NEAR EAST UNIVERSITY

Faculty of Engineering

Departmen of Computer Engineering

Genetic Algorithm Based Optimization

Graduation Project
COM-400

Student: Gülsever Güler

Supervisor: Assoc. Prof. Dr. Rahib Abiyev

Nicosia-2004

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor Assoc. Prof. Dr. Rahib Abiyev for his great advice and recommendations to finish this work properly.Although, I faced many problem collection data but has quiding me the appropriate references.(Dr Rahib) thanks a lot for your invaluable and continual support.

Second, I would like to thank my family for their constant encouragement and support during the preparation of this work.

Third, I thank all the staff of the faculty of engineering for giving me the facilities to practice and solving any problem I was facing during working in this project.

Fanally, thanks for all of my friends for their advices and support.

ABSTRACT

By increasing complexity of processes, it has become very difficult to control them on the base of traditional methods.In such condition it is necessary to use modern methods for solving these problems.One of such is global optimization algorithm based on mechanics of natural selection and natural genetics,which is called Genetic Algorithm. In this project the application problems of genetic algorithms for optimization problems , its specific characters and structures are given. The basic genetic operation : Selection, reproduction, crossover and mutation operations ae widely described the affectivity of genetic algorithms for optimization problem solving is shown. After the representation of optimization problem, structural optimization.

The practical application for selection , reproductio, crossover, and mutation are shown.Also the multi-objective optimization problem, some methods for global optimization are discussed.

TABLE OF CONTENS

INTRODUCTION

This is an introduction to genetic algorithm methods for optimization. Genetic Algorithms were formally introduced in the United States in the 1970s by John Holland at University of Michigan. The continuing price/performance improvements of computational systems has made them attractive for some types of optimization. In Particular, genetic algorithms work very well on mixed (continuous and discrete), Combinatorial problems. They are less susceptible to getting 'stuck' at local optima than gradient search methods. But they tend to be computationally expensive.

To use a genetic algorithm, you must represent a solution to your problem as a genome (or chromosome). The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s).

This presentation outlines some of the basics of genetic algorithms. The three most important aspects of using genetic algorithms are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and implementation of the genetic operators. Once these three have been defined, the generic genetic algorithm should work fairly well. Beyond that you can try many different variations to improve performance, find multiple optima (species - if they exist), or parallelize the algorithms.

The g enetic a lgorithm u ses st ochastic p rocesses, b ut t he r esult i s d istinctly n on random (better than random).

GENETIC Algorithms are used for a number of different application areas. An example of this would be multidimensional OPTIMIZATION problems in which the character s tring o f t he C HROMOSOME c an b e u sed t o e ncode t he v alues f or t he different parameters being optimized.

In p ractice, therefore, w e can implement this g enetic model of c omputation by having arrays of bits or characters to represent the Chromosomes. Simple bit manipulation operations allow the implementation of CROSSOVER, MUTATION and other operations. Although a substantial amount of research has been performed on variable- length strings and other structures, the majority of work with GENETIC Algorithms is focused on fixed-length character strings. We should focus on both this aspect of fixed-length ness and the need to encode the representation of the solution being sought as a character string, since these are crucial aspects that distinguish GENETIC PROGRAMMING, which does not have a fixed length representation and there is typically no encoding of the problem.

When the GENETIC ALGORITHM is implemented it is usually done in a manner that involves the following cycle: Evaluate the FITNESS of all of the Individuals in the POPULATION. Create a new population by performing operations such as CROSSOVER, fitness-proportionate REPRODUCTION and MUTATION on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population.

One iteration of this loop is referred to as a GENERATION. There is no theoretical reason for this as an implementation model. Indeed, we do not see this punctuated behavior in Populations in nature as a whole, but it is a convenient implementation model.

The first GENERATION (generation 0) of this process operates on a POPULATION of randomly generated Individuals. From there on, the genetic operations, in concert with the FITNESS measure, operate to improve the population.

History

Evolutionary computing was introduced in the 1960s by I. Rechenberg in his work "Evolution strategies" (Evolutionsstrategie in original). His idea was then developed by other researchers. Genetic Algorithms (GAs) were invented by John Holland and developed by him and his students and colleagues. This lead to Holland's book "Adaption in Natural and Artificial Systems" published in 1975.

In 1992 John Koza has used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP). LISP programs were used, because programs in this language can expressed in the form of a "parse tree", which is the object the GA works on.

CHAPTER ONE

WHAT ARE GENETIC ALGORITHMS (GAs) ?

1.1 The Genetic Algorithm and data mining

The Genetic Algorithm can be used to produce a variety of objects, as long as it is possible to somehow evaluate their quality (or "fitness"). More specifically, it is possible to build statistical predictors, not by regular computation from the data as in traditional statistics, but by evolving a predictor that is closest to (has the smallest deviation from) reality.

For classifiers, fitness is just the rate of correct predicted memberships on validation data. The genetic process favours emergence of those predictors that correctly predict the most examples;- they are fitter than inferior ones. No knowledge of statistics is necessary!

This process of generating useful knowledge from raw data is name induction.This belongs to the realm of data mining.

1.2 What the Genetic Algorithm is useful for ?

The Genetic Algorithm can solve problems that do not have a precisely-defined solving method, or if they do, when following the exact solving method would take far too much time. There are many such problems; actually, all still-open, interesting problems are like that.

Such problems are often characterised by multiple and complex, sometimes even contradictory constraints, that must be all satisfied at the same time. Examples are crew and team planning, delivery itineraries, finding the most beneficial locations for stores or warehouses, building statistical models, etc.

1.3 How the Genetic Algorithm works ?

The Genetic Algorithm works by creating many random "solutions" to the problem at hand.Being random, these starting "solutions" are not very good: schedules overlap and itineraries do not traverse every necessary location. This "population" of many solutions will then be subjected to an imitation of the evolution of species.

All of these solutions are coded the only way computers know: as a series of zeroes and ones. The evolution-like process consists in considering these 0s and 1s as genetic "chromosomes" that, like their real-life, biological equivalents, will be made to "mate" by hybridisation, also throwing in the occasional spontaneous mutation. The "offspring" generated will include some solutions that are better than the original, purely randomones.

The best offspring are added to the population while inferior ones are eliminated. By repeating this process among the better elements, repeated improvements will occur in the population, survive and generate their own offspring.

1.4 Why the Genetic Algorithm is a good idea ?

This crossover-then-selection process favours the apparition of better and bette solutions. By encouraging the best solutions generated and throwing away the worst ones ("only the fittest survive"), the original population keeps improving as a whole. This is called" selective pressure".

We are not actually calculating a solution to the problem being treated; we are merely selecting and encouraging the best emerging ones after certain, random operations. This is why we actually do not need to know how to solve the problem; we just have to be able to evaluate the quality of the generated solutions coming our way.

All we have to do to let a Genetic Algorithm solve our problem is write a "fitness function";- nothing else! This very surprising mechanism has been mathematically shown to eventually "converge" to the best possible solution. Of course, "eventually" comes much faster using skilfully written implementations.

The evolution and selection procedure is problem-independent; only the fitness function and one that decodes the chromosomes into a readable form are problem -specific. Once again, these functions do not require us to know how to solve the problem.

1.5 Why this technique is interesting ?

This is a rather brutal approach, requiring large amounts of processing power, but with the immense advantage of supplying solutions to things we don't know how to solve, or don't know how to solve quickly. For instance, what is the shortest path linking a number of cities ?

The only exact solution to this is to try them all and compare;- this will take geological time on any real-world problem. The Genetic Algorithm provides excellent, fast approximations to that.

No knowledge of how to solve the problem is needed: you only need to be able to assess the fitness of any given solution. This means implementation is easy and can rely on a problem-independent "engine", requiring little problem-related work.

The difference in computing speed between an ad hoc approach (specific, very fast) anda Genetic Algorithm-based solution (general but slower) is but one of the differences between the two approaches:

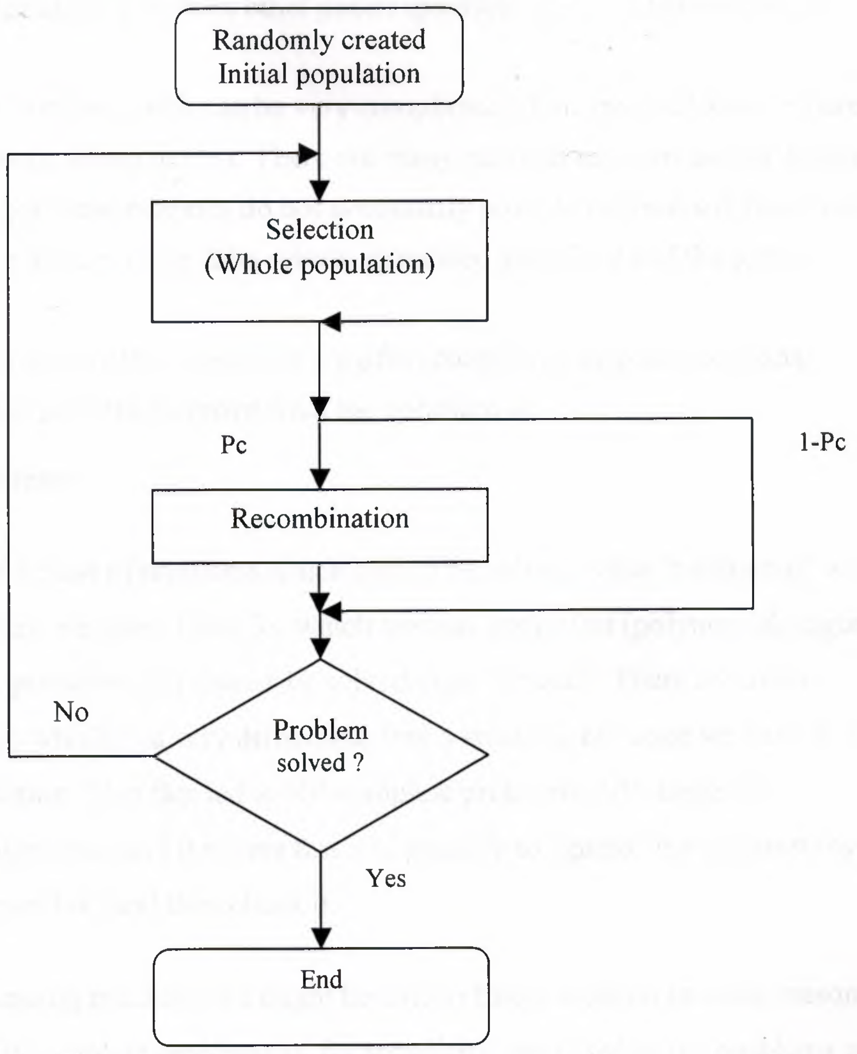
;	*Ad hoc* **approach (analytical, specific)**	**Genetic approach**
Speed	Depending on solution, generally good	Median or low
Performance	Depending on solution	Fair to excellent
Problem understanding	Necessary	Not necessary
Human work needed	A few minutes to a few theses	A few days
Applicability	Low: Most interesting problems have no usable mathematical expression, or are non-computable, or "NP-complete" (too many solutions to try them all)	General
Intermediary steps	are not solutions (you must wait until the end of computation)	are solutions (the solving process can be interrupted at any time, though the later the better)

1.6 Biology

Genetic algorithms are used in search and optimization, such as finding the maximum of a function over some domain space.

1. In contrast to deterministic methods like hill climbing or brute force complete enumeration, genetic algorithms use randomization.
2. Points in the domain space of the search, usually real numbers over some range, are encoded as bit strings, called chromosomes.
3. Each bit position in the string is called a gene.
4. Chromosomes may also be composed over some other alphabet than {0,1}, such as integers or real numbers, particularly if the search domain is multidimensional
5. GAs are called "blind" because they have no knowledge of the problem.

1.7 The Iteration Loop of a Basic Genetic Algorithm


Randomly created Initial population
Selection (Whole population)
Pc
1-Pc
Recombination
No
Problem solved ?
Yes
End

1.8 Search Space

If we are solving a problem, we are usually looking for some solution which will be the best among others. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called search space (also state space). Each point in the search space represents one possible solution. Each possible solution can be "marked" by its value (or fitness) for the problem. With GA we look for the best solution among among a number of possible solutions - represented by one point in the search space.

Looking for a solution is then equal to looking for some extreme value (minimum or maximum) in the search space. At times the search space may be well defined, but usually we know only a few points in the search space. In the process of using GA, the process of finding solutions generates other points (possible solutions) as evolution proceeds.

The problem is that the search can be very complicated. One may not know where to look for a solution or where to start. There are many methods one can use for finding a suitable solution, but these methods do not necessarily provide the best solution. Some of these methods are hill climbing, tabu search, simulated annealing and the genetic algorithm.

The solutions found by these methods are often considered as good solutions, because it is not often possible to prove what the optimum is.

1.8.1 NP-hard Problems

One example of a class of problems which cannot be solved in the "traditional" way, areNP problems. There are many tasks for which we may apply fast (polynomial) algorithms. There are also some problems that cannot be solved algorithmicall. There are many importantproblems in which it is very difficult to find a solution, but once we have it, it is easy to check thesolution. This fact led to NP-complete problems. NP stands for nondeterministic polynomial and it means that it is possible to "guess" the solution (by some nondeterministic algorithm) and then check it.

If we had a guessing machine, we might be able to find a solution in some reasonable time. Studying of NP-complete problems is, for simplicity, restricted to the problems where the answer can be yes or no. Because there are tasks with complicated outputs, a class of problems called NP-hard problems has been introduced. This class is not as limited as class of NP-complete problems. A characteristic of NP-problems is that a simple algorithm, perhaps obvious at a firstsight, can be used to find usable solutions.But this approach generally provides many possible solutions - just trying all possible solutions is very slow process (e.g. O(2^n)). For even slightly bigger instances of these type of problems this approach is not usable at all. Examples of the NP problems are satisfiability problem, travelling salesman problem or knapsack problem.

1.9 Operators of GA

1.9.1 Overview

The crossover and mutation are the most important parts of the genetic algorithm.The performance is influenced mainly by these two operators. Before we can explain more about crossover and mutation, some information about chromosomes will be given.

1.9.2 Encoding of a Chromosome

A chromosome should in some way contain information about solution that it represents.

The most used way of encoding is a binary string. A chromosome then could look like this:

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Each chromosome is represented by a binary string. Each bit in the string can represent some characteristics of the solution. Another possibility is that the whole string can represent a number - this has been used in the basic GA applet.

Of course, there are many other ways of encoding. The encoding depends mainly on the solved problem. For example, one can encode directly integer or real numbers, sometimes it is useful to encode some permutations and so on.

1.9.3Crossover

After we have decided what encoding we will use, we can proceed to crossover operation. Crossover operates on selected genes from parent chromosomes and creates newoffspring. The simplest way how to do that is to choose randomly some crossover point and copy everything before this point from the first parent and then copy everything after the crossover point from the other parent. Crossover can be illustrated as follows: (| is the crossover point):

Chromosome 1	11011 | 00100110110
Chromosome 2	11011 | 11000011110
Offspring 1	11011 | 11000011110
Offspring 2	11011 | 00100110110

There are other ways how to make crossover, for example we can choose more crossover points. Crossover can be quite complicated and depends mainly on the encoding ofchromosomes. Specific crossover made for a specific problem can improve performance of the genetic algorithm.

1.9.4 Mutation

After a crossover is performed, mutation takes place. Mutation is intended to prevent falling of all solutions in the population into a local optimum of the solved problem.

Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1.

Mutation can be then illustrated as follows:

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110110

The technique of mutation (as well as crossover) depends mainly on the encoding of chromosomes. For example when we are encoding permutations, mutation could be performed as an exchange of two genes.

1.9.5 Selection

A simple method of implementing fitness-proportionate selection is "Roulettewheel sampling", which is conceptually equivalent to giving each individual a slice of a roulette wheel equal in area to the individual's fitness. The wheel is spun and the ball comes to rest on the wedge shaped slice, and the corresponding individual is selected.

One of the most common methods is the binary tournament mating subset selection method. In this mating selection method, each chromosome in the population competes for a position in the mating subset. Two chromosomes are drawn at random from the population, the chromosome with the highest fitness score is placed in the mating subset.

Both chromosomes are returned to the population and another tournament begins. This procedure continues until the mating subset is full. A characteristic of this scheme is that the worst chromosome in the population will never be selected for inclusion in the mating subset.

1.9.6 Fitness Landscapes

The Basic Genetic Algorithm

Given a clearly defined problem to be solved and a bit string representation for candidate solutions , a simple genetic algorithm works as:

1. Start with a randomly generated population of n L-bit chromosomes(candidate solutions to a problem).OR the initial population of chromosomes is created by perturbing an input chromosome. How the initialization is done is not critical as long as the initial population spans a wide range of variable settings (i.e., has a diverse population). Thus, if we have explicit knowledge about the system being optimized that information can be included in the initial population.

2. In the second step, evaluation, the fitness f(x) of each chromosome x is computed. The goal of the fitness function is to numerically encode the performance of the chromosome. For real-world applications of optimization methods such as GAs the choice of the fitness function is the most critical step.

3. The third step is the exploitation or natural selection step. In this step, the chromosomes with the largest fitness scores are placed one or more times into a mating subset in a semi-random fashion. Chromosomes with low fitness scores are removed from the population. There are several methods for performing selection.

4. The fourth step, exploration, consists of the recombination and mutation operators.

Two chromosomes (parents) from the mating subset are randomly selected to be mated. The probability (Pc-Crossover Probability) that these chromosomes are recombined is a user-controlled option and is usually set to a high value (e.g., 0.95). If the parents are allowed to mate, a recombination operator is employed to exchange genes between the two parents to produce two children. If they are not allowed to mate, the parents are placed into the next generation unchanged.

The probability that a mutation will occur is another user-controlled option and is usually set to a low value (e.g., 0.01) so that good chromosomes are not destroyed. A mutation simply changes the value for a particular gene.

After the exploration step, the population is full of newly created chromosomes (children) and steps two through four are repeated. This process continues for a fixed number of generations.

1.10 Parameters of GA

1.10.1 Crossover and Mutation Probability

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability: how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!).

Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survive to next generation.

Mutation probability: how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.

Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

1.10.2 Other Parameters

There are also some other parameters of GA. One another particularly important parameter is population size.

Population size: how many chromosomes are in population (in one generation). If there are too few chromosomes, GA have few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to use very large populations because it does not solve the problem faster than moderate sized populations.

1.11 Encoding of GA

Introduction

Encoding of chromosomes is the first question to ask when starting to solve a problem with GA. Encoding depends on the problem heavily.

In this section some encodings will be introduced that have been already used with some success.

1.11.1 Binary Encoding

Binary encoding is the most common one, mainly because the first research of GA used this type of encoding and because of its relative simplicity.

In binary encoding, every chromosome is a string of bits - 0 or 1.

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Example of chromosomes with binary encoding

Binary encoding gives many possible chromosomes even with a small number of alleles.On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

1.11.2 Permutation Encoding

Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem.

In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Example of chromosomes with permutation encoding

Permutation encoding is useful for ordering problems. For some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have real sequence in it) for some problems.

1.11.3 Value Encoding

Direct value encoding can be used in problems where some more complicated values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.

In the value encoding, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, chars or any objects.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

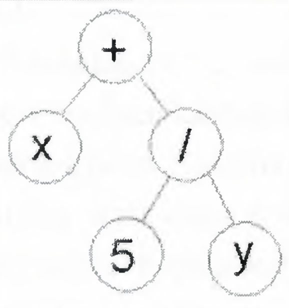
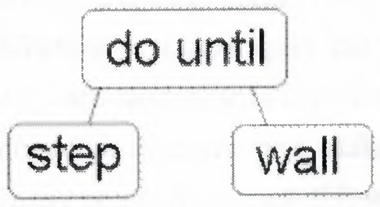
Example of chromosomes with value encoding

Value encoding is a good choice for some special problems. However, for this encoding it is often necessary to develop some new crossover and mutation specific for the problem.

1.11.4 Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, i.e. for genetic programming.

In the tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

Chromosome A	Chromosome B
	
(+ x (/ 5 y))	(do_until step wall)

Example of chromosomes with tree encoding

Tree encoding is useful for evolving programs or any other structures that can be encoded in trees. Programing language LISP is often used for this purpose, since programs in LISP are represented directly in the form of tree and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

1.12 Genetic Algorithms *Vs* Traditional methods

Traditional Methods

Calculus-based Search The main disadvantages of Calculus-based Search are, firstly,a tendency for the search to get trapped on local maxima - even a though a better solution may exist, all moves from the local maxima seem to decrease the fitness of the solution.

Secondly, the application of such searches depends on the existence of derivatives, for example the gradient of the graph under investigation.

Dynamic Programming This is a method for solving multi-step control problems, but can only be used where the overall fitness function is the sum of the fitness functions for each stage of the problem, and there is no interaction between stages.

Random Search This is a brute force approach to difficult functions, also called an enumerated search. Points in the search space are selected randomly. This is a very unintelligent strategy.

Gradient Methods Such methods are generally referred to as hill-climbing, and perform well on functions with only one peak. However, on functions with many peaks, the first peak found will be climbed, whether is it the highest peak or not, and no further program will be made.

Iterated Hillclimbing This is a combination of random search and gradient search. Once one peak has been located, the hillclimb is started again, but with another randomly chosen starting point. However, since each random trial is performed in isolation, no overall idea of the shape of the domain is obtained; also trials are randomly allocated over the entire search space and as many points in regions of low fitness will be evaluated as points in high fitness regions. A Genetic Algorithm, however, starts with an initial random population, and allocated more trials to regions of the search space found to have high fitness.

Simulated Annealing This was invented by Kirkpatrick in 1982, it is essentially a modified version of hill-climbing. Starting from a random point in the search space, a random move is made. If this move takes us to a higher point, it is accepted, otherwise it is accepted only with probability p(t), where t is time. The function p(t) begins close to 1, but gradually reduces towards zero - an analogy with the cooling of a solid.

Therefore, initially, any movesare accepted, but as the "temperature" reduces, the probability of accepting a negative move is lowered. Negative moves are essential sometimes, if local maxim are to be escaped, but too many negative moves would lead the search away from the maxima.

Simulated annealing deals with only one candidate at a time, so like random search, does not build an overall picture of the search space, and no information from previous moves is used to guide the selection of new moves. This technique has been successful in many applications, for example VLSI circuit layout.

1.13 Four differences that separate genetic algorithms from conventional optimization techniques:

Direct manipulation of a coding. Genetic algorithms manipulate decision or control variable representations at a string level to exploit similarities among high-performance strings. Other methods usually deal with functions and their control variables directly.

GAs deal with parameters of finite length, which are coded using a finite alphabet, rather than directly manipulating the parameters themselves. This means that the search is unconstrained neither by the continuity of the function under investigation, nor the existence of a derivative function. Moreover, by exploring similarities in coding, Gas can deal effectively with a broader class of functions than can many other procedures (see Building Block Hypothesis).

Evaluation of the performance of candidate solutions is found using objective, Payoff information. While this makes the search domain transparent to the algorithm and frees it from the constraint of having to use auxiliary or derivative information, it also means that there is an upper bound to its performance potential.

Search from a population, not a single point. In this way, GAs find safety in numbers. By maintaining a population of well-adapted sample points, the probability of reaching a false peak is reduced.

The search starts from a population of many points, rather than starting from just one point. This parallelism means that the search will not become trapped on a local maxima – especially if a measure of diversity-maintenance is incorporated into the algorithm, for then, one candidate may become trapped on a local maxima, but the need to maintain diverity in the search population means that other candidates will therefore avoid that particular area of the search space trapped on a local maxima - especially if a measure of diversity-maintenance is incorporated into the algorithm, for then, one candidate may become trapped on a local maxima, but the need to maintain diverity in the search population means that other candidates will therefore avoid that particular area of the search space.

Search via sampling, a blind search. GAs achieve much of their breadth by ignoring information except that concerning payoff. Other methods rely heavily on such information, and in problems where the necessary information is not available or difficult to obtain, these other techniques break down. GAs remain general by exploiting information available in any search problem. GAs process similarities in the underlying coding together with information ranking the structures according to their survival capability in the current environment. By exploiting such widely-available information, GAs may be applied to virtually any problem.

Search using stochastic operators, not deterministic rules. The transition rules used by genetic algorithms are probabilistic, not deterministic.

A distinction, however, exists between the randomised operators of GAs and other methods that are simple random walks. GAs use random choice to guide a highly exploitative search.

1.14 Advantages of using genetic algorithms

· They require no knowledge or gradient information about the response surface

· Discontinuities present on the response surface have little effect on overall optimization performance

· They are resistant to becoming trapped in local optima

· They perform very well for large-scale optimization problems

· Can be employed for a wide variety of optimization problems

1.15 Disadvantages of using genetic algorithms

· Have trouble finding the exact global optimum

· Require large number of response (fitness) function evaluations

· Configuration is not straightforward

CHAPTER TWO

OPTIMIZATION PROBLEM

2.1 Definition for Optimization

A series of operations that can be performed periodically to keep a computer in optimum shape. Optimization is done by running a maintenance check, scanning for viruses, and defragmenting the hard disk. Norton Utilities is one program used for optimizing.

2.1.1 Optimization problems are made up of three basic ingredients:

- An objective function which we want to minimize or maximize. For instance, in a manufacturing process, we might want to maximize the profit or minimize the cost. In fitting experimental data to a user-defined model, we might minimize the total deviation of observed data from predictions based on the model. In designing an automobile panel, we might want to maximize the strength.
- A set of unknowns or variables which affect the value of the objective function. In the manufacturing problem, the variables might include the amounts of different resources used or the time spent on each activity. In fitting-the-data problem, the unknowns are the parameters that define the model. In the panel design problem, the variables used define the shape and dimensions of the panel.
- A set of constraints that allow the unknowns to take on certain values but exclude others. For the manufacturing problem, it does not make sense to spend a negative amount of time on any activity, so we constrain all the "time" variables to be non-negative. In the panel design problem, we would probably want to limit the weight of the product and to constrain its shape.

Find values of the variables that minimize or maximize the objective function while satisfying the constraints.

2.1.2 Are All these ingredients necessary?

Objective Function

Almost all optimization problems have a single objective function. (When they don't they can often be reformulated so that they do!) The two interesting exceptions are:

- *No objective function.* In some cases (for example, design of integrated circuit layouts), the goal is to find a set of variables that satisfies the constraints of the model. The user does not particularly want to optimize anything so there is no reason to define an objective function. This type of problems is usually called a *feasibility problem.*

- *Multiple objective functions.* Often, the user would actually like to optimize a number of different objectives at once. For instance, in the panel design problem, it would be nice to minimize weight and maximize strength simultaneously. Usually, the different objectives are not compatible; the variables that optimize one objective may be far from optimal for the others. In practice, problems with multiple objectives are reformulated as single-objective problems by either forming a weighted combination of the different objectives or else replacing some of the objectives by constraints. These approaches and others are described in our section on multi-objective optimization.

Variables

These are essential. If there are no variables, we cannot define the objective function and the problem constraints.

Constraints

Constraints are not essential. In fact, the field of unconstrained optimization is a large and important one for which a lot of algorithms and software are available. It's been argued that almost all problems really do have constraints. For example, any variable denoting the "number of objects" in a system can only be useful if it is less than the number of elementary particles in the known universe! In practice though, answers that make good sense in terms of the underlying physical or economic problem can often be obtained without putting constraints on the variables.

2.2 The Optimization Problem

The gradient-based optimization algorithms most often used with structural equation models (Levenberg-Marquardt, Newton-Raphson, quasi-Newton) are inadequate because they too often fail to find the global maximum of the likelihood function. The discrepancy function is not globally convex. Multiple, local minima and saddle points often exist, so that there is no guarantee that gradient-based methods will converge to the global maximum. Indeed, saddle points and other complexities in the curvature of the likelihood function can make it difficult for gradient-based optimization methods to find any maximum at all. Such difficulties are intrinsic to linear structure models for two reasons. First, the LISREL likelihood is not globally concave. Second,

linear structure models' identification conditions do not require and do not guarantee that the model (as a function of the data) will determine a unique set of parameter values outside a neighborhood of the true values. The derivatives of the likelihood function with respect to the parameters are not well defined outside of the neighborhood of the solution.

Therefore, outside of the neighborhood of the solution, derivative based methods often have little or no information upon which to advance to the global maximum.

Bootstrap methodology accentuates optimization difficulties, because the bootstrap resampling distribution draws from the entire distribution of the parameter estimates. Even if optimization in the original sample is not problematic, one can expect to encounter difficulties in a significant number of bootstrap resamples. Even if the model being estimated is correctly specified, problematic resamples contain crucial information about the tails of the distribution of the parameter estimates. Indeed, what
otstrap methods primarily do is make corrections for skewness-for asymmetry between the tails of the distribution of each parameter estimate—that is ignored by normal-theory confidence limit estimates. Tossing out the tail information basically defeats the purpose of using the bootstrap to improve estimated confidence intervals. In general, any procedure of replacing problematic resamples with new resampling draws until optimization is easy must fail, as making such replacements would induce incomplete coverage of the parameter estimates[1] sampling distribution and therefore incorrect inferences. Ichikawa and Konishi (1995) make this mistake.

Because the nonexistence of good MLEs in bootstrap resamples is evidence of misspecification and because the occurrence of failures affects the coverage of the bootstrap confidence intervals, it is crucial to use an optimization method that finds the global minimum of the discrepancy function if one exists. In order to overcome the problems of local minima and nonconvergence from poor starting values., GENBLIS combines a gradient-based method with an evolutionary programming (EP) algorithm. Our EP algorithm uses a collection of random and homotopy search operators that combine members of a population of candidate solutions to produce a population that on average better fits the current data. Nix and Vose (1992; Vose 1993) prove that genetic algorithms are asymptotically correct, in the sense that the probability of converging to the b est p ossible population o f candidate solutions goes t o o ne as t he population size i ncreases t o i nfinity. Because they have a similar Markov chain structure, EP algorithms of the kind we use are asymptotically correct in the same sense. For a linear structure model and a data set for which a good MLE (global minimum) exists, the best possible population is the one in which all but a small fraction of the candidate solutions have that value. A fraction of the population will have different values because the algorithm must i nclude certain random variations in order to have effective global search properties. The probability of not finding a good MLE when one exists can be made arbitrarily small by increasing the population size used in the algorithm.

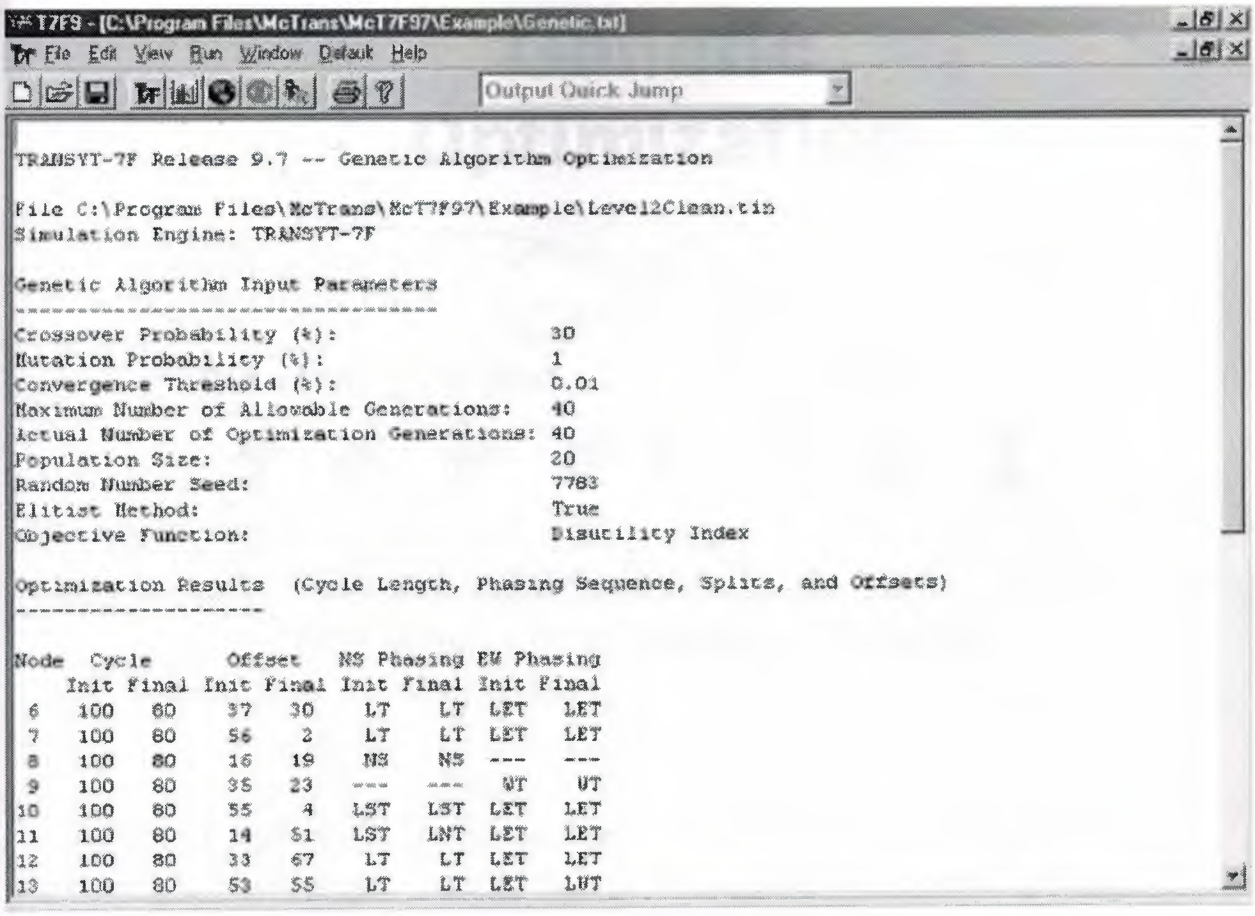
The EP is very good at finding a neighborhood of the global minimum in which the discrepancy function is convex. But the search operators, which do not use derivatives, are quite slow at getting from an arbitrary point in that neighborhood to the global minimum value. We add the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton optimizer as an operator to do the final hill-climbing. We developed and implemented a general form of this EP-BFGS algorithm in a C program called Genetic Optimization Using Derivatives (GENOUD). GENBLIS is a version of GENOUD specifically tuned to estimate linear structure models.

In our experience the program finds the global minimum solution for the LISREL estimation problem in all cases where the most widely used software fails, except where extensive examination suggests that a solution does not exist.

2.3 Genetic Algorithm Optimization

Genetic algorithm optimization is a theoretical improvement over the traditional hill-climb optimization technique that has been employed by TRANSYT-7F for many years, The genetic algorithm has the ability to avoid becoming trapped in a "local optimum" solution, and is mathematically best qualified to locate the "global optimum" solution.

Releases 9 features genetic algorithm optimization of offsets and yields points, using either TRANSYT-7F or CORSIM as the simulation engine. Phasing sequence optimization was introduced in release 9.4 (January 2002), and requires TRANSYT-7F' as the simulation engine. Genetic algorithm optimization of cycle length and splits was introduced in release 9.6 (September 2002), and also requires TRANSYT-7F as the simulation engine,


T7F9 - [C:\Program Files\McTrans\McT7F97\Example\Genetic.txt]
File Edit View Run Window Default Help
Output Quick Jump

```
TRANSYT-7F Release 9.7 -- Genetic Algorithm Optimization

File C:\Program Files\McTrans\McT7F97\Example\Level2Clean.tin
Simulation Engine: TRANSYT-7F

Genetic Algorithm Input Parameters
----------------------------------
Crossover Probability (%):                   30
Mutation Probability (%):                    1
Convergence Threshold (%):                   0.01
Maximum Number of Allowable Generations:     40
Actual Number of Optimization Generations:   40
Population Size:                             20
Random Number Seed:                          7783
Elitist Method:                              True
Objective Function:                          Disutility Index

Optimization Results  (Cycle Length, Phasing Sequence, Splits, and Offsets)
--------------------

Node  Cycle       Offset    NS Phasing EW Phasing
    Init Final Init Final Init Final Init Final
 6   100   60    37   30    LT    LT  LET   LET
 7   100   80    56    2    LT    LT  LET   LET
 8   100   80    16   19    NS    NS  ---   ---
 9   100   80    35   23   ---   ---   UT    UT
10   100   60    55    4   LST   LST  LET   LET
11   100   80    14   51   LST   LNT  LET   LET
12   100   80    33   67    LT    LT  LET   LET
13   100   80    53   55    LT    LT  LET   LUT
```

For years, signal timing designers have known that they could often come up with a better control plan by making minor modifications to the so-called "optimal result"recommended by a computer program. This is a byproduct of the hill-climb optimization process, where most timing plan candidates are not examined, in an effort to save time. Unfortunately, the global optimum solution is often skipped over during the hill-climb optimization process.

Fortunately, with genetic algorithm optimization, the user may have a much more difficult time in coming up with a better solution than the computer program. The genetic algorithm does not examine every single timing plan candidate either, but is a random guided search, capable of intelligently tracking down the global optimum solution. As with the human race, the weakest candidates are eliminated from the gene pool, and each successive generation of individuals contains stronger and stronger characteristics. It's survival of the fittest, and the unique processes of crossover and mutation conspire to keep the species as strong as possible.

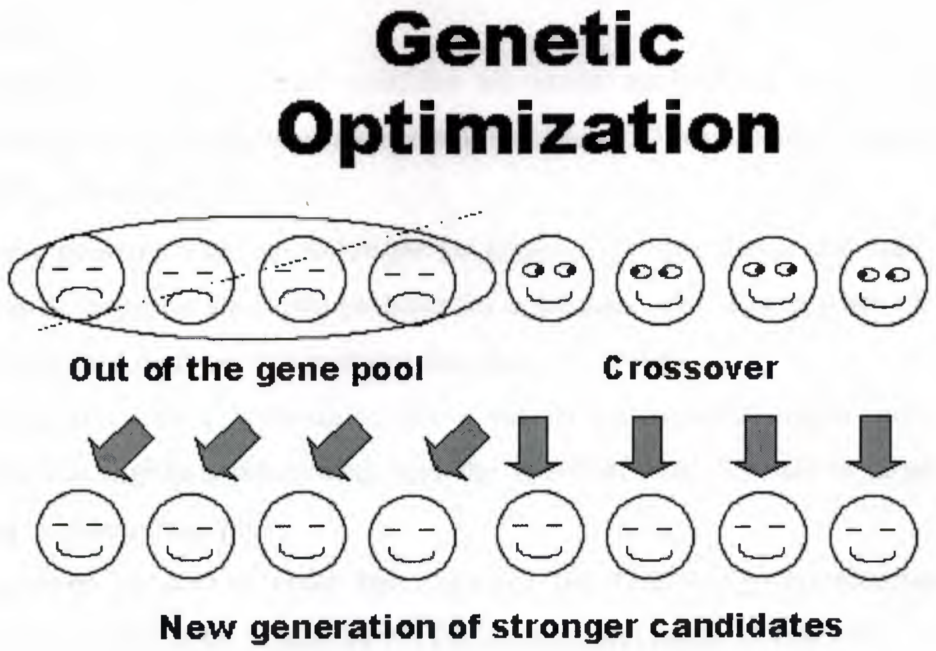


Figure 2.3

Although it produces the best timing plans, a potential drawback of the genetic algorithm is increased program running times on the computer when optimizing large networks. Fortunately the drawback of increased program running times continues to be minimized by the ever-increasing processing speeds of today's computers. In addition, the TRANSYT-7F electronic Help system offers practical suggestions for reducing the genetic algorithm running times associated with large traffic networks.

2.4 Continuous Optimization

2.4.1 Constrained Optimization

CO is an applications module written in the GAUSS programming language. It solves the Nonlinear Programming problem, subject to general constraints on the parameters - linear or nonlinear, equality or inequality, using the Sequential Quadratic Programming method in combination with several descent methods selectable by the user - Newton-Raphson, quasi-Newton (BFGS and DFP), and scaled quasi-Newton. There are also several selectable line search methods. A Trust Region method is also available which prevents saddle point solutions. Gradients can be user-provided or numerically calculated.

CO is fast and can handle large, time-consuming problems because it takes advantage of the speed and number-crunching capabilities of GAUSS. It is thus ideal for large scale Monte Carlo or bootstrap simulations.

Example

A Markowitz mean/variance portfolio allocation analysis on a thousand or more securities would be an example of a large-scale problem CO could handle (about 20 minutes on a 133 Mhz Pentium-based PC).

CO also contains a special technique for semi-definite problems, and thus it will solve the Markowitz portfolio allocation problem for a thousand stocks even when the covariance matrix is computed on fewer observations than there are securities.

In summary, CO is well-suited for a variety of financial applications from the ordinary to the highly sophisticated, and the speed of GAUSS makes large and time-consuming problems feasible.

CO comes as source code and requires the GAUSS programming language software. It is available for Windows 95, OS/2, DOS, and major UNIX platforms.

Available for both PC and UNIX system versions of GAUSS and GAUSS Light, CO is an advanced GAUSS Application.

GAUSS Applications are modules written in GAUSS for performing specific modeling and analysis tasks. They are designed to minimize or eliminate the need for user programming while maintaining flexibility for non-standard problems.

CO requires GAUSS version 3.2.19 or higher (3.2.15+ for DOS version). GAUSS program source code is included.

CO is available for DOS, OS/2, Windows NT, Windows 95, and UNIX versions of GAUSS.

2.4.2 Unconstrained optimization

The unconstrained optimization problem is central to the development of optimization software. Constrained optimization algorithms are often extensions of unconstrained algorithms, while nonlinear least squares and nonlinear equation algorithms tend to be specializations. In the unconstrained optimization problem, we seek a local minimizer of a real-valued function, f(x), where x is a vector of real variables. In other words, we seek a vector, x*, such that f(x*) <= f(x) for all x close to x*.

Global optimization algorithms try to find an x* that minimizes f over all possible vectors x. This is a much harder problem to solve. We do not discuss it here because, at present, no efficient algorithm is known for performing this task. For many applications, local minima are good enough, particularly when the user can draw on his/her own experience and provide a good starting point for the algorithm.

Newton's method gives rise to a wide and important class of algorithms that require computation of the gradient vector $\nabla f(x)=(\partial_i f(x))$

And the Hessian matrix, $\nabla^2 f(x)=(\partial_j \partial_i(x))$.

Although the computation or approximation of the Hessian can be a time-consuming operation, there are many problems for which this computation is justified.We describe algorithms in which the user supplies the Hessian explicitly before moving on to a discussion of algorithms that don't require the Hessian.Newton's method forms a quadratic model of the objective function around thecurrent iterate x_k. The model function is defined :

$$q_k(\partial)=f(x_k)+\nabla f(x_k)^T \partial+1/2\partial^T \nabla^2 f(x_k)\partial.$$

In the basic Newton method, the next iterate is obtained from the minimizer of q_k. When the Hessian matrix, $\nabla^2 f(x_k)$, is positive definite, the quadratic model has a unique minimizer that can be obtained by solving the symmetric $m\times n$ linear system:

$\nabla^2 f(x_k)\,\partial_k = -\nabla f(x_k)$.The next iterate is then $x_k+1=x_k+\partial_k$

Convergence is guaranteed if the starting point is sufficiently close to a local minimizer x* at which the Hessian is positive definite. Moreover, the rate of convergence is quadratic, that is, $\| x_k+1-x^* \| \leq \beta \| x_k-x^* \|^2$,for some positive constant β.

In most circumstances, however, the basic Newton method has to be modified to achieve convergence. Versions of Newton's method are implemented in the following software packages:

BTN, GAUSS, IMSL, LANCELOT, NAG, OPTIMA, PORT 3, PROC NLP, TENMIN, TN, TNPACK, UNCMIN, and VE08.

The NEOS Server also has an unconstrained minimization facility to solve these problems remotely over the Internet.

These codes obtain convergence when the starting point is not close to a minimize. by using either a line-search or a trust-region approach.

The line-search variant modifies the search direction to obtain another a downhill, or descent direction for f. It then tries different step lengths along this direction until it finds a step that not only decreases f, but also achieves at least a small fraction of this direction's potential.

The trust-region variant uses the original quadratic model function, but they constrain the new iterate to stay in a local neighborhood of the current iterate. To find the step, then, we have to minimize the quadratic subject to staying in this neighborhood, which is generally ellipsoidal in shape.

Line-search and trust-region techniques are suitable if the number of variables n is not too large, because the cost per iteration is of order n^3. Codes for problems with a large number of variables tend to use truncated Newton methods , which usually settle for an approximate minimizer of the quadratic model.

So far, we have assumed that the Hessian matrix is available, but the algorithms are unchanged if the Hessian matrix is replaced by a reasonable approximation. Two kinds of methods use approximate Hessians in place of the real thing: The first possibility is to use difference approximations to the exact Hessian. We exploit the fact that each column of the Hessian can be approximated by taking the difference between two instances of the gradient vector evaluated at two nearby points. For sparse Hessians, we can often approximate many columns of the Hessian with a single gradient evaluation by choosing the evaluation points judiciously.

Quasi-Newton Methods buildup an approximation to the Hessian by keeping track of the gradient differences along each step taken by the algorithm. Various conditions are imposed on the approximate Hessian. For example, its behavior along the step just taken is forced to mimic the behavior of the exact Hessian, and it is usually kept positive definite.

Finally, we mention two other approaches for unconstrained problems that are not so closely related to Newton's method:

Nonlinear conjugate gradient methods are motivated by the success of the linear conjugate gradient method in minimizing quadratic functions with positive definite Hessians. They use search directions that combine the negative gradient direction with another direction, chosen so that the search will take place along a direction not previously explored by the algorithm. At least, this property holds for the quadratic case, for which the minimizer is found exactly within just n iterations. For nonlinear problems, performace is problematic, but these methods do have the advantage that they require only gradient evaluations and do not use much storage.

The nonlinear Simplex method (not to be confused with the simplex method for linear programming) requires neither gradient nor Hessian evaluations. Instead, it performs a pattern search based only on function values. Because it makes little use of information about f, it typically requires a great many iterations to find a solution that is even in the ballpark. It can be useful when f is nonsmooth or when derivatives are impossible to find, but it is unfortunately often used when one of the algorithms above would be more appropriate.

2.4.2.1 Systems of Nonlinear Equations

Systems of nonlinear equations arise as constraints in optimization problems, but also arise, for example, when differential and integral equations are discretized. In solving a system of nonlinear equations, we seek a vector such that f(x)=0 where x is a n n-dimensional of n variables. Most algorithms in this section are closely related to algorithms for unconstrained optimization and nonlinear least squares. Indeed, algorithms for systems of nonlinear equations usually proceed by seeking a local minimizer to the problem

$\min\{\| f(x)\|: x \in R^n\}$ for some norm $\|.\|$,usually the 2-norm. This strategy is reasonable, since any solution of the nonlinear equations is a global solution of the minimization problem.

Of linear equations $f'(x_k)\partial_k = -f(x_k)$,(1.1)

Newton's method, modified and enhanced, forms the basis for most of the software used to solve systems of nonlinear equations. Given an iterate, Newton's method computes f(x) and its Jacobian matrix, finds a step by solving the system of linear and then sets $x_k+1=x_k+\partial_k$.

Most of the computational cost of Newton's method is associated with two operations: evaluation of the function and the Jacobian matrix, and the solution of the linear system (1.1). Since the Jacobian is $f'(x)=(\partial_1 f(x),...., \partial_n f(x))$,

The computation of the *i*th column requires the partial derivative of f with respect to the *i*th variable, while the solution of the linear system (1.1) requires order n^3 when the Jacobian is dense.

Convergence of Newton's method is guaranteed if the starting is sufficiently close to the solution and the Jacobian at the solution is nonsingular. Under these conditions the rate of convergence is quadratic; that is, $\| x_k+1-x^* \| \leq \beta \| x_k - x^* \|2$,for some positive constant . This rapid local convergence is the main advantage of Newton's method. The disadvantages include the need to calculate the Jacobian matrix and the lack of guaranteed global convergence; that is, convergence from remote starting points.

The following software attempts to overcome these two disadvantages of Newton's method by allowing approximations to be used in place of the exact Jacobian matrix and by using two basic strategies-trust region and line search-to improve global convergence behavior: GAUSS , IMSL , LANCELOT , MATLAB , MINPACK-1 , NAG(FORTRAN) , NAG(C) , NITSOL , and OPTIMA .

2.5 Global Optimization (GO)

A globally optimal solution is one where there are no other feasible solutions with better objective function values. A locally optimal solution is one where there are no other feasible solutions "in the vicinity" with better objective fiinction values. You can picture this as a point at the top of a "peak" or at the bottom of a "valley" which may be formed by the objective fiinction and/or the constraints — but there may be a higher peak or a deeper valley far away from the current point.

In certain types of problems, a locally optimal solution is also globally optimal. These include LP problems; QP problems where the objective is positive definite (if minimizing; negative definite if maximizing); and NLP problems where the objective is a convex fiinction (if minimizing; concave if maximizing) and the constraints form a convex set. But most nonlinear problems are likely to have multiple locally optimal solutions.

Global optimization seeks to find the globally optimal solution. GO problems are' intrinsically very difficult to solve; based on both theoretical analysis and practical experience, you should expect the time required to solve a GO problem to increase rapidly — perhaps exponentially — with the number of variables and constraints.

2.5.1 Problem Formulation

Many important practical problems can be posed as mathematical programming problems.

This has been internationally appreciated since 1944 and has lead to major research activities in many countries, in all of which the aim has been to write efficient computer programs to solve subclasses of this problem. An important subclass that has proved very difficult to solve occurs in many practical engineering applications. Let us consider the design of a system that has to meet certain design criterion. The system will include features that may be varied by the designer within certain limits. The values given to these features will be the optimization variables of the problem. Frequently when the system performance is expressed as a mathematical function of the optimization variables, this function, which will sometimes be called the objective function, is not convex and possesses more than one local minimum. The problem of writing computer algorithms that distinguish between these local minima and locate the best local minimum is known as the global optimization problem.

2.5.2 Some Classes of Problems

Let the features that may be varied be represented by the vector $x=(x_1,\dots,x_n)$ of reals. Let A be the feasible region in which the global minimum of f(x) is to be estimated.

Combinatorial problems are those that have a finite but large number of feasible points in A .

Constrained Global Optimization Problems are those for which the border of A comes into play. A subset of these problems are those for which f(x) and the constraints have a special form, eg. quadratic. Such problems are treated in [Pardalos and Rosen 1978] and will not be dealt with here.

Essentially Unconstrained Global Optimization Problems are those for which the global minimum is in the interior of A. We will here concentrate on these, see [Törn and Zilinskas 1987]. Normally A will be a box giving the limits for each feature.

Only essential local minima are considered, i.e., those having a sub-optimal surrounding of positive measure. A solution to the problem will of course not be the global minimum f^* exactly, but for instance a value less than f^*_e, where $f^*_e = f^*$ +epsilon.

The problem to determine x^*, where $f^* = f(x^*)$ is not a properly posed problem, i.e., there exist continuous functions in A with maximum absolute difference in function values over A arbitrary small but with global optima wide apart.

2.5.3 Complexity of Problems

2.5.3.1Complexity Measures

Methods for generally solving such essentially unconstrained problems are those containing some technique, which explores the search region A by evaluating f for points spread out in A (eg. random sampling) and then use some local technique to possibly find the global minimum.

We postulate that the complexity in solving such problems is dependent on the following features of the problem:
The size p^* of the region of attraction of the global minimum in relation to A.
The affordable number of function evaluations N_f. Embedded or isolated global minimizers. The number of local minimizers.

2.5.3.2 The size p^* of the region of attraction of the global minimum in relation to A.

The region of attraction of a local minimizer $S(x_m)$ is defined as the largest region containing x_m, such that when starting an infinitely small step strictly descending local minimization from any point in $S(x_m)$, then the minimizer x_m will be found each time. The region of attraction of of a minimum m is the union of regions of attraction of all minimizers x for which f(x) = m.

If the region of attraction of f^* is large then this region is easy to detect when sampling in A and such a problem is of course easier to solve than a problem with smaller such region.

2.5.3.3The affordable number of function evaluations N_f

The value of the expression $(1-p^*)^{N_f}$ is the chance to miss the region of attraction of f^* when sampling N_f points at random in A. If the function f is cheap to evaluate then N_f is large and and the probability that even a very small region of attraction is missed becomes small. However, if only a small number of function evaluations can be performed then the probability to miss the region of attraction of the global minimum even for large p^* is large.

2.5.4 Elements of Global Optimization Methods

All methods will evaluate the function f(x) in some points $x_1,\ldots,x_N$ in *A*, and they differ only in their choice of these points.

2.5.4.1 Strategies in choosing points.

Because we do not have any information about where in A to find the global minimum, one strategy that must be used is to spread out some of the points to cover A. We call any realization of this strategy a global technique. A possible global technique is uniform sampling. Any serious GO method must use some global technique.

Given a point x it is normally possible to find a nearby point with a smaller function value. We call any technique realizing this a local technique. A possible local technique is local optimization. Any serious GO method will use local optimization, at least to improve upon the estimates of the global minimum found.

A special local technique is to adapt the probes in the global technique so that more effort is put on sampling in regions where relatively small function values are found. We call this technique adaption. Adaption can range from no adaption, i.e., global technique, to extreme adaption, i.e., local technique. A GO method can be based exclusively on successive adaption.

2.5.4.2 Stopping Conditions and Solvability

In any computer algorithm there must be some stopping condition, which after some finite number of computing steps stops the computation. The condition should of course relate to the quality of the solution achieved.

This is a very crucial point in global optimization. Without some additional information or assumptions about the problem there is no way to decide on the quality of the solution after a given number of steps (eg. number of function evaluations made).

A necessary condition for solving the problem is that at least one of the points $x_1,\ldots,x_N$ is in the region of attraction of the global minimum S(x<SUP*< sup>). If no information about the size of S(x<SUP*< sup>) (eg. lower bound) is known then of course there is no hope to formulate a proper stopping condition. It is of course not possible for the algorithm to decide on the proper stopping condition based on the function values $f(x_1),...,f(x_N)$.

This means that either additional information or assumptions must be utilized for establishing a proper stopping condition or else the stopping condition is more or less heuristic.

From this we may conclude that the global optimization problem in general is unsolvable and that we must be prepared to accept non-global minima as solutions.

2.5.4.3 Convergence with Probability One

Because convergence is not in general possible, we may then lower our ambition and only require convergence with probability 1. This is of course possible only for algorithms that can be made to run forever.

This seems to be a modest requirement for any serious algorithm. However, the requirement is very weak and is not of very much use in practice, which can be seen from the following reasoning.

It has been pointed out that some methods do not have any theoretical convergence properties (eg. Price's CSR) and they are therefore considered inferior to other methods. Assume that we have a method that can be made to run forever. Add the following: For i= 1,2,3,... when N reaches $10iN_f$ sample a point at random in A to be a candidate to include in the converging set of CSR.

The modified method will converge with probability 1, but because N_f is the maximal affordable number of function evaluations, the two algorithms will give equivalent results in any real application.

2.5.4.4 Comparing Methods

It should be clear that a comparison of methods must be based on empirical evaluation rather than on theoretical.

Assume that we apply a probabilistic method M to a problem P. We can see this as a mapping from (M,P) --> (E,m,q), where E is the effort applied and q is the probability that some minimum m is reached.

Assume that two methods are applied and that the same minimum is achieved. This gives

(M1,P) --->(E1,m,q1)

(M2,P) --->(E2,m,q2)

If Ei < Ej and qi > qj then obviously Mi is better than Mj for this problem.

However, if we have that Ei < Ej and qi < qj then neither dominates the other and no conclusion about which is better can be made.

Because the results may vary depending on P and the levels of E and q a conclusive decision about the superiority of one method over another needs a lot of computations.

2.5.5 Solving GO Problems

Multistart methods are a popular way to seek globally optimal solutions with the aid of a "c lassical" s mooth n onlinear s olver (that b y i tself f inds o nly l ocally o ptimal s olutions). The basic idea behind these methods is to automatically start the nonlinear Solver from randomly selected starting points, reaching different locally optimal solutions, then select the best of these as the proposed globally optimal solution. Multistart methods have a limited guarantee that (given certain assumptions about the problem) they will "converge in probability" to a globally optimal solution. This means that as the number of runs of the nonlinear Solver increases, the probability that the globally optimal solution has been found also increases towards 100%.

Where Multistart methods rely on random sampling of starting points, Continuous Branch and Bound methods are designed to systematically subdivide the feasible region into successively smaller subregions, and find locally optimal solutions in each subregion. The best of the locally optimally solutions is proposed as the globally optimal solution. Continuous Branch and Bound methods have a theoretical guarantee of convergence to the globally optimal solution, but this guarantee usually cannot be realized in a reasonable amount of computing time, for problems of more than a small number of variables. Hence many Continuous Branch and Bound methods also use some kind of random or statistical sampling to improve performance.

Genetic Algorithms, Tabu Search and Scatter Search are designed to find "good" solutions to nonsmooth optimization problems, but they can also be applied to smooth nonlinear problems to seek a globally optimal solution. They are often effective at finding better solutions than a "classic" smooth nonlinear solver alone, but they usually take much more computing time, and they offer no guarantees of convergence, or tests for having reached the globally optimal solution.

2.6 Smooth Nonlinear Optimization (NLP) Problems

A smooth nonlinear programming (NLP) or nonlinear optimization problem is one in which the objective or at least one of the constraints is a smooth nonlinear function of the decision variables. An example of a smooth nonlinear function is:

$$2\,x1^2 + x2^3 + \log x3$$

where x1, x2 and x3 are decision variables. A quadratic programming (QP) problem is a special case of a smooth nonlinear optimization problem, but it is usually solved by specialized, more efficient methods.

Nonlinear functions, unlike linear functions, may involve variables that are raised to a power or multiplied or divided by other variables. They may also use transcendental functions such as exp, log, sine and cosine.

NLP problems and their solution methods require nonlinear functions that are continuous, and (usually) further require functions that are smooth -- which means that derivatives of these functions with respect to each decision variable, i.e. the function gradients, are continuous.

2.6.1 Solving NLP Problems

There are a variety of methods for solving NLP problems, and no single method is best for all problems. Two of the most widely used and effective methods, used by Frontline's solvers, are the Generalized Reduced Gradient (GRG) and Sequential Quadratic Programming (SQP) methods. Interior Point or Newton-Barrier methods are also being adapted to solve at least some NLP problems.

NLP solvers generally exploit the smoothness of the problem functions by computing gradient values at various trial solutions, and moving in the direction of the negative gradient (when minimizing; the positive gradient when maximizing). They usually also exploit second derivative information to follow the curvature as well as the direction of the problem functions. To solve constrained problems, NLP solvers must take into account feasibility and the direction and curvature of the constraints as well as the objective.

As noted above, NLP solvers normally can find only a locally optimal solution, in the vicinity of the starting point of the optimization given by the user. It is frequently possible, but considerably more difficult, to find the globally optimal solution.

2.7 Nonsmooth Optimization (NSP)

The most difficult type of optimization problem to solve is a nonsmooth problem (NSP). Such a problem may not only have multiple feasible regions and multiple locally optimal points within each region -- because some of the functions are non-smooth or even discontinuous, derivative or gradient information generally cannot be used to determine the direction in which the function is increasing (or decreasing). In other words, the situation at one possible solution gives very little information about where to look for a better solution.

In all but the simplest problems, it is impractical to exhaustively enumerate all of the possible solutions and pick the best one, even on a fast computer. Hence, most methods rely on some sort of random sampling of possible solutions. Such methods ar nondeterministic or stochastic -- they may yield different solutions on different runs, even when started from the same point on the same model, depending on which points are randomly sampled.

2.7.1 Solving NSP Problems

Genetic or Evolutionary Algorithms offer one way to find "good" solutions to nonsmooth optimization problems. (In a genetic algorithm the problem is encoded in a series of bit strings that are manipulated by the algorithm; in an "evolutionary algorithm," the decision variables and problem functions are used directly. Most commercial Solver products are based on evolutionary algorithms.)

These algorithms maintain a population of candidate solutions, rather than a single best solution so far. From existing candidate solutions, they generate new solutions through either random mutation of single points or crossover or recombination of two or more existing points. The population is then subject to selection that tends to eliminate the worst candidate solutions and keep the best ones. This process is repeated, generating better and better solutions; however, there is no way for these methods to determine that a given solution is truly optimal.

Tabu Search and Scatter Search offer another approach to find "good" solutions to nonsmooth optimization problems. These algorithms also maintain a population of candidate solutions, rather than a single best solution so far, and they generate new solutions from old ones. However, they rely less on random selection and more on deterministic methods. Tabu search uses memory of past search results to guide the direction and intensity of future searches. These methods generate successively better solutions, but as with genetic and evolutionary algorithms, there is no way for these methods to determine that a given solution is truly optimal.

2.8 Multi-Objective Optimization for Highway Management Programming

Highway infrastructure is a major national investment, and a well-managed highway network forms an integral element of a sustainable economy. An ideal management program for a highway network is one that would maintain all highway sections at a sufficiently high level of service and structural condition, but requires only a reasonably low budget and use of resources, without creating any significant adverse impact on the environment, safe traffic operations, and social and community activities. Unfortunately, many of these are conflicting requirements. For instance, more resources and higher budgets may be needed if the highways are to be maintained at a high state of operability. But this could lead to pavement activities causing longer traffic delays, increased pollution, more disruption of social activities, and inconvenience to the community. Therefore, the decision processes involved in highway management activities requires a multi-objective consideration that addresses the competing requirements of different objectives.

Practically all the pavement management programming tools in use currently are based on single-objective optimization. In single-objective analysis, the requirements hich are not incorporated into the objective function are imposed as constraints in the formulation. This can be viewed as an interference of the optimization process which artificially sets limits to selected problem parameters. As a result, the solutions obtained from single-objective analysis are sub-optimal with respect to one's derived from multi-objective formulations.

A genetic-algorithm (GA) based formulation for multi-objective programming of highway management activities has been developed at the Centre for Transportation Research. Genetic algorithms, which are a robust search technique formulated on the principles of natural selection and natural genetics, are employed to generate and identify better solutions until convergence is reached. The selection of good solutions is based on the so-called Pareto based fitness evaluation procedure by comparing the relative strength of the generated solutions with respect to each of the adopte objectives.

An important aspect of multi-objective GA analysis is the definition of "fitness" of a solution. The "fitness" of a solution directly influences the probability of the solution being selected for reproduction to generate new offspring solutions. To overcome the afore-mentioned problem, a rank-based fitness assignment scheme is used where all non-dominated solutions in a given pool of solutions are assigned a rank of 1. Solutions that are dominated by one solution will have a rank of 2, those dominated by two solutions are given a rank of 3, and so on.

In the evaluation of a pool of solutions, one can identify a curve (for the case of two-objective problems) or a surface (for the case three- or higher multi-objective problems) which are composed of all non-dominated solutions. This curve or is known as the Pareto frontier. The GA optimization process seeks to generate new solutions that would give an improved frontier that dominates the existing frontier. This process continues until, ideally, a set of globally non-dominated solutions is found. These globally non-dominated solutions, called the Pareto optimalset,definethe Pareto optimal frontier.

The general procedure of GA operations and offspring generation in multi -objective programming is similar to that for single-objective programming. The main difference lies with the evaluation of "fitness" of each solution, which is the driving criterion of the search mechanism of genetic algorithms. An important consideration of the optimization process is to produce representative solutions that are spread more or less evenly along the Pareto frontier. This can be achieved by using an appropriate reproduction scheme to generate offspring solutions and to form a new pool of parent solutions.

The proposed procedure is illustrated by means of a highway maintenance optimisation problem at network level subject to five forms of resources and operation constraints. The five constraints were production requirements, budget, manpower availability, equipment availability, and rehabilitation schedule. The problem considered four highway classes, four pavement defects and three levels of maintenance-need urgency. Its objective was to select an optimal set of maintenance activities for an analysis period of 45 working days.

The decision variables were the respective amounts of maintenance work, measured in workdays, assigned to each maintenance treatment type. There were 48 treatment types referring to maintenance repairs arising from 4 distress forms of 3 maintenance-need urgency levels on 4 highway classes. The coded string structure of GA representation thus consisted of 48 cells. Each cell could assume an integer value of workdays from 0 to 45.

In this example, the following four objective functions were considered: (a) minimization of the total maintenance cost; (b) maximization of overall network pavement condition; (c)minimization of total manpower requirement, and (d) maximization of work production in total workday units. For the purpose of illustrating the flexibility of GAs in handling multi-objective problems, two different analyses were performed. They consisted of a two-objective and a three-objective problem. The objectives of the former were minimization of total maintenance cost and maximization of maintenance work production, while those of the latter were minimisation of total maintenance cost, maximisation of maintenance work production, and maximisation of overall network condition.

Figure 1 shows the convergence pattern of the Pareto frontier for the case with two- objective problem. The convergence trend was apparent with the Pareto front moving in the direction towards lower maintenance costs and higher maintenance work production. Figure 2 presents the Pareto optimal frontiers of both the two-objective and three-objective optimization solutions in the same two-dimensional plot of (maintenance cost) vs. (work production) for comparison. It can be seen that although the two Pareto frontiers are very close to each other, the Pareto frontier of the two-objective solutions appear to dominate the frontier of the three-objective solutions in terms of maintenance cost and work production. This was clearly the effect of introducing a third objective, i.e. maximizing pavement condition, in the three-objective problem, in compromising the two initial objectives.

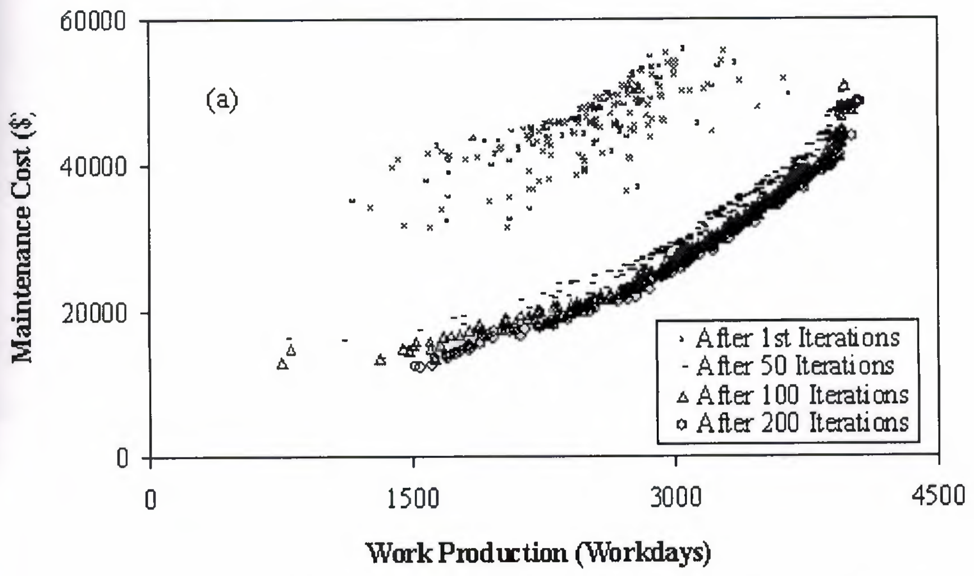


Figure 1 : Convergence of Pareto frontier in GA Solutions

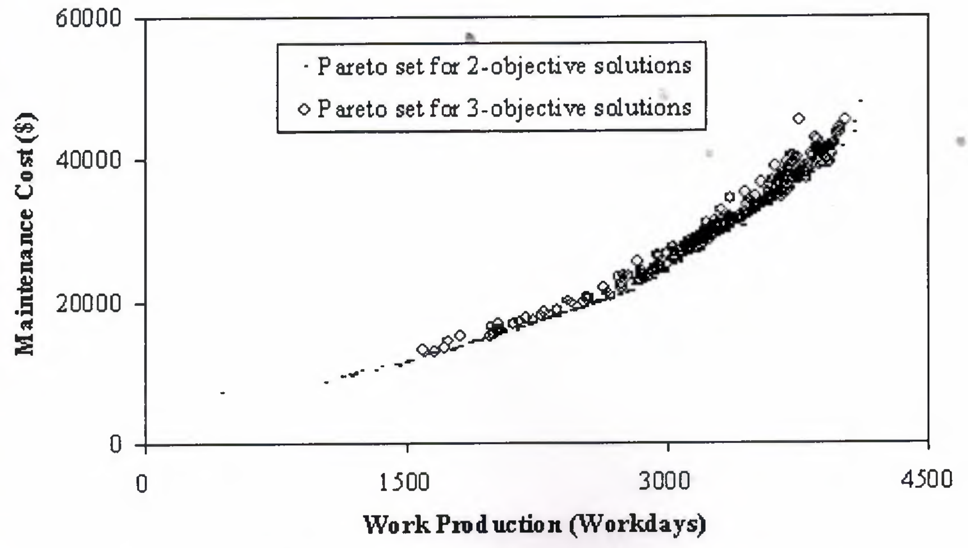


Figure 2: Comparison of Pareto frontiers for two- and three-objective solutions.

The robust search characteristics and multiple-solution handling ability of GAs are well suited for multi-objective optimization analysis of multi-objective highway infrastructure management and planning. The concepts of Pareto frontier and rank-based fitness evaluation, and a method of selecting optimal solution from the Pareto frontier were found to be effective in solving the problems. The proposed algorithm was able to produce a set of optimal solutions which were well spread on the Pareto frontier.

CHAPTER THREE

A GENETIC ALGORITHM-BASED OPTIMIZATION

3.1 Main Features for Optimization

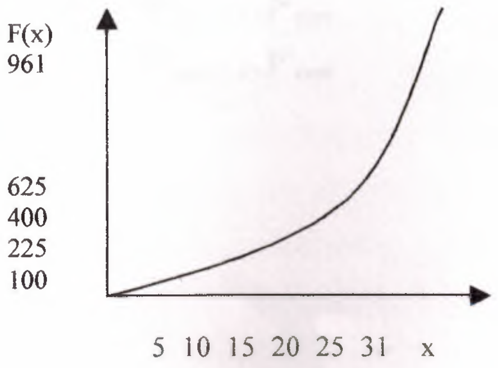
- Probabilistic algorithms for searching and optimization
- Mimic natural evolution process
- Capable of handling nonlinear, non-convex problem
- Optimization procedure does not require differentiation operation
- Capable of locating global and local optima within search domain
- Optimizations is based on population instead of a single point.

Let's consider a simple problem of maximization of the function $F(x)=x^2$, where $x \in [0,31]$ (see Fig).

In order to use GA we should first code variables in to bit strings as any integer number between 0 to 31 may be represented in binary number 5 symbols between (00000)=0 and (11111)=31 the length of chromosomes will be five.

Lets take 4 chromosomes with random set of genes as an initial population as:

01101
11000
01000
10011



Then define their fitness inserting appropriate real values into the function

as:$f(01000)_2=f(8=64)$

Thus the fitness of all individuals in calculated then accordance with the

Formula $P^i_5 = F_i / \sum^I_{J=1} F^i_5$, $i=1..4$

Survival probability for each individual is calculated where as cumulatives

$P^i_{cum} = \sum^I_{J=1} P^i_5$ $i=1..4$

Let's enter all the calculated values in table

Table3.1

Initial Population	Their integer values	$F(\ )=x^2$	P_5	P_{cum}	Num After Selection
01101	13	169	0.14	0.14	1
11000	24	576	0.49	0.63	2
01000	8	64	0.06	0.69	0
10011	19	361	0.31	1	1

Average	293	0.25	1
Maximum	576	0.49	2
Sum	1170	1	4

For the selection process we generate 4 random numbers from the range [0,1]. Suppose that we generated 0.1; 0.25; 0.5 and 0.8.

Comparing these values with cumulative probabilities, we obtaining the following

$0.1 < P^i_{cum}$

$P^I_{cum} < 0..25 < P^2_{cum}$

$P^1_{cum} < 0.5 < P^2_{cum}$

$P^3_{cum} < 0.8 < P^4_{cum}$

Looking at the right sides of these inequalities, one can easily see, that the first and the fourth chromosomes have passed the selection, each four taking a place in the new generation, the second chromosome-the most highly fitted has got 2 copies, while the third one did not survive at all. These indices are written in the right column of table 3.1.

Then crossover operation is applied. If the probability of crossover Pc=l is given, it means that, 4.1=4 chromosomes will participate in crossover process.

Let's c hoose t hem at r andom. S uppose t hat t he first a nd t he s econd s trings mate f rom crossover point 4, and the third with the forth-from crossover point 2.

0110 | 1
1100 | 0
11 | 000
00 | 011

Table3.2

Population after selection	Crossover	New Population	Value of X	$F(x)=X^2$
01101	4	01100	12	144
11000	4	11001	25	625
11000	2	11011	27	729
10011	2	10000	16	256

Average	439
Maximum	729
Sum	1754

Having compared both of the tables we see, for ourselves, that the population fitness is improved and we have come close to the solution.

The next recombination operation mutation, is performed on a bit-by-bit basis. If Pm=0.05 is given, it means that only one of twenty bits in population will be changed: 20- 0.05=1. Suppose, that the third bit of the fourth string undergoes mutation. I.e. X_4=10100. Repeating these operations in a finite number of generations we will get the chromosome (11111) corresponding to problem optimal solution. It is necessary to mention, that GA is especially effective for multi-extreme problems in the global solution search process. For example, if the junction is of the type shown. It is rather difficult to find its global maximum by means of traditional methods. Suppose that the junction is defined as [1]

$$f(x)=x.\sin(10\pi x)+1.$$

The problem is to find x from the range [-1,2], which maximizes the function f, i.e., to find x_0 such that

$$f(xo) \geq f(x), \text{ for all } x \in [-1,2].$$

It is relatively easy to analyze the junction f. The zeros of the first derivative f should be determined

$$F(x)=\sin(10\pi x\pi)+10\pi.x.\cos(10\pi.x)=0$$

The formula is equivalent to

$\tan(10\pi x)=(10\pi.x)$ It is clear that the above equation has an infinite number of solutions,

$$x_i=(2i-1)/20+\xi_i \qquad \text{for } i=1, 2,...,$$

$$X_0=0,$$

$$x_i=(2i-1)/20+\xi_i \qquad \text{for } i=1,-2,...,$$

Where terms ξ_1 represent decreasing sequences of real numbers (for İ = 1, 2,..., and i=-1, -2, ...) approaching zero. Note also that the function f reaches its local maxima for x, it i is an odd integer, and its local minima for x, if i is an even integer. Since the domain of the problem is $x \in$ [-1,2], the function reaches its maximum for

$$X_{19}=(37/20)+\xi_{19}=1.85+\xi$$

Where $f(X_{19})$ is slightly larger than

$$F(1.85)=1.85 \bullet \sin(18\pi+\pi/2)=1.0=2.85.$$

Assume that we wish to construct a genetic algorithm to solve the above problem, I.e., to maximize the function f. Let us discuss the major components of such a genetic algorithm in turn. [4]

3.1.1 Representation

We use a binary vector as a chromosome to represent real values of the variable x. The length of the vector depends on the required precision, which, in this example, is six places after the decimal point. The domain of the variable x has length 3; the precision requirement implies that the range [-1,2] should be divided into at least 3 • 1000000 equal size ranges. This means that 22 bits are required as a binary vector (chromosome):

$$2097152=2^{31} < 3000000 \leq 2^{33} = 4194304$$

The mapping from a binary (b_{21} b_{20} b_0) string into a real number x from the range [-1,2] is straightforward and is completed in two steps:

Convert the binary string (b_{21} b_{20} b_0) from the base 2 to base 10:

$$(<b_{21}\, b_{20}\, b_o>)_2 = (\sum {}^{21}{}_{I=0} b_i \ .\ 2^i)_{10} = x$$

Find a corresponding real number x:

$$x = -10 + x.(3/(2^{22}-1)),$$

Where -1,0 is the left boundary of the domain and 3 is the length of the domain. For example, a chromosome

(1000101110110101000111)

Represents the number 0.637197, since

$x'=(1000101110110101000111)_3 = 2288967$ and

$$x=1,0+2288967 .(3 / 4194303)= 0.637197.$$

Of course, the chromosomes

(0000000000000000000000) and (1111111111111111111111) Represent boundaries of the domain-1.0 and 2.0, respectively.

Initial population. The initialization process is very simple; we create a population of chromosomes, where each chromosome is a binary vector of 22 bits. All 22 bits for each chromosome are initialized randomly.

Evaluation function. Evaluation function eval for binary vectors v is equivalent to the function f:

$$Eval(v)=f(x),$$

Where the chromosome v represents the real value x.

As noted earlier, the evaluation junction plays the role of the environment, rating potential solutions in terms of their fitness. For example, three chromosomes:

Eval(v_1)=f=1.586345

Eval(v_2)=f(x_2)=0.078878

Eval (v_3)=f(x_3)=2.250650

Clearly, the chromosome v_3 is the best of the three chromosomes, since its evaluation returns the highest value.

During the reproduction phase of the genetic algorithm we would use two classical genetic operators; mutation and crossover.

As mentioned earlier, mutation alters one or more genes (positions in a chromosome) with a probability equal to the mutation rate. Assume that the fifth gene from the v_3 chromosome was selected for a mutation. Since the fifth gene in this chromosome is 0, it would be flipped into 1. So the chromosome v_3 after this mutation would be

V_3=(1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 01)

This chromosome represents the value x_3=1.721638 and f(x_3)=-0.082257. This means that this particular mutation resulted in a significant decrease of the value of the chromosome v_3 . On the other hand, if the 10th gene was selected for mutation in the chromosome v_3 them

V_3=(1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1)

The corresponding value x_3=1.630818 and f(x_3)=2.343555, an improve-ment over the original value of f(x_3)=2.250650.

Let us illustrate the crossover operator on chromosomes v_2 and v_3 . Assume that the crossover point was (randomly) selected after the 5th gene:

V_2=(00000|01110000000010000)

V_3=(11100|00000111111000101)

The two resulting offspring are

V'_2=(00000|00000111111000101)

V'_3=(1 1100|01110000000010000)

These offspring evaluate to

F(v'_2)=f(-0.998113)=0.940865.

F(v'_3)=f(1 .666028)=2.459245

Note that the second offspring has a better evaluation than both of its parents.

Parameters. For this particular problem we have used the following parameters population size ps=50, probability of crossover p_c =0.25, probability of mutation p_m =0.01.The following section presents some experimental results for such a genetic system.

Experimental results. We provide the generation number for which we noted an improvement in the evaluation fonction together with the value of the fonction. The best chromosome after 150 generations was

V_{max}=(1 111001101000100000101), Which

corresponds to a value x_{max}=1.850773

AS exprected, $x_{max}=1.85+\xi$ and $f(x_{max})$ is slightly larger than 2.85.

$\{\}\rightarrow$ $\{f(x)\}$ $x^{t}_{1}[1]$ | $\in<\geq$ • ξ π

Table 3.3.

Generation Number	Fitness Function
1	1.441942
5	2.250003
8	
9	2.250284
10	2.250363
12	2.328077
36	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

3.2 Applications

Genetic algorithms have been very successful at solving many types of problems.

1. Maximizing discontinuous, multimodal, multidimensional functions.
2. They have also been used on discrete problems, including the traveling salesperson, bin packing, and job-shop scheduling.

Here are two example problems, one discrete and one continuous, that are more realistic.

1. For an arbitrary expression in *n* Boolean variables, find an assignment of true and false values to the Boolean variables in the expression that make the expression true, if such an assignment exists.
2. This is called the Boolean satisfiability problem (SAT).
3. Each assignment of true and false values to the variables can be encoded in a bit string of length *n,* 0 for false and 1 for true.
4. The fitness of such a chromosome is how "close" to being true the assignment makes the Boolean expression, such as how many clauses in the expression are individually made true by the assignment.

5. Some (in Postscript) SATs are very hard for GAs

2. Maximize the function:

$f(x_1, x_2, x_3) = 21.5 + x_1 \sin(4 \text{ pi } x_1) + x_2 \sin(20 \text{ pi } x_2) + x_3$

for x_1 in [-3.0,12.1], x_2 in [4.1,5.8], and x_3 in [0.0,1.0].

1. This function is highly multimodal.
2. A chromosome can be encoded in three real numbers, one for each of x_1, x_2 and x_3, over their respective ranges.
3. The fitness of a chromosome is the value *of* $f\{x_1,x_2,x_3)$ for the x_1, x_2 and x_3 encoded in the chromosome.

3.2.1 Difficulties

Hitchhiking.

1. This may cause a GA to perform poorly.
2. Once a schema attains high fitness in a population, then the members of the population containing the schema will proliferate.
3. Bit values in positions not in the schema will hitchhike along with the schema as it propagates and will delay the discovery of high-fitness schemas with different values in those other bit positions.
4. For example, consider these 8 schemas.

   11111111..

   ....11111111..

   .... 11111111...

   ....1111111 ....

   ....11111111.........................

   ....11111111 ....

   ....11111111........

  11111111

5. Define a fitness function *f(x)* where *x* is a 64-bit string to be 8 times the number of these schemas it contains.
6. So *f (all* 64 bits 1) = 64,*f(*all 64 bits 0) = 0.
7. A GA performs poorly relative to hill-climbing on this because of hitchhiking.

3.3 The Neighborhood Constraint Method: A Genetic Algorithm-Based Multiobjective Optimization Technique

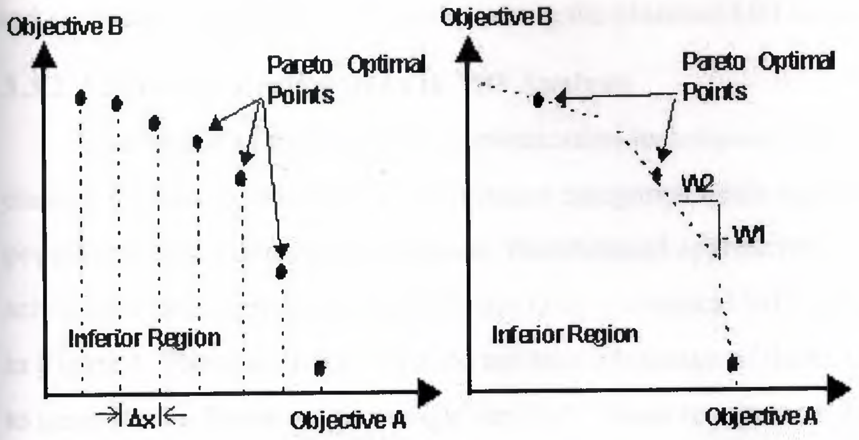
Multiobjective genetic algorithms (MOGAs), including VEGA, Pareto optimal ranking, and niched-Pareto, allow the tradeoffs among design objectives to be estimated in a single optimization model run. This paper presents a new MOGA technique called the neighborhood constraint method (NCM) that uses a combination of a neighborhood selection technique and location-dependent constraints. NCM is demonstrated for a complex, real -world problem, where it is compared with an integer programming (IP) procedure, an iterative procedure using a single-objective GA, and implementations of a Pareto and hybrid niched-Pareto MOGA technique. Preliminary results suggest that NCM is computationally more efficient than the SO approach and may perform better that the other MOGA techniques at promoting and maintaining population diversity over the entire tradeoff curve, using relatively small population sizes. These results show that further evaluation and comparisons of NCM are warranted.

3.3.1 Introduction

Engineering problems that are faced in everyday decision-making are often complex and multiobjective (MO), requiring solutions to be judged on their ability to satisfy competing design goals. For most multiobjective problems, there exists a set of nondominated, or Pareto, solutions that represent the optimal tradeoff relationship among objectives. For any solution in the Pareto set, any one design objective cannot be improved without sacrificing another.

The shape of the Pareto set can provide useful information to a decision maker. For example, in a pollution abatement problem, the Pareto set could identify the increasing costs associated with increasing levels of pollutant control. A decision-maker might use this relationship to determine reasonable control levels.

The purpose of an MO analysis is often to estimate the shape of an interesting portion of the Pareto set and to identify solutions along that portion. Several mathematical programming procedures, including the constraint method and the weighting method (Figure 1), are available for this purpose. These generating techniques estimate the noninferior set through iterative runs of an optimization model.



(A) Constraint Method (B) Weighting Method

Figure 1.Classical MO Generating Techniques. (A) All objectives except one are converted into constraints. The remaining objective is maximized. The constraints are incremented and the model is run iteratively. (B) All objectives are weighted and considered in the objective function simultaneously. A different set of weights is used in each run.

Alternatively, preference-based MO techniques, such as goal programming, may require only a single optimization model run to identify the solution that best meets prespecified "ideal" levels of each objective. Both classes of classical MO procedures are discussed in detail by Cohon (1978).

MO procedures have been used successfully within traditional mathematical optimization procedures, such as linear programming, integer programming and nonlinear programming. Many realistic problems, however, involve complex, nonlinear relationships that do not fit readily into one of these traditional frameworks.

Recently, many such problems have been solved by employing non-gradient-based, probabilistic search techniques such as genetic algorithms (GAs) and simulated annealing (SA). Both GA and SA may be used in the classical MO procedures. However, the inherent parallel structure of a GA provides some important advantages in MO analysis: with the appropriate schemes and operators, the population of solutions carried in a GA can be directed to converge along the Pareto set in a single run. Several multiobjective GA (MOGA) techniques that take advantage of this behavior are summarized in the following section.

The objective of this paper is to present a new method for MOGA, called the neighborhood constraint method (NCM). NCM differs from previous MOGA approaches by using a unique combination of a neighborhood selection technique and location-dependent constraints. A description of NCM and its application to an illustrative air quality management problem are provided. Performance of NCM is compared with those of several other MOGA techniques, as well as with the performance of a single-objective GA (SOGA) and an integer programming approach using the classical MO constraint method.

3.3.2. Literature Review :GAs in MO Analysis

In a review of multiobjective optimization techniques, Fonseca and Fleming (1995) classify MOGA techniques into four major categories: plain aggregation methods, population-based non-Pareto methods, Pareto-based approaches, and niche induction schemes. Plain aggregation methods use GAs in classical MO procedures like those depicted in Figure 1. These methodologies do not take advantage of the GA population-based search to generate the Pareto set in a single run, but instead require many iterations.

The most well-known population-based, non-Pareto method is the Vector Evaluated Genetic Algorithm, or VEGA (Schaffer, 1991). In the VEGA scheme, the population is divided into groups in each generation, and each group is evaluated with only a single objective. Individuals from different groups are allowed to mate with the intent that offspring will perform well with respect to the objectives of both their parents. While this approach and several of its variations have been successful in locating the elbow and extreme points of the Pareto set, it is recognized that exploration of the entire Pareto set is difficult.

In an intriguing non-Pareto technique, Hajela and Lin (1992) used a combination of restrictive mating, objective weighting, and fitness sharing. Their variable weighting approach treats the weights on objectives as decision variables that are encoded into the GA chromosome for each individual. Combined with fitness sharing, this approach allows individuals to evolve to represent a large number of different weightings. Fleming and Pashkevich (1985) noted that any technique that aggregates objectives linearly in the objective function, as in VEGA and Hajela and Lin's techniques, is not capable of exploring concave regions of the Pareto set.

Pareto-based approaches were proposed by Goldberg (in 1989) and have become a major focus of MOGA research. These techniques explicitly make use of the definition of Pareto optimality. In Pareto schemes, the fitness of an individual is defined in terms of rank. For each generation of the GA, the non-dominated set of solutions in the population is given the top rank and then removed from that population. This step is repeated, where in each iteration the non-dominated set of solutions remaining in the population is given the next rank, until all solutions in the population are assigned a rank. Rankings are then typically used in a tournament selection to encourage the exploration in the direction of nondominated individuals.

Successful applications of this procedure have been reported by Cieniawski et al. (1995) and Ritzel et al. (1994). Liepins et al. (1988) found the Pareto optimal ranking scheme to be superior to VEGA. An important advantage of Pareto approaches is that they are able to identify solutions in concave areas of the Pareto set.

Pareto schemes typically suffer from population drift, where the population migrates to a small portion of the Pareto set (Goldberg and Segrest, 1987). To assure a uniform sampling of the Pareto set, recent applications of Pareto-based MOGA by Fonseca and Fleming (1993), Horn et al. (1994), and Srinivas and Deb (1994) have incorporated niching schemes.

In both the Pareto and niched-Pareto methods, there may be a likelihood of mating between individuals that are far apart in the nondominated set. These solutions likely will be very different in decision space, and as a result, their children may not meet either objective satisfactorily (Goldberg, 1989). The niched-Parteto scheme by Fonseca and Fleming (1993)and the non-Pareto scheme by Hajela and Lin (1992) deal with this problem by restricting mating to individuals in similar regions in the objective space.

3.3.3 Neighborhood Constrained Method

3.3.3.1 Overview

The Neighborhood Constraint Method (NCM) presented here is a MOGA technique that: samples effectively from along the entire Pareto set, including the endpoints examines both convex and concave regions of the Pareto set, and provides a computationally-efficient approach .While MOGA techniques by Horn et al. (1994) and Fonesca and Fleming (1993) also address these issues, NCM was developed concurrently and uses a very different approach.

NCM falls into the class of non-Pareto MOGA techniques because it does not make explicit use of Pareto dominance in fitness evaluation. NCM is inspired by the natural phenomena of adaptation to environmental conditions that are locationally, or geographically, dependent (eg. the gradual darkening of skin color seen in human populations as one travels south from northern Europe to the equator). Adaptation, in this case, arises either from the preferential selection of traits that perform well with respect to the local environment, or through migration of traits to environments where they are better suited.

NCM includes the following major operations: population indexing, location-dependent constraints, and restrictive mating using a neighborhood selection scheme. These operations are described below.

3.3.3.2 Population Indexing

Each individual is given an index to define its location in the population. Indices are assigned in the following manner: for problems with two objectives, individuals are simply indexed from 1 to n, where n is the population size. For three objectives, the population is indexed in a matrix form so that each individual is given a set of x-y ordinal coordinates. The dimension of the matrix is m-by-n, where the product of m-by-n is the population size. Examples for two and three objectives are shown in Figure 2.

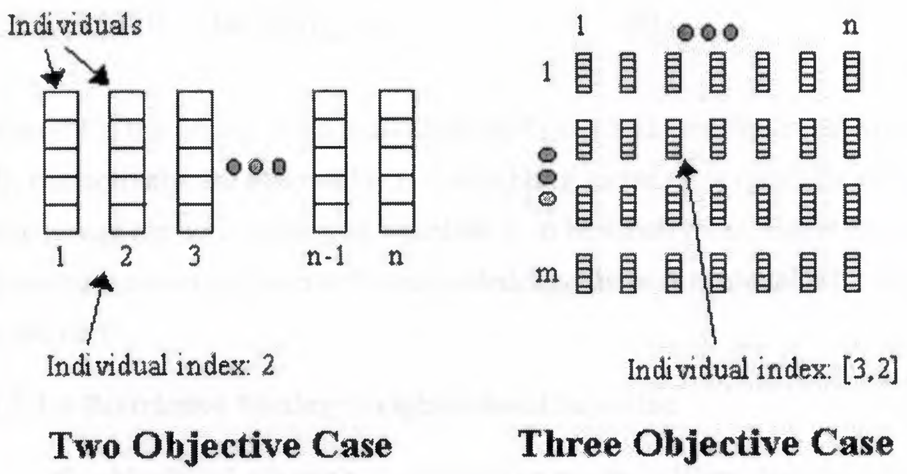


Figure 2. Indices used in the two and three objective cases. For a problem with K objective, there are K-1 index numbers used to describe each individual.

For problems of four or more objectives, individuals are indexed in a similar fashion, with the number of index values set equal to the number of objectives minus 1.

3.3.3 Location-Dependent Constraints

NCM handles multiple objectives in a very similar manner to the classical MO constraint method. For a K-objective problem, K-1 objectives are converted into constraints while the remaining objective is optimized. What makes this method unique is that the values of the constrained objectives are gradually varied at each index location in the population.

Consider the two-objective problem with objectives A and B and a population size of n. The individual with index 1 would be constrained to satisfy objective B at a minimum target level T_{Bmin}. The individual with index n would instead be constrained to a maximum target level T_{Bmax}. The constraint for any individual i in the population is generalized by:

$$T_{Bi} = T_{Bmin} + \frac{(i-1)*(T_{Bmax} - T_{Bmin})}{(n-1)} \quad (1)$$

Conversion of these targets to constraints in the GA is done through the use of penalties. Although the form of the penalty function may be problem specific, it generally can be incorporated in the fitness function as:

$$f = S(A) - w * g(S(B)) \quad (2)$$

$$g(S(B)) = \frac{(T_{Bi} - S(B))}{(T_{Bmax} - T_{Bmin})} \quad \text{for } S(B) < T_{Bi} \quad (3)$$

$$g(S(B)) = 0 \quad \text{for } S(B) \geq T_{Bi} \quad (4)$$

where: f is the fitness of an individual, S(A) and S(B) are the levels to which objectives A and B, respectively, are met, and w is a weighting factor. w is typically set to a very high value to encourage the constraint g, in Equation 2, to be strictly met. Equations 3 and 4 make the constraint on objective B to be one-sided, i.e., there is no penalty for exceeding the constraint.

3.3.3.4 Restrictive Mating: Neighborhood Selection

Neighborhood selection is a restrictive mating scheme discussed in the context of single objective, parallel GAs by DeJong and Sarma (1995). Neighborhood selection schemes restrict the mating of individuals to those within a specified "neighborhood," or vicinity, of each other. A number of different selection techniques, including roulette wheel and tournament selection, can be modified to promote neighborhood selection.

The implementation of neighborhood selection in NCM uses the notions of a selection radius, r, and a reference index location, L. The parameter r is fixed to be some fraction of n, the population size. The parameter L is an index that represents the location of an individual with respect to all other individuals in the population.

The likelihood of an individual being selected for mating is dependent on the proximity of that individual to the location L. An individual is considered for selection only if the distance between that individual and location L is within the selection radius, r. For instance, in a two-objective case, the distance between a reference location at L=4 and an individual at index 8 is 4. If r is greater than 4, the individual is considered for selection. Likewise, in the three-objective case, the distance between L at [4,2] and an individual at [3,7] is $((4-3)^2 + (2-7)^2)^{0.5}$, or 5.1. If r<5.1, then the individual will not be considered for selection.

The neighborhood selection process is described below for a two-objective problem. Assume that the population size, n, is 100; the selection radius, r, is defined as 0.05 * n, i.e., 5; and a tournament selection scheme is carried out. The selection process begins by setting the reference location, L, to 1. Individuals at index locations up to r (=5) locations away from

L(=1), i.e., from 1 to 6, will be considered as candidates for selection. Two of the six individuals in this range are selected using tournament selection. The winners of the selection undergo crossover, and the resulting individuals are placed into a new population at locations L and L+1, i.e., at locations 1 and 2. L is then incremented by two, and the selection and crossover processes are repeated. These steps are repeated until the new population has the same number of individuals as the previous population. This implementation of neighborhood selection is depicted in Figure 3.

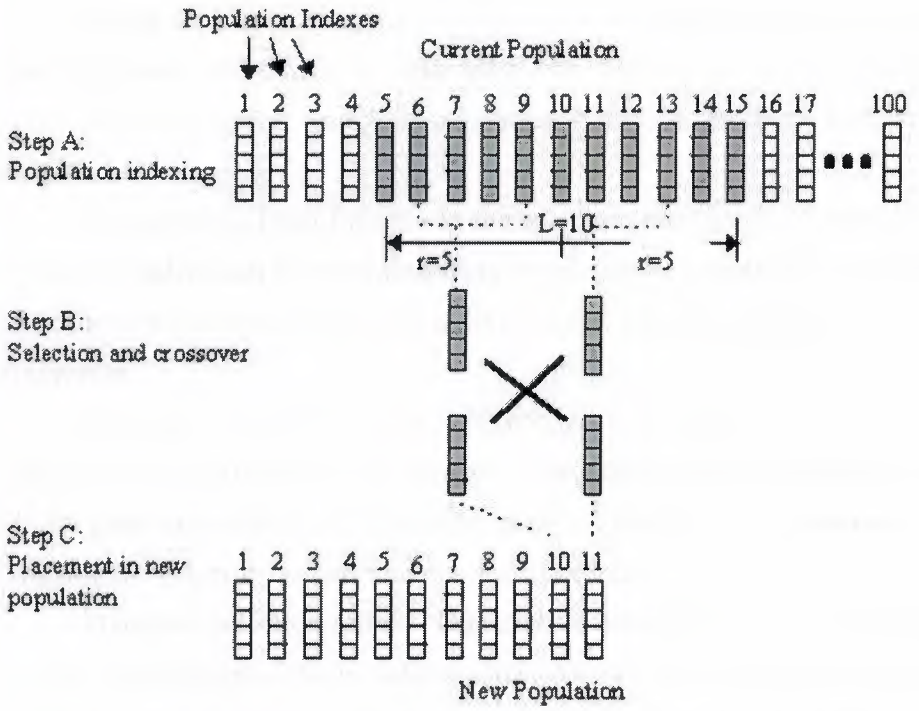


Figure 3. Neighborhood Selection. Step A: individuals are identified that fall within r places of L. Step B: two individuals are selected from this set and undergo crossover. Step C: the resulting individuals are placed into the next population at locations L and L+1.

3.3.3.5 Other Special Operations

In addition to the NCM-specific operations described above, other special operations may be used to improve performance for various types of problems. Several potential modifications are listed and described below.

Population sorting - Because the initial population is often generated randomly and is not necessarily in any order, individuals will likely not be in a location where they perform best. NCM performance may be improved by sorting the initial population, placing individuals closer to their ideal locations.

Dynamic scaling of T_{Bmax} and T_{Bmin} - It may be difficult to estimate the initial values of the minimum and maximum objective targets, e.g., T_{Bmax} and T_{Bmin}, for some problems.

When this occurs, T_{Bmax} and T_{Bmin} may be dynamically changed in each generation to reflect the maximum and minimum values in the algorithm to that point.

Locational elitism - To speed convergence, an elitism scheme compares the individual at each location in the population with the individual in that location in the previous generation. If the individual in the previous generation had a better fitness value than that of the current generation, it replaces the current individual.

Population distribution - Using Equation 1, individuals will be spaced evenly along objective B. Alternative distributions of individuals along objective B may be useful in concentrating the search toward areas in which the decision maker is interested.

Allow multiple individuals at a location - Instead of having one set of constraint values per individual, more than one individual can share the same set of constraints. This may provide a more robust search by allocating more GA resources to each of the objective targets.

Evaluation of local fitness - In the selection process, the fitness of individuals within the selection radius can be re-evaluated to determine how well they would perform at location L. Because the children of selected individuals will be placed at L and L+1, this promotes gene migration.

Selection of individuals outside the selection radius - It may be advantageous to select occasionally individuals from outside of the selection radius, which is expected to promote faster gene migration across the population. A factor can be introduced to allow sampling outside the selection radius at a pre-specified rate.

Dynamic selection radius - Our trials to date have used a selection radius that is equal to a fixed percentage of the population size. An alternative would be to allow the selection radius to change dynamically. For instance, the radius would start large and reduce as the population converges to the Pareto set.

3.3.4 Application

NCM was applied to a real-world, air quality management application with two conflicting objectives: minimizing the cost of controlling air pollutant emissions and maximizing the amount of emissions reduction.

This problem included approximately 300 air pollutant sources that varied in emissions amounts from 1 to nearly 8000 tons per year. Each source had from four to seven mutually -exclusive control options, including a no-control option. The average number of control options at a source was approximately five.

The choice of control techniques was discrete, i.e., when a control was chosen, it was applied to the entire emissions emanating from a source. Therefore, this problem requires a combinatorial search through approximately 5^{300} potential control strategies.

Associated with each control technique are an emissions removal efficiency and a cost -per-ton of emissions removed. The mathematical representation of the problem is as follows:

$$\textbf{minimize}\quad C = \sum_j \sum_t A_j[c_{j,t}, u_j] \qquad (9)$$

$$\text{s.t.}\quad \frac{\sum_j [u_j * (e_{j,t} * c_{j,t})]}{\sum_j u_j} \geq T \qquad (10)$$

$$c_{j,t} \in \{0,1\} \quad \forall j,t \qquad (11)$$

$$\sum_t c_{j,t} = 1 \quad \forall j \qquad (12)$$

where: C is the total cost; A_j is the cost at source j; $c_{j,t}$ is the binary variable representing whether control option t is used at source j; u_j is the emissions at source j; $e_{j,t}$ is the removal efficiency of control t at source j, and T is the target level of emissions reduction. The problem is described in more detail by Loughlin (1995).

This problem can be modeled and solved as an integer programming (IP) formulation, allowing the comparison of the GA results with the actual Pareto otimal solutions obtained using the IP. Also, this problem is closely related to another formulation that we have been examining in our research, in which a highly nonlinear air quality model is used in the

evaluation function to predict air quality impacts. This alternative GA formulation seeks to minimize costs subject to an ambient air quality constraint. The complexity of the atmospheric transport and chemistry makes the use of more classical optimization methodologies intractable.

Also, because of the duration of an atmospheric model run, the efficiency and quick convergence of a MOGA technique are important, providing the impetus for the work presented in this paper.

3.3.4.1 Methodology

Four MO methods were tested and compared for the case study problem. Each method is described briefly.

1. Integer Programming

First, an integer programming (IP) model was set up and solved to determine the true Pareto optimal costs for achieving emissions reductions of 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, and 55%. The IP model contained approximately 1300 binary decision variables.

2. Single-Objective GA

Next, a single-objective GA (SOGA) was run iteratively, one run for each level of reduction, i.e. 5%, 10%, etc. The SOGA, as well as the MOGA techniques tested in this paper, used real-coded genes. Coding of the decision variables is discussed by Loughlin (1995).

Other parameters used in the SOGA runs included: a mutation rate of 0.1%, a uniform crossover rate of 35%, tournament selection, and elitism to preserve the best individual in each generation. 30% of genes that underwent crossover were subjected to a linear recombination. These parameter values were found to perform well through trials of various parameterizations. Population sizes of 50 and 150 individuals were tested, each for 15 different random number seeds. The runs were terminated when no improvement in fitness was observed in 20 consecutive generations.

3. The Neighborhood Constraint Method

NCM was then used to generate the tradeoff curve in a single run. The NCM-specific parameters used in this problem are listed in Table 1.

Table 1. Parameters and options used in the neighborhood constraint method.

Parameter	Implementation
Decision variable en-coding:	real-valued
Mutation rate:	0.1 %
Selection method:	tournament selection (within selection radius)
Uniform crossover rate:	15 %
Selection radius:	4% of population size
Selection outside radius:	2 %
Elitism:	Locational
Fitness evaluation:	Dynamic T_{Bmin} and T_{Bmax},
	Local fitness evaluation
Population sorting:	After evaluation of the initial population

For NCM, only 10% of the genes that underwent crossover were linearly recombined. While this value may appear low, trials suggested that a low value allowed for better gene migration across the population with less chance of disruption.

Once again, runs were made for population sizes of 50 and 150 individuals for 15 different random number seeds. Termination occurred after 20 generations with no improvement in the average cost of individuals in the population.

Pareto Ranking and Hybrid Niched-Pareto MOGAs

Pareto ranking and a hybrid niched-Pareto MOGA technique were implemented and tested for the same population sizes and random seeds. The Pareto ranking scheme was based on the procedure described by Goldberg (1989).

The hybrid technique was a combination of two techniques described in the literature, and made use of:

Objective sharing - Combined with tournament selection, this technique encouraged exploration of the tradeoff curve by promoting selection of individuals in relatively nonpopulated areas. The objective sharing implementation was based on the procedure described by Horn et al. (1994).

Restrictive mating - This technique, adapted from Fonseca and Fleming (1993), used a parameter, $\square_{mating}$, to encourage the mating of individuals that are similar in objective space. Tournament selection was used in this hybrid method. A number of individuals equal to 10% (arbitrarily chosen) of the population size was sampled. If there was a tie in the ranking of the best individuals in the sample, continuously- updated objective sharing was used to identify the winner, as described by Horn et al. (1994). Another sampling of roughly 10% of the population was taken to find a mate. After the sampling was completed, the new set of individuals was evaluated to determine whether the individuals fell within an objective-space distance, □$_{mating}$, from the winner of the first tournament. The fitness of individuals that fell within this distance was increased to encourage selection. If there was a tie, objective sharing was carried out in the second sampled set to identify the mate.

4. Results

Table 2 summarizes the preliminary results of these trials. Costs obtained from each of the techniques, in millions of dollars per year, are provided for various target levels of reduction. Since the controls being considered are discrete, it may not be possible to meet a target reduction level exactly. Therefore, the cost displayed is for the strategy that most cost-effectively reduces greater than or equal to the current target, but less than the next higher target level.

Table 2. Results of the IP, SOGA, and MOGA runs. Population sizes of 50 and 150 were tested for 15 random number seeds.

		Multiobjective Technique for Various Population Sizes								
		IP (1)	SOGA		NCM		Pareto (2)		Hybrid (3)	
Redux	Performance	n/a	50	150	50	150	50	150	50	150
5%	Best	0.59	0.67	0.86	0.77	0.62	n/a	n/a	n/a	0.97
	Mean	n/a	0.93	0.95	1.18	0.72	n/a	n/a	n/a	1.46
	Stdev	n/a	0.25	0.06	0.33	0.07	n/a	n/a	n/a	0.40
10%	Best	1.26	1.33	1.41	1.54	1.31	n/a	n/a	2.32	1.81
	Mean	n/a	1.69	1.57	1.93	1.40	n/a	n/a	2.32	2.23
	Stdev	n/a	0.28	0.12	0.26	0.09	n/a	n/a	0.00	0.33
15%	Best	1.97	2.10	2.18	2.25	2.10	7.47	7.47	2.82	2.49
	Mean	n/a	2.65	2.39	2.84	2.20	7.47	7.47	3.78	3.40
	Stdev	n/a	0.51	0.12	0.41	0.08	0.00	0.00	1.35	0.63
20%	Best	2.73	3.67	3.03	3.12	2.91	n/a	n/a	3.63	3.42
	Mean	n/a	4.34	3.35	3.73	3.06	n/a	n/a	6.45	4.36
	Stdev	n/a	0.51	0.21	0.38	0.09	n/a	n/a	1.44	0.60
25%	Best	3.47	4.67	3.90	4.17	3.66	n/a	n/a	6.51	4.56
	Mean	n/a	5.85	4.11	4.86	3.77	n/a	n/a	7.52	5.78
	Stdev	n/a	0.80	0.26	0.55	0.07	n/a	n/a	0.87	0.81
30%	Best	5.47	6.17	6.17	6.42	6.24	n/a	n/a	7.39	6.74
	Mean	n/a	6.47	6.34	7.16	6.36	n/a	n/a	8.12	8.01
	Stdev	n/a	0.44	0.14	0.53	0.09	n/a	n/a	0.97	0.94
35%	Best	8.74	9.76	9.78	9.59	9.51	13.07	13.71	8.17	11.87
	Mean	n/a	9.78	9.78	11.08	10.17	13.95	13.97	8.53	12.78
	Stdev	n/a	0.00	0.00	2.79	0.40	0.60	0.16	0.46	0.80
40%	Best	13.60	16.05	18.36	17.39	17.84	15.12	14.97	17.10	16.90
	Mean	n/a	20.74	21.65	23.22	22.19	15.24	15.05	18.67	17.82
	Stdev	n/a	3.52	2.18	2.38	1.98	0.14	0.05	1.16	0.86
45%	Best	28.90	31.91	36.57	36.35	33.00	31.38	31.14	33.03	32.68
	Mean	n/a	34.72	36.70	39.11	36.45	31.46	31.17	34.34	33.20
	Stdev	n/a	2.57	0.10	2.82	1.05	0.05	0.02	1.30	0.42
50%	Best	44.20	47.89	47.61	53.96	49.20	47.74	47.61	49.50	48.68
	Mean	n/a	49.18	47.74	61.58	50.19	48.01	47.83	50.73	49.21
	Stdev	n/a	1.35	0.11	6.14	1.02	0.16	0.15	0.91	0.38
Approx. Total Evaluations		n/a	129000	330000	45500	198000	55000	200000	55000	200000

Notes:

(1) Population size, mean, and standard deviations are not applicable for the IP; results are Pareto optimal

(2) No nondominated solutions were found for ranges 5%, 10%, 20% and 25%. For 15%, the solution was one of the seeds used in the initial population.

(3) In roughly half the hybrid runs, no new nondominated solutions in the 0 to 35% reduction range were identified

Results are reported for population sizes of 50 and 150. Also compared are the number of fitness evaluations required by each method.

Figure 4 shows the nondominated sets and the final populations obtained through the NCM, the Pareto method, and the hybrid method, and compares them with IP the solutions.

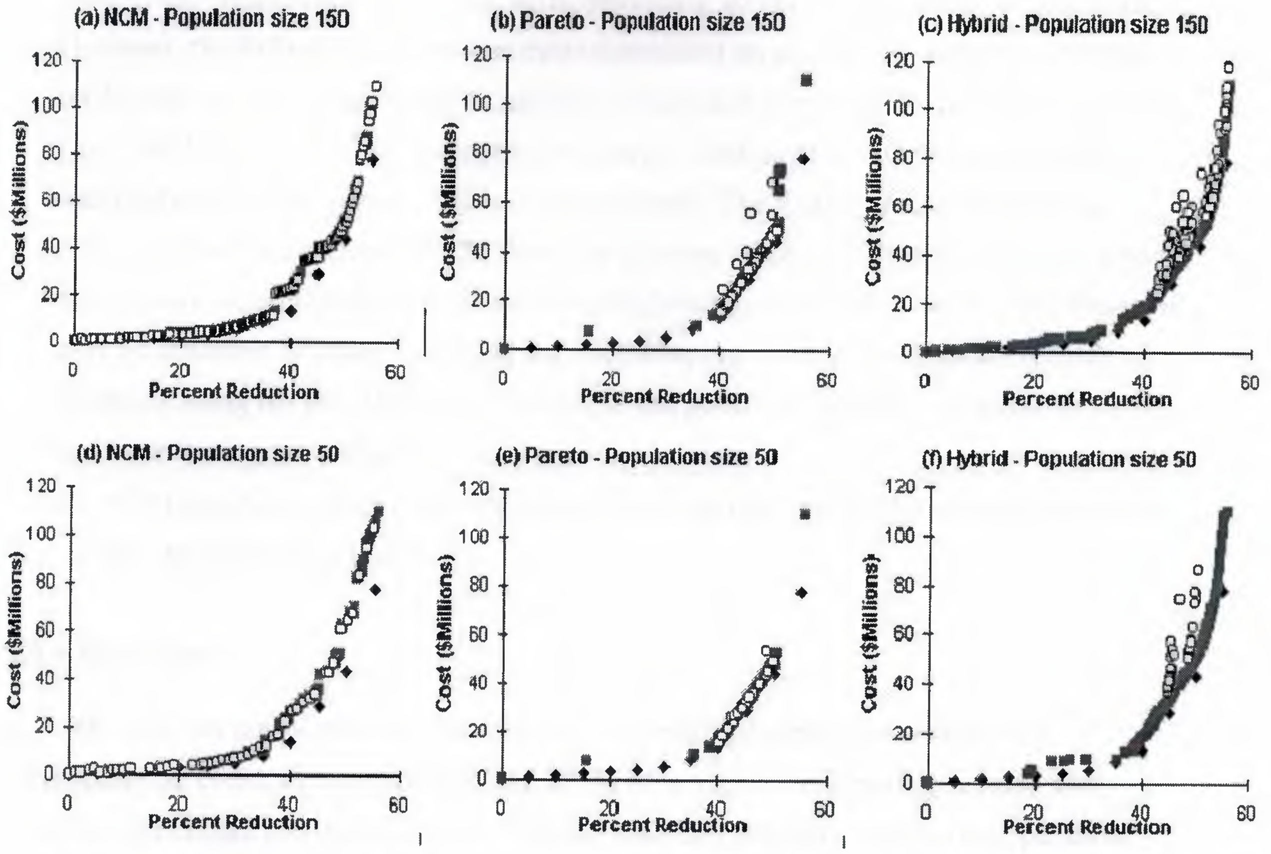


Figure 4. Final populations and nondominated sets identified throughout a typical run, for each MOGA technique. Individuals in the final population are represented as light-colored circles. IP-generated Pareto solutions are black diamonds. The nondominated set is represented as gray squares.

5. Discussion

Several observations may be made from the results. These include:

- None of the GA methods identified the true optimal points found by the IP. This was notunexpected because of the stochastic nature of GA search and the difficulty of the problem. However, the SOGA and MOGA techniques, with the exception of the Pareto method, did approximate the Pareto set to a degree that may be acceptable to a decision-maker.
- All three MOGA techniques required considerably fewer fitness evaluations and computational time than did the iterative SOGA technique. In about 1/3 of the duration of a SOGA run, which only identified 10 nondominated solutions, the MOGA techniques were able to identify several hundred to thousands of nondominated solutions.

- The Pareto method found better solutions near the elbow region of the curve, Compared to the hybrid, NCM, and SOGA. The results suggest that the Pareto technique was superior at the elbow because it tended to direct more search effort in that region than did the NCM and hybrid.
- Of the MOGA runs, the hybrid technique, with a population size of 150, identified the best nondominated solutions (Fig. 4c,f), providing good coverage of the entire curve. However, the hybrid technique was more dependent on the random number seed than NCM,and in several runs failed to identify solutions in the lower portion of the tradeoff curve. Although the hybrid technique was able to find good nondominated solutions, it was unableto maintain them throughout the search. The final population tended to converge toward the elbow. NCM was able to cover the noninferior set well, and also maintained the nondominated solutions throughout the search (Fig. 4a,d). This behavior may be desirable because it reduces the book-keeping required to maintain records of solutions along the set. The Pareto technique did poorly in covering the noninferior set and maintaining the nondominated solutions.
- NCM exhibited better coverage of the Pareto set than the hybrid technique for runs where the population was 50.

3.3.5 Summary

This section represent a new multiobjective genetic algorithm technique, the neighborhood constraint method (NCM). NCM uses a restrictive mating scheme and location-dependent constraints to promote and maintain diversity in the GA population.

The application of the NCM technique to a realistic problem was demonstrated and its performance was compared with those of an integer programming procedure, an iterative procedure using a single-objective GA optimization, and a Pareto and a hybrid niched-Pareto multiobjective GA (MOGA) techniques.

Preliminary results show that NCM performs better than the SOGA and the other MOGA techniques. NCM appears to provide good coverage of the Pareto set, and is capable of promoting and maintaining population diversity throughout the GA run with a relatively small population size. It does appear, however, that the Pareto-based techniques may be better at identifying regions near the elbow of the Pareto set.

These results suggest that NCM is a viable MOGA technique, and point to the need for a more comprehensive investigation of NCM to better identify the effects of the restrictive mating and the location-dependent constraints. A determination of the best selection approach (i.e., roulette wheel, tournament, etc.), selection radius, and recombination and crossover rates to use with NCM would be beneficial. NCM should also undergo a more rigorous comparison with the other MOGA techniques for a set of well-known problems.

Another potential use of NCM that deserves attention is in parallel GAs, since the methodology adapts well to the decentralized selection schemes discussed by De Jong and Sarma (1995).

CONCLUSION

A couple of conclusions from building block theory are of importance to note. Strings with very fit schemata of short length will have a high likelihood of being selected to create the next population, and thus pass on those schemata to strings in the new population. It has been shown that schemata of this form increase in number from one population to the next in an exponential fashion. In other words, n''' useful schemata are processed per generation, and the majority of these have small orders and lengths associated with them. These schemata are what give a GA the power to efficiently search through a problem space. This n^{J} feature is so important to GAs that it has been given a special name, implicit parallelism.

If the conception of a computer algorithms being based on the evolutionary of organism is surprising, the extensiveness with which this algorithms is applied in so many areas is no less than astonishing. These applications, be they commercial, educational and scientific, are increasingly dependent on this algorithms, the Genetic Algorithms. Its usefulness and gracefulness of solving problems has made it the more favorite choice among the traditional methods, namely gradient search, random search and others. GAs are very helpful when the developer does not have precise domain expertise, because GAs possess the ability to explore and learn from their domain.

In this project, the use of operators of GAs in optimization of engineering and commerce are considered. We believe that, by these interesting examples, one could grasp the idea of GAs with greater ease. The different optimization problems are described. The application of GA to solve optimization problem are given the selection procedure model parameters by using GA operators are represented. Also different problems solution, by using GA, is given.

In future, the developments of variants of GAs to tailor for some very specific tasks will be interesting. This, might defy the very principle of GAs that it is ignorant of the problem domain when used to solve problem. But we would realize that this practice could make GAs even more powerful.

REFERENCES

[1] Algorithms and Complexity, by Herbert S.Wilf, in 1986 published by Prentice-Hall, Inc. Obtained: Central Library of Imperial College (3 Computing 5.25 WIL)

[2] Recombination Variability and Evolution: algorithms of estimation and population-genetic models, by A.B.Korol, I.A.Preygel & S.I.Preygel, in 1994 published by Chapman & Hall.

[3] Proceedings of the Fifth International Conference On Genetic Algorithms, Fonseca, C. M, Fleming, P. J. (1993).tained : Central Library of Imperial College (4 Life Sciences 575.116.12 KOR)

[4] Genetic Algorithms in Business and Their Supportive Role in Decision Making, by Tom Bodnovich, in 16 November 1995, published by College of Business Administration Kent State

[5] Genetic Algorithms for Order Dependent Processes applied to Robot Path-Planning, by Yuval Davidor, in April 1989 published by Imperial College

[6] Genetic Algorithms in Engineering and Computer Science , edited by G.Winter, J.Periaux & M.Galan, published by JOHN WILEY & SON Ltd. in 1995.

[7] R. K. Ahuya, T. L. Magnanti, and J. B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Englewood Cliffs, NJ, 1993.

[8] E. L. Allgower and K. Georg, Numerical Continuation: An Introduction, Springer-Verlag, Berlin, 1990.

[9] D. M. Bates and D. G. Watts, Nonlinear Regression Analysis and Its Applications, John Wiley &Sons, Inc., New York, 1988.

[10] A. Bj.rck, *Least squares methods*, in Handbook of Numerical Analysis, P. G. Ciarlet and J. L. Lions, eds., North-Holland, Amsterdam, 1990, pp. 465-647.

[11] http://www.cs.qub.ac.uk/~M.Sullivan/ga/ga3.htmI

[12] http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2

[13] http://ltswww.epfl.ch/pub_files/brigger/thesis_html

[14] http://www-fp.mcs.anl.gov/otc/Guide/

[15] http://www.genetic-programming.com/creation.gif

[16] http://www.yahoo.com.tr

[17] http: //www.google.com