do 305 i = 1, survctr
      do 305 j = 1, ngene
305     chrnext(i,j) = chromo(surv(i),j)

c------------------
c     get random number and apply mutations
c------------------

      next = survctr + 1

      do 310 i = 1, survctr
      do 310 j = 1, nmut

        do k = 1, ngene
          ranval = rand()
          mut = (ranval - 0.5) * 2. * maxmut(k)
          chrnext(next,k) = chromo(surv(i),k) + mut
        end do

        next = next + 1

310   continue


c------------------
c     mate each survivor with one of the other randomly selected
c     survivors by swapping the genes before/after the cutpoint
c------------------

      next = survctr * (1 + nmut) + 1

      do 350 i=1, ncross

        if ((halt .eq. 'y') .or. (halt .eq. 'Y')) then
          write(*,331)'survivor', 'mate', 'cutpoint', 'offspring'
331       format(/,3a10, a30)
        end if

        do 350 j = 1, survctr
          ranval = rand()
          mate(surv(j)) = surv(int(1 + (ranval * survctr)))
          ranval = rand()
          cutpoint = int(1. + ngene * ranval)
          do k = 1, cutpoint
            chrnext(next,k) = chromo(surv(j),k)
            chrnext(next+1,k) = chromo(mate(surv(j)),k)
          end do
          do k = cutpoint+1, ngene
```

```
            chrnext(next,k) = chromo(mate(surv(j)),k)
            chrnext(next+1,k) = chromo(surv(j),k)
          end do

          if ((halt .eq. 'y') .or. (halt .eq.'Y')) then
            write(*,341) surv(j), mate(surv(j)), cutpoint,
     +          (chrnext(next,k),k=1,ngene), (chrnext(next+1,k),
     +          k=1,ngene)
341         format(3i10, 9f7.3, /,30x, 9f7.3,/)
          end if

        next = next + 2
350    continue

       do 370 i = 1, nind
       do 370 j = 1, ngene
370      chromo(i,j) = chrnext(i,j)

       end
```

APPENDIX C: AEROSOLVE SUBROUTINE

```
c=====================================================================
c
c     subroutine aerosolv:  evaluates the fitness of each airfoil.
c     this subroutine uses the smith-hess panel method to calculate
c     the flowfield about an arbitrary 2d body in an ideal flow.
c
c               written by dr. a arena 10 november, 1993
c                modified by peter jahns 19 april, 1995
c
c=====================================================================

      subroutine aerosolv(m, temp, nshow, fitlm, ngene)

      real ep(200,2), ept(200,2), pt1(200,2), pt2(200,2)
      real co(200,2), a(200,200), b(200,200), g(200)
      real th(200), dl(200), cp(200), locvel(200), temp(200)
      common /rey/ re

      pi=4.*atan(1.)
      n = m + 2
      alpha = 10. * pi * temp(ngene) / 180.

c     read in the panel endpoints using subroutine body

      call body(m, ept, temp, ngene, thickmax,ithflag)

c     convert panelling to clockwise

      do i = 1, m+1
        ep(i,1) = ept(m+1-i+1,1)
        ep(i,2) = ept(m+1-i+1,2)
      end do

c     establish coordinates of panel end points

      do i=1,m
        pt1(i,1)=ep(i,1)
        pt2(i,1)=ep(i+1,1)
        pt1(i,2)=ep(i,2)
        pt2(i,2)=ep(i+1,2)
      end do

c     find panel angles th(j)

      do i=1,m
        dz=pt2(i,2)-pt1(i,2)
        dx=pt2(i,1)-pt1(i,1)
        th(i) = atan2(dz,dx)
        dl(i) = sqrt(dz*dz + dx*dx)
      end do
```

```
c       establish collocation points

      do i=1,m
        co(i,1)=(pt2(i,1)-pt1(i,1))/2+pt1(i,1)
        co(i,2)=(pt2(i,2)-pt1(i,2))/2+pt1(i,2)
      end do

c       calculate trailing edge angle to determine cl and cm corrections

      zte = ep(9*m/10,2) - ep(m+1,2)
      ste = 1. - ep(9*m/10,1)
        thte = atan2(zte,ste)

c     establish influence coefficients and convert collocation points
c     to local panel coordinates


      do i=1,m
        uv=0.
        wv=0.

        do j=1,m
          xt=co(i,1)-pt1(j,1)
          zt=co(i,2)-pt1(j,2)
          x2t=pt2(j,1)-pt1(j,1)
          z2t=pt2(j,2)-pt1(j,2)

          x=xt*cos(th(j))+zt*sin(th(j))
          z=-xt*sin(th(j))+zt*cos(th(j))
          x2=x2t*cos(th(j))+z2t*sin(th(j))
          z2=0.

c     find r1, r2, th1, th2

          r1=sqrt(x*x+z*z)
          r2=sqrt((x-x2)*(x-x2) + z*z)

          th1=atan2(z,x)
          th2=atan2(z,x-x2)

c       compute velocity in local panel reference frame

          if(i.eq.j) then
            ul=0.
            wl=0.5
            ulv= 0.5
            wlv= 0.
          else
            ul=1./(2.*pi)*log(r1/r2)
            wl=1./(2.*pi)*(th2-th1)
            ulv = 1./(2.*pi)*(th2-th1)
            wlv = 1./(2.*pi)*log(r2/r1)
          end if

c     return velocity to global reference frame
          u=ul*cos(-th(j))+wl*sin(-th(j))
          w=-ul*sin(-th(j))+wl*cos(-th(j))
          uv = uv + ulv*cos(-th(j)) + wlv*sin(-th(j))
          wv = wv - ulv*sin(-th(j)) + wlv*cos(-th(j))
```

```
c     a(i,j) is the influence coefficient defined by the
c     tangency condition.  b(i,j) is the induced local
c     tangential velocity to be used in cp calculation

          a(i,j)=-u*sin(th(i))+w*cos(th(i))
          b(i,j)=u*cos(th(i))+w*sin(th(i))

        end do

        a(i,m+1) = -uv*sin(th(i)) + wv*cos(th(i))
        b(i,m+1) = uv*cos(th(i)) + wv*sin(th(i))

c     rhs

        a(i,n)=cos(alpha)*sin(th(i))-sin(alpha)*cos(th(i))

      end do

c     kutta condition

      do j=1,m+1
        a(m+1,j) = b(1,j) + b(m,j)
        a(m+1,m+2) = - ( cos(alpha-th(1)) + cos(alpha-th(m)) )
      end do

c     solve for solution vector

      call matrx(a,n,g)

c     convert source strengths into tangential velocities
c     along the airfoil surface and cp's on each panel

      cl = 0.
      cf = 0.
      cm0 = 0.
      cpmin = 0.

      do i=1,m
        vel=0.
        do j=1,m+1
          vel=vel+b(i,j)*g(j)
        end do
        cp(i)=1.-(vel+cos(alpha-th(i)))**2
        cpmin = min(cpmin, cp(i))
        locvel(i) = vel + cos(alpha-th(i))
        cf= cf-cp(i)*dl(i)*( cos(th(i)) )
        cl = cl - cp(i) * dl(i) * cos(th(i) - alpha)
        cm0 = cm0-cp(i)*dl(i)*(-co(i,1)*cos(th(i))+co(i,2)*sin(th(i)))

        if (nshow .eq. 1) then
          write(8,116)co(i,1),co(i,2),cp(i),locvel(i)
          write(*,116)co(i,1),co(i,2),cp(i),locvel(i)
116       format(4f16.4)
        end if
      end do

      call sep_pred(co, cp, locvel, m, xsep)

      xcp = -cm0/cf
      cmc4 = -(xcp-0.25)*cf

      ssep = (1. - xsep) / cos(thte)
      delcl = -1. * pi * ssep * (thte + alpha)
      delcm = -0.25 * delcl * (xsep**1.5)
```

```
      clcorr = cl + delcl
      cmcorr = cmc4 + delcm

c--------------------------
c       set constraints on alpha, cm, thickness, and cp 'spikes'
c--------------------------

      fitlm = clcorr
      if (temp(ngene) .gt. 1.0) fitlm = 0.
      if (cmcorr .lt. -0.1) fitlm = 0.
      if (thickmax .gt. .1) fitlm = 0.
      if (ithflag .eq. 1) fitlm = 0.
      if (cpmin .lt. -6.) fitlm = 0.

      if (nshow .eq. 1) then
        write(*,117)cl, xcp, cmc4, xsep, clcorr, fitlm, cmcorr
        write(8,117)cl, xcp, cmc4, xsep, clcorr, fitlm, cmcorr
117     format(/,' cl:',f6.4,' cp:',f6.4,' cm:',f6.4,
     +           ' xsep:',f6.4,' clc:',f6.4,' fit:',f6.4,' cmc:',f6.4)
      end if

      end

c=======================================================================
c
c       subroutine matrx is a matrix reducer of the gaussian type
c       a(i,j) is the matrix, g(i) is the solution vector
c
c=======================================================================


      subroutine matrx(a,n,g)

      real a(200,200), g(200)

c               initialize the g vector to all zeros

      do i=1,n-1
        g(i)=0.
      end do

c               convert coefficient matrix
c               to upper triangular form

      do i=1,n-1
        if(abs(a(i,i)).lt.0.0000001)write(*,*)'zero diag. error'

        p=a(i,i)
        do j=i,n
          a(i,j)=a(i,j)/p
        end do

        do k=i+1,n-1
          p2=a(k,i)
          do l=i,n
            a(k,l)=a(k,l)-p2*a(i,l)
          end do
        end do
      end do
```

```
c               back substitute triangularized matrix to get
c               values of solution vector

      do i=n-1,1,-1
        g(i)=a(i,n)
        do j=1,n-1
          a(i,i)=0.
          g(i)=g(i)-a(i,j)*g(j)
        end do
      end do

      return

c       stop
      end

c=======================================================================
c
c     subroutine body:  calculates the nodal coordinates of the body
c     surface panels for an airfoil. data is nondimensionalized by chord
c     note: panel 1 @ te. top sfc., numbering scheme counterclockwise
c
c=======================================================================

        subroutine body(m, ept, temp, ngene, thickmax, ithflag)

      integer     cutpoint
      real ept(200,2), temp(200), tbar(10), ybar(10), a(10), b(10)

      pi=4.*atan(1.)
      cutpoint = ngene / 2
      ithflag = 0
      thickmax = 0.

      do i = 1, cutpoint
        a(i) = temp(i)
        b(i) = temp(cutpoint+i)
      end do

      do i=1,1+m
        theta=2.*pi*float(i-1)/float(m)
        xc =  ( 1. + cos(theta) ) * 0.5

c     tbar(i) represents the thickness function of each basis airfoil
c     a(i) represents the weighting factors (genes)

c     tbar(1) = liebeck la 2566
c     tbar(2) = wortman fx 63-137
c     tbar(3) = selig s1223
c     tbar(4) = naca4412


      tbar(1) = .2908*sqrt(xc) - .4943*xc + 1.5941*xc**2 - 5.0118*
     +    xc**3 + 7.2482*xc**4 - 4.9212*xc**5 + 1.294*xc**6 + .0002
      tbar(2) = .1900*sqrt(xc) -.0809*xc - .1527*xc**2 -
     +                  .0723*xc**3 + .1154*xc**4 + 0.0005
      tbar(3) = .2352*sqrt(xc) - .2079*xc + .0976*xc**2 -
     +        1.4713*xc**3 + 3.3114*xc**4 - 2.8047*xc**5 + .8397*xc**6
      tbar(4) = 0.6*(.2969*sqrt(xc) - .126*xc - .3537*xc**2
     +             + .2843*xc**3 - .1015*xc**4)

      thick = a(1)*tbar(1)+a(2)*tbar(2)+a(3)*tbar(3)+a(4)*tbar(4)
```

```
      if ((xc .lt. 0.0001) .or. (xc .gt. 0.9999)) then
        ithick = 0.
      else if (thick .lt. .5*tbar(3)) then
        ithflag = 1
      end if
      thickmax = max(thickmax, thick)

      z = thick*sign(1.,sin(theta))

c     ybar(i) represents the camber function of each basis airfoil
c     b(i) represents the weighting factors (genes)

      ybar(1) = .017*sqrt(xc) + .311*xc - .762*xc**2 + .5865*xc**3
     +         - .4073*xc**4 + .4907*xc**5 - .2363*xc**6 + .0004
      ybar(2) = .0017 + .3958*xc - 1.826*xc**2 + 5.5337*xc**3 -
     +               9.268*xc**4 + 7.6531*xc**5 - 2.4893*xc**6
      ybar(3) = .0265*sqrt(xc) + .476*xc - 1.6674*xc**2 +
     +    4.4108*xc**3 - 7.7664*xc**4 + 7.0934*xc**5 - 2.5729*xc**6
      if (xc .le. 0.4) then
        ybar(4) = 0.25*(.8*xc - xc**2)
      else
        ybar(4) = 1./9. * (.2 + .8*xc - xc**2)
      end if


      y = b(1)*ybar(1) + b(2)*ybar(2) + b(3)*ybar(3) + b(4)*ybar(4)

      z = y + z

      ept(i,1)=xc
      ept(i,2)=z

      end do


      return
      end

c=======================================================================
c
c       subroutine sep_pred
c
c       integral method for calculation of boundary layer
c       growth on an airfoil, starting at a stagnation point
c
c       thwaites's method used for laminar-layer flow
c       michel's method used to fix transition
c       head's method used for turbulent-flow region
c
c=======================================================================

      subroutine sep_pred(co, copres, locvel, datanumber, xsep)

      common /num/ pi,nx
      dimension  yy(200)
      common  xx(200),vgrad(200),theta(200)
      common /inp1/ psx(200),psy(200),pcp(200),pve(200)
      common /inp2/ x(200),y(200),cp(200),ve(200)
      common /rey/ re
      common /bod/ tau
      real  lambda, l, co(200,2), copres(200), locvel(200)
      integer datanumber, velmax

      pi = 4.*atan(1.)
```

```
      stagpoint = 1
      k = 0

      do 8 i=1,datanumber
        psx(i) = co(i,1)
        psy(i) = co(i,2)
        pcp(i) = copres(i)
        pve(i) = locvel(i)
        if((pve(i) .ge. 0.0) .and. (k .eq. 0)) then
          stagpoint = i
          k = 1
        endif
8     continue

c
c     re-order data starting at stagnation point
c

      nx = datanumber - stagpoint +1

      do 12 i=1,nx
        x(i) = psx(stagpoint - 1 + i)
        y(i) = psy(stagpoint - 1 + i)
        cp(i) = pcp(stagpoint - 1 + i)
        ve(i) = pve(stagpoint - 1 + i)
12    continue

c
c     find the point where ve is maximum
c

      cpm = 1
      velmax = 1
      do 10 i=1,nx
        if(cp(i) .lt. cpm) then
          cpm = cp(i)
          velmax = i
        endif
10    continue

c
c     find distances between nodes along surface
c

      xx(1) = 0.0
      do 100 i=2,nx
        dx = x(i) - x(i-1)
        dy = y(i) - y(i-1)
100   xx(i) = xx(i-1) + sqrt(dx * dx + dy * dy)

c
c     find velocity gradient at nodes
c

      v1 = ve(3)
      x1 = xx(3)
      v2 = ve(1)
      x2 = xx(1)
      xx(nx+1) = xx(nx-2)
      do 110 i=1,nx
        v3 = v1
        x3 = x1
        v1 = v2
```

```
        x1 = x2
        v2 = ve(nx-2)
        if(i .lt. nx)   v2 = ve(i+1)
          x2 = xx(i+1)
          fact = (x3-x1)/(x2-x1)
          vgrad(i) = ((v2 - v1)*fact - (v3 - v1)/fact)/(x3 - x2)
110   continue

      xtrans = x(velmax)

c
c     laminar-flow region
c
      if (vgrad(1) .le. 0.0) then
        xsep = 0.
        go to 1040
      end if

      theta(1) = sqrt(.075/re/vgrad(1))
      i = 1
200   lambda = theta(i)*theta(i) * vgrad(i)*re
      if(lambda .lt. -.0842)  goto 400
      call thwats(lambda,h,l)
      cf = 2. * l/re/theta(i)
      if(i .gt. 1)  cf = cf/ve(i)
      i = i + 1
      dth2ve6 = .225 * (ve(i)**5 + ve(i-1)**5) * (xx(i) - xx(i-1))/re
      theta(i) = sqrt(((theta(i-1)**2) * (ve(i-1)**6) + dth2ve6)
     +             /(ve(i)**6))
      if(i .eq. 2)    theta(2) = theta(1)

c
c       test for transition
c

      if((x(i) .gt. xtrans) .and. (y(i) .gt. 0.0)) goto 300
      goto 200

210   rex = re * xx(i) * ve(i)
      ret = re * theta(i) * ve(i)
      retmax = 1.174 * (1. + 22400./rex) * rex**.46
      if(ret .lt. retmax)   goto 200

c
c     turbulent-flow region
c

300   itrans = i

310   h = 1.5
      i = itrans
      yy(2) = h1ofh(h)
      yy(1) = theta(i-1)
320   dx = xx(i)-xx(i-1)
      call runge2(i-1,i,dx,yy,2)
      theta(i) = yy(1)
      h = hofh1(yy(2))
      rtheta = re * ve(i) * theta(i)
      cf = cfturb(rtheta,h)
      if(h .gt. 2.4)  goto 410
      i = i + 1
      if(i .le. nx)  goto 320

      xsep = x(nx)
```

```
        x1 = x2
        v2 = ve(nx-2)
        if(i .lt. nx)    v2 = ve(i+1)
          x2 = xx(i+1)
          fact = (x3-x1)/(x2-x1)
          vgrad(i) = ((v2 - v1)*fact - (v3 - v1)/fact)/(x3 - x2)
110   continue

      xtrans = x(velmax)

c
c     laminar-flow region
c
      if (vgrad(1) .le. 0.0) then
        xsep = 0.
        go to 1040
      end if

      theta(1) = sqrt(.075/re/vgrad(1))
      i = 1
200   lambda = theta(i)*theta(i) * vgrad(i)*re
      if(lambda .lt. -.0842)  goto 400
      call thwats(lambda,h,l)
      cf = 2. * l/re/theta(i)
      if(i .gt. 1)  cf = cf/ve(i)
      i = i + 1
      dth2ve6 = .225 * (ve(i)**5 + ve(i-1)**5) * (xx(i) - xx(i-1))/re
      theta(i) = sqrt(((theta(i-1)**2) * (ve(i-1)**6) + dth2ve6)
     +             /(ve(i)**6))
      if(i .eq. 2)    theta(2) = theta(1)

c
c       test for transition
c

      if((x(i) .gt. xtrans) .and. (y(i) .gt. 0.0)) goto 300
      goto 200

210   rex = re * xx(i) * ve(i)
      ret = re * theta(i) * ve(i)
      retmax = 1.174 * (1. + 22400./rex) * rex**.46
      if(ret .lt. retmax)   goto 200

c
c     turbulent-flow region
c

300   itrans = i

310   h = 1.5
      i = itrans
      yy(2) = h1ofh(h)
      yy(1) = theta(i-1)
320   dx = xx(i)-xx(i-1)
      call runge2(i-1,i,dx,yy,2)
      theta(i) = yy(1)
      h = hofh1(yy(2))
      rtheta = re * ve(i) * theta(i)
      cf = cfturb(rtheta,h)
      if(h .gt. 2.4)  goto 410
      i = i + 1
      if(i .le. nx)  goto 320

      xsep = x(nx)
```

```
      goto 1040


400   xsep = x(i)
      goto 1040

410   xsep = x(i)


1040  end


C=====================================================================

      subroutine thwats(lambda,h,l)
C
C       thwaites's correlation formulas
C

      real l,lambda

      if(lambda .lt. 0.0)   goto 100
      l = .22 + lambda * (1.57 - 1.8 * lambda)
      h = 2.61 - lambda * (3.75 - 5.24 * lambda)
      goto 200
100     l = .22 + 1.402 * lambda + .018 * lambda/(.107 + lambda)
      h = 2.088 + .0731 /(.14 + lambda)
200     return
      end

C=====================================================================

      function h1ofh(h)
C
C       head's correlation formula for h1(h)
C

      if(h .gt. 1.6)   goto 100
      h1ofh = 3.3 + .8234 * (h - 1.1)**(-1.287)
      return
100     h1ofh = 3.3 + 1.5501 * (h - .6778)**(-3.064)
      return
      end

C=====================================================================

      function hofh1(h1)
C
C       inverse of h1(h)
C

      if(h1 .lt. 3.3)   goto 110
      if(h1 .lt. 5.3)   goto 100
      hofh1 = 1.1 + .86 * (h1 - 3.3)**(-.777)
      return
100     hofh1 = .6778 + 1.1536 * (h1 - 3.3)**(-.326)
      return
110     hofh1 = 3.0
      return
      end
```

```
c=======================================================================

      function cfturb(rtheta,h)
c
c       ludwieg-tillman skin friction formula
c

        cfturb = .246 * (10. **(-.678 * h)) * (abs(rtheta))**(-.268)
      return
      end

c=======================================================================

        subroutine derivs(i)
c
c       set derivatives oy vector y
c

        common /rnk/ yt(200),yp(200)
        common       xx(200),vgrad(200),theta(200)
        common /inp2/ x(200),y(200),cp(200),ve(200)
        common /rey/ re

      h1 = yt(2)
      if(h1 .le. 3.)  return
      h = hofh1(h1)
      rtheta = re * ve(i) * yt(1)
      yp(1) = -(h + 2.) * yt(1) *vgrad(i)/ve(i) + .5 * cfturb(rtheta,h)
      yp(2) = -h1 * (vgrad(i)/ve(i) + yp(1)/yt(1))
     +          + .0306 * (h1 - 3.)**(-.6169)/yt(1)
      return
      end

c=======================================================================

      subroutine runge2(i0,i1,dx,yy,n)
c
c       2nd-orderrunge-kutta method for system of n first
c       order equations
c

        dimension ys(200),yy(200)
        common /rnk/ yt(200),yp(200)

      intvls = i1-i0
      if (intvls .lt. 1)   goto 200
      do 130 i=1,intvls
      do 100 j=1,n
100     yt(j) = yy(j)
        call derivs(i0 + i - 1)
      do 110 j=1,n
        yt(j) = yy(j) + dx * yp(j)
110     ys(j) = yy(j) + .5 * dx * yp(j)
        call derivs(i0 + 1)
      do 120 j=1,n
120     yy(j) = ys(j) + .5 * dx * yp(j)
130     continue
200     return
      end
```

VITA

Peter Jahns

Candidate for the Degree of

Master of Science

Thesis: A GENETIC ALGORITHM APPLIED TO THE OPTIMIZATION OF AIRFOIL DESIGN

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Tulsa, Oklahoma, on September 8, 1963, the son of Hans O. and Suse Jahns. Married Rhonda K. Gibson on May 20, 1995.

Education: Graduated from Memorial Senior High School, Houston, Texas, in May 1981; received Bachelor of Science degree in Aeronautical Engineering from United States Air Force Academy, Colorado Springs, Colorado, in May 1985; received Master of Education degree in International Relations from Northwestern Oklahoma State University, Alva, Oklahoma, in July 1993; completed requirements for the Master of Science degree with a major in Mechanical Engineering at Oklahoma State University in December 1995.

Experience: F-15 Instructor Pilot, United States Air Force, 1986-1990; T-37 Instructor Pilot, United States Air Force, 1990-1993.