NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Genetic Algorithm Based Optimization

Graduation Project
COM- 400

Samer Alkhatib

Supervisor: Assoc. Prof. Dr. Rahib Abiyev

Nicosia - 2002

ACKNOWLEDGEMENTS

First of all I am happy to complete the task which I had given with blessing of God and also I am grateful to all the people in my life who have, supported me, advised me. taught me and who have always encouraged me to follow my dreams and ambitions. My dearest parents, my brothers and sister, my friends and my tutors. They have taught me that no dream is unachievable. As in the words of Walt Disney "If you can dream it, you can do it."

I wish to thank my advisor, Assoc. Prof. Dr. Rahib Abiyev, for intellectual support, encouragement, and enthusiasm, which made this project possible, and his patience for correcting both my stylistic and scientific errors.

My sincerest thanks must go to my friends, Maher Alkhatib, and Haider Alagha who shared their suggestions and evaluations throughout the completion of my project. The comments from these friends enabled me to present this project successfully.

And above, I thank God for giving me stamina and courage to achieve my objectives.

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 method is global optimization algorithm based on mechanics of natural selection and natural genetics, which is called Genetic Algorithms. In this project the application problems of genetic algorithms for optimization problems, its specific characters and structures are given. The basic genetic operation: Selections, reproduction, crossover and mutation operations are widely described the affectivity of genetic algorithms for optimization problem solving is shown. After the representation of optimizations problem, structural optimization and the finding of optimal solution of quadratic equation are given.

The practical application for selection, reproduction, crossover, and mutation operation are shown. The functional implementation of GA based optimization in MATLAB programming language is considered. Also the multi-modal optimization problem, some methods for global optimization and the application of Niching method for multi-modal optimization are discussed.

TABLE OF CONTANTS

INTRODUCTION OF GENETIC ALGORITHM

The GENETIC ALGORITHM is a model of machine learning which derives its behavior from a metaphor of the processes of EVOLUTION in nature. This is done by the creation within a machine of a POPULATION of Individuals represented by Chromosomes, in essence a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of evolution.

We should note that EVOLUTION (in nature or anywhere else) is not a purposive or directed process. That is, there is no evidence to support the assertion that the goal of evolution is to produce Mankind. Indeed, the processes of nature seem to boil down to different Individuals competing for resources in the ENVIRONMENT.

Some are better than others. Those that are better are more likely to survive and propagate their genetic material.

In nature, we see that the encoding for our genetic information (GENOME) is done in a way that admits asexual REPRODUCTION (such as by budding) typically results in OFFSPRING that are genetically identical to the PARENT. Sexual REPRODUCTION allows the creation of genetically radically different offspring that are still of the same general flavor (SPECIES).

At the molecular level what occurs (wild oversimplification alert!) is that pair of Chromosomes bumps into one another, exchange chunks of genetic information and drift apart. This is the RECOMBINATION operation, which GA/GPers generally refer to as CROSSOVER because of the way that genetic material crosses over from one chromosome to another.

The CROSSOVER operation happens in an ENVIRONMENT where the SELECTION of who gets to mate is a function of the FITNESS of the INDIVIDUAL, i.e. How good the individual is at competing in its environment.

Some GENETIC Algorithms use a simple function of the fitness measure to select individuals (probabilistically) to undergo genetic operations such as crossover or asexual REPRODUCTION (the propagation of genetic material unaltered). This is fitness-proportionate selection. Other implementations use a model in which certain randomly selected individuals in a subgroup compete and the fittest is selected. This is called tournament selection and is the form of selection we see in nature when stags rut

to vie for the privilege of mating with a herd of hinds. The two processes that most contribute to EVOLUTION are crossover and fitness based selection/reproduction.

As it turns out, there are mathematical proofs that indicate that the process of FITNESS proportionate REPRODUCTION is, in fact, near optimal in some senses.

MUTATION also plays a role in this process, although how important its role is continues to bad a matter of debate (some refer to it as a background operator, while others view it as playing the dominant role in the evolutionary process). It cannot be stressed too strongly that the GENETIC ALGORITHM (as a SIMULATION of a genetic process) is not a random search for a solution to a problem (highly fit INDIVIDUAL).

The genetic algorithm uses stochastic processes, but the result is distinctly non-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 string of the CHROMOSOME can be used to encode the values for the different parameters being optimized.

In practice, therefore, we can implement this genetic model of computation 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.

CHAPTER 1. WHAT ARE GENETIC ALGORITHMS (GAS)?

1.1. Evolution in a Changing World

Looking at the world around us, we see a staggering diversity of life. Millions of species, each with its own unique behaviors patterns and characteristics, abound. Yet, all of these plants and creatures have evolved, and continue evolving, over millions of years. They have adapted themselves to a constantly shifting and changing environment in order to survive. Those weaker members of a species tend to die away, leaving the stronger and fitter to mate, create offspring and ensure the continuing survival of the species. Their lives are dictated by the laws of natural selection and Darwinian evolution. And it is upon these ideas that genetic algorithms are based.

1.2. Defining Genetic Algorithms

What exactly do we mean by the term Genetic Algorithms? Goldberg (1989) defines it as:

> Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics.

Bauer (1993) gives a similar definition in his book:

> Genetic algorithms are software, procedures modeled after genetics and evolution.

GAs exploits the idea of the survival of the fittest and an interbreeding population to create a novel and innovative search strategy. A population of strings, representing solutions to a specified problem, is maintained by the GA. The GA then iteratively creates new populations from the old by ranking the strings and interbreeding the fittest to create new strings, which are (hopefully) closer to the optimum solution to the problem at hand. So in each generation, the GA creates a set of strings from the bits and pieces of the previous strings, occasionally adding random new data to keep the population from stagnating. The end result is a search strategy that is tailored for vast, complex, multimodal search spaces.

GAs is a form of randomized search, in that the way in which strings are chosen and combined is a stochastic process. This is a radically different approach to the problem solving methods used by more traditional algorithms, which tend to be more

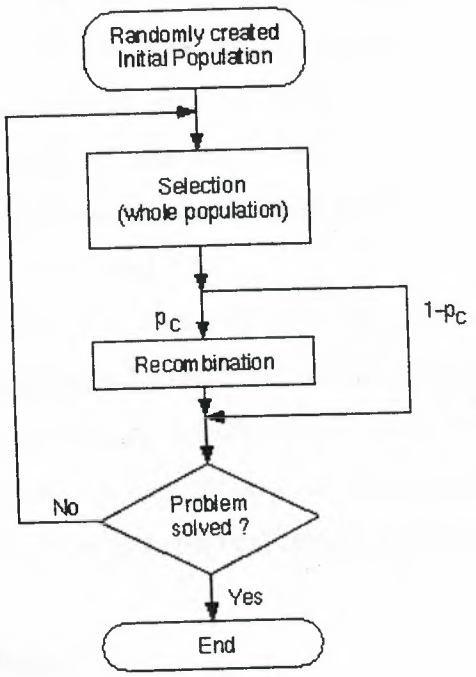
deterministic in nature, such as the gradient methods used to find minima in graph theory.

The idea of survival of the fittest is of great importance to genetic algorithms. GAs use what is termed as a fitness function in order to select the fittest string that will be used to create new, and conceivably better, populations of strings. The fitness function takes a string and assigns a relative fitness value to the string. The method by which it does this and the nature of the fitness value does not matter. The only thing that the fitness function must do is to rank the strings in some way by producing the fitness value. These values are then used to select the fittest strings. The concept of a fitness function is, in fact, a particular instance of a more general AI concept, the objective function.

1.3. Genetic Algorithms: A Natural Perspective

The population can be simply viewed as a collection of interacting creatures. As each generation of creatures comes and goes, the weaker ones tend to die away without producing children, while the stronger mate, combining attributes of both parents, to produce new, and perhaps unique children to continue the cycle. Occasionally, a mutation creeps into one of the creatures, diversifying the population even more. Remember that in nature, a diverse population within a species tends to allow the species to adapt to it's environment with more ease. The same holds true for genetic algorithms.

1.4. The Iteration Loop of a Basic Genetic Algorithm



The following flowchart shows the interactive cycle of a basic genetic algorithm. Firstly, an initial population of strings is created. The process then iteratively selects individuals from the population that undergo some form of transformation (via the recombination step) to create new a population. The new population is then tested to see if it fulfills some stopping criteria. If it does, then the process halts, otherwise another iteration is performed. (Diagram taken from Blickle, 1995).

1.5. Biological Metaphors for GAs

1.5.1. Genetics

Within most cells in the human body (and in most other living organisms) are rods like structures called chromosomes. These chromosomes dictate various hereditary aspects of the individual. Within the chromosomes are individual genes. A gene encodes a specific feature of the individual. For example, a person's eye color is dictated by a specific gene. The actual value of the gene is called an allele. So the eye color gene may produce brown eyes.

This is a grossly oversimplified look at genetics, but will suffice to show its correlation with genetic algorithms. A hierarchical picture is built up, with alleles being encoded as genes, with sequences of genes being chained together in chromosomes, which makes up the DNA of an individual.

When two individuals mate, both parents pass their chromosomes onto their offspring. In humans, who have 46 paired chromosomes in total, both parents pass on 23 chromosomes each to their child. Each chromosome passed to the child is an amalgamation of two chromosomes from a parent. The two chromosomes come together and swap genetic material, and only one of the new chromosome strands is passed to the child. So the chromosome strands undergo a crossover of genetic material, which leads to a unique new individual.

As if this were not enough, genetic material can undergo mutations, resulting from imperfect crossovers or other external stimuli. Although mutation is rare, it does lead to an even greater diversification in the population. It must be noted however that a significant number of mutations are harmful and can destroy good genetic code, so the rate of mutation must be low in order to prevent severe degradation of the genetic code.

1.5.2. Genetic Algorithms

Genetic algorithms are modeled closely on the ideas presented above. GAs maintains string structures that are analogous to chromosomes. The gene idea maps to the elements within the string and the values stored in these string elements are analogous to alleles.

The strings are rated by a fitness function. Strings are then selected for mating based on their ratings. When the strings are mated, crossover may occur, with a new child string being formed from parts of both parent strings. Mutation may also occur within the child string, based on a low mutation probability. Thus a new population is formed as a new generation of strings are created. The process then repeats itself, and a dynamically evolving population of strings runs through a number of iterations.

1.6. Traditional Optimization Methods vs. GAs

1.6.1. GAs and Robustness.

Two of the most remarkable traits of biological systems in general are their robustness and flexibility. Biological systems have methods for self-guidance, self-repair and reproduction. Very few artificial systems have any of these features.

Do GAs demonstrate at least some of these desirable traits from nature? Intuitively, we may think so as genetic algorithms are modeled closely on evolution in the biological world. And we would be right. Genetic algorithms have been proven to be robust,

flexible and efficient in vast complex spaces. For a discussion of this see (Holland, 1975).

1.6.2. Genetic Algorithm Traits

So genetic algorithms provide robustness, efficiency and flexibility when searching a problem space for the optimum solution. But why is this? For a more technical look at the power of GAs, a discussion on building blocks and schemata is required. For the moment, we shall just give a very brief look at the GAs search strategy.

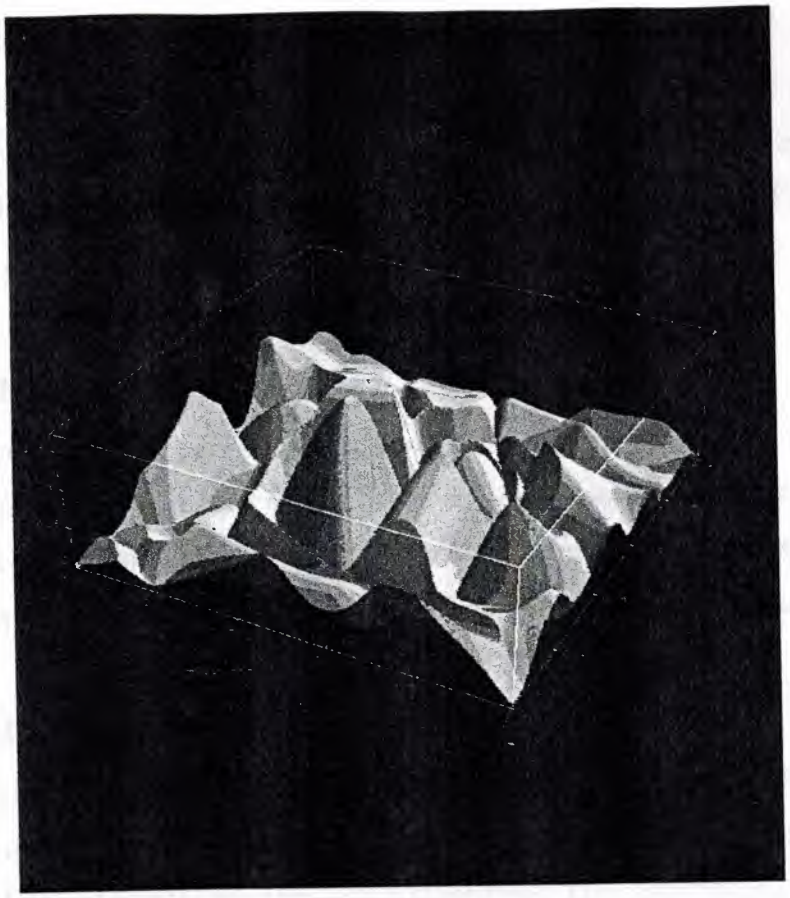
GAs judiciously uses the idea of randomness when performing a search. However, it must be clearly understood that GAs are not simply random search algorithms, which will be dealt with later. Random search algorithms can be inherently inefficient due to the directionless nature of their search. GAs is not directionless. They utilize knowledge from previous generations of strings in order to construct a new generation that will approach the optimal solution. In other words, they use past knowledge to direct the search. Such search algorithms are known as randomized search techniques, and are discussed further below.

1.6.3. Other Search Techniques

We will now look at some of the other, more traditional, optimization techniques, and show both their strengths and shortcomings when compared with GAs.

1.6.4. Some Sample Problem Spaces

In order to understand the problems of optimization, it is helpful to visualize exactly what the problem spaces may look like. Pictures of some difficult problem spaces can be viewed here. 1.6.5 Hill Climbing



Hill climbing optimization techniques have their roots in the classical mathematics developed in the 18th and 19th centuries. In essence, this class of search methods finds an optimum by following the local gradient of the function (they are sometimes known as gradient methods). They are deterministic in their searches. They generate successive results based solely on the previous results.

There are several drawbacks to hill climbing methods. Firstly, they assume that the problem space being searched is continuous in nature. In other words, derivatives of the function representing the problem space exist. This is not true of many real world problems where the problem space is noisy and discontinuous.

Another major disadvantage of using hill climbing is that hill climbing algorithms only find the local optimum in the neighborhood of the current point. They have no way of looking at the global picture in general. However, parallel methods of hill-climbing can be used to search multiple points in the problem space. This still suffers from the problem that there is no guarantee of finding the optimum value, especially in very noisy spaces with a multitude of local peaks or troughs.

1.6.6. Enumerative

The basis for enumerative techniques is simplicity itself. To find the optimum value in a problem space (which is finite), look at the function values at every point in the space. The problem here is obvious. This is horribly inefficient. For very large problem spaces, the computational task is massive, perhaps intractably so.

1.6.7. Random Search Algorithms

Random searches simply perform random walks of the problem space, recording the best optimum values discovered so far. Efficiency is a problem here as well. For large problem spaces, they should perform no better than enumerative searches. They do not use any knowledge gained from previous results and thus are both dumb and blind.

1.6.8. Randomized Search Techniques

A randomized search algorithm uses random choice to guide themselves through the problem search space. But these are not just simply random walks. These techniques are not directionless like the random search algorithms. They use the knowledge gained from previous results in the search and combine them with some randomizing features. The result is a powerful search technique that can handle noisy, multimodal search spaces with some relative efficiency. The two most popular forms of randomized search algorithms are simulated annealing and genetic algorithms.

1.6.9. The Differences between Genetic Algorithms and Traditional Methods

The following list is a very quick look at the essential differences between GAs and other forms of optimization. For a more complete discussion, see (Goldberg, 1989).

1. Genetic algorithms a coded form of the function values (parameter set), rather than with the actual values them. So, for example, if we want to find the minimum of the function f(x) =x3+x2+5, the GA would not deal directly with x or y values, but with strings that encode these values. For this case, strings representing the binary x values should be used.
2. Genetic algorithms use a set, or population, of points to conduct a search, not just a single point on the problem space. This gives GAs the power to search noisy spaces littered with local optimum points. Instead of relying on a single

point to search through the space, the GAs looks at many different areas of the problem space at once, and uses all of this information to guide it.

3. Genetic algorithms use only payoff information to guide themselves through the problem space. Many search techniques need a variety of information to guide them. Hill climbing methods require derivatives, for example. The only information a GA needs is some measure of fitness about a point in the space (sometimes known as an objective function value). Once the GA knows the current measure of "goodness" about a point, it can use this to continue searching for the optimum.
4. GAs is probabilistic in nature, not deterministic. This is a direct result of the randomization techniques used by GAs.
5. GAs is inherently parallel. Here lies one of the most powerful features of genetic algorithms. GAs, by their nature, is very parallel, dealing with a large number of points (strings) simultaneously. Holland has estimated that a GA processing n strings at each generation, the GA in reality processes n^3 useful substrings. This becomes clearer when schemata are examined.

1.7. Basic Genetic Algorithm Operations

1.7.1. The Inner Workings of a Genetic Algorithm

With GAs having such a solid basis in genetics and evolutionary biological systems, one might think that the inner workings of a GA would be very complex. In fact, the opposite is true. Simple GAs are based on simple string copying and substring concatenation, nothing more, nothing less. Even more complex versions of GAs still use these two ideas as the core of their search engine. All this will become clear when we walk through a simple GA optimization problem.

1.7.2. The Basic Genetic Algorithm Operations

There are three basic operators found in every genetic algorithm. (Although some algorithms may not employ the crossover operator, we shall refer to them as evolutionary algorithms rather than genetic algorithms.)

1. Reproduction
2. Crossover
3. Mutation

1.7.3. Reproduction

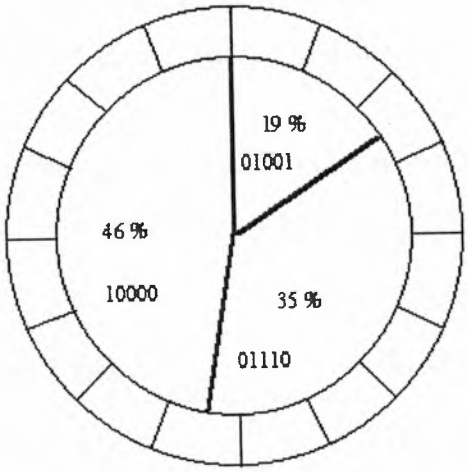
The reproduction operator allows individual strings to be copied for possible inclusion in the next generation. The chance that a string will be copied is based on the string's fitness value, calculated from a fitness function. For each generation, the reproduction operator chooses strings that are placed into a mating pool, which is used as the basis for creating the next generation. For example, look at the table below :

String	Fitness Value	Percentage
01001	5	19%
10000	12	46%
01110	9	35%

From this table, it is obvious that the string 10000 is the fittest, and should be selected for reproduction approximately 46% of the time. 01001 is the weakest, and should only be selected 19% of the time.

There are many different types of reproduction operators. One always selects the fittest and discards the worst, statistically selecting the rest of the mating pool from the remainder of the population. There are hundreds of variants of this scheme. None are right or wrong. In fact, some will perform better than others depending on the problem domain being explored. For a detailed, mathematical comparison of reproduction/selection strategies for genetic algorithms, see (Blickle, 1995).

For the moment, we shall look at the most commonly used reproduction method in GAs. The Roulette Wheel Method simply chooses the strings in a statistical fashion based solely upon their relative (ie. percentage) fitness values. To look abstractly at this method, consider the roulette wheel below, which is based on the previous example above.



Roulette Wheel Selection

When selecting the three strings that will be placed in the mating pool, the roulette wheel is spun three times, with the results indicating the string to be placed in the pool. It is obvious from the above wheel that there's a good chance that string 10000 will be selected more than once. This is fine. Multiple copies of the same string can exist in the mating pool. This is even desirable, since the stronger strings will begin to dominate, eradicating the weaker ones from the population. There are difficulties with this, as it can lead to premature convergence on a local optimum.

1.7.4. Crossover

Once the mating pool is created, the next operator in the GA's arsenal comes into play. Remember that crossover in biological terms refers to the blending of chromosomes from the parents to produce new chromosomes for the offspring. The analogy carries over to crossover in GAs.

The GA selects two strings at random from the mating pool. The strings selected may be different or identical, it does not matter. The GA then calculates whether crossover should take place using a parameter called the crossover probability. This is simply a probability value p and is calculated by flipping a weighted coin. The value of p is set by the user, and the suggested value is p=0.6, although this value can be domain dependant.

If the GA decides not to perform crossover, the two selected strings are simply copied to the new population (they are not deleted from the mating pool. They may be used multiple times during crossover). If crossover does take place, then a random splicing

point is chosen in a string, the two strings are spliced and the spliced regions are mixed to create two (potentially) new strings. These child strings are then placed in the new population.

As an example, say that the strings 10000 and 01110 are selected for crossover and the GA decides to mate them. The GA selects a splicing point of 3. The following then occurs:

Crossover in Action

The newly created strings are 10010 and 01100.

Crossover is performed until the new population is created. Then the cycle starts again with selection. This iterative process continues untii any user specified criteria are met (for example, fifty generations, or a string is found to have a fitness exceeding a certain threshold).

1.7.5. Mutation

Selection and crossover alone can obviously generate a staggering amount of differing strings. However, depending on the initial population chosen, there may not be enough variety of strings to ensure the GA sees the entire problem space. Or the GA may find itself converging on strings that are not quite close to the optimum it seeks due to a bad initial population.

Some of these problems are overcome by introducing a mutation operator into the GA. The GA has a mutation probability, m, which dictates the frequency at which mutation occurs. Mutation can be performed either during selection or crossover (though crossover is more usual). For each string element in each string in the mating pool, the GA checks to see if it should perform a mutation. If it should, it randomly changes the element value to a new one. In our binary strings, 1s are changed to 0s and 0s to 1s. For example, the GA decides to mutate bit position 4 in the string 10000:

The resulting string is 10010 as the fourth bit in the string is flipped. The mutation probability should be kept very low (usually about 0.001%) as a high mutation rate will destroy fit strings and degenerate the GA algorithm into a random walk, with all the associated problems.

But mutation will help prevent the population from stagnating, adding "fresh blood", as it were, to a population. Remember that much of the power of a GA comes from the fact that it contains a rich set of strings of great diversity. Mutation helps to maintain that diversity throughout the GA's iterations.

CAPTER2. OPTIMIZATION PROBLEM

2.1. What is Optimization

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.

The optimization problem is then:

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

2.2. Are all these ingredients necessary?

2.2.1. 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.

2.2.2. Variables

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

2.2.3. 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

2.3. Continuous Optimization

2.3.1. 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 minimizes of a

real-valued function, f(x), where x is a vector of n 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 $\nabla f(x) = \begin{pmatrix} \partial_1 f(x) \\ \vdots \\ \partial_n f(x) \end{pmatrix}$,

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 the current iterate x_k. The model function is defined by $q_k(\partial) = f(x_k) + \nabla f(x_k)^T \partial + \frac{1}{2}\partial^T \nabla^2 f(x_k)\partial$.

In the basic Newton method, the next iterate is obtained from the minimizes of q_k. When the Hessian matrix, $\nabla^2 f(x_k)$, is positive definite, the quadratic model has a unique minimizes that can be obtained by solving the symmetric $n \times n$ 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 minimizes x* at which the Hessian is positive definite. Moreover, the rate of convergence is quadratic, that is, $\|x_k + 1 - x^*\| \le \beta \|x_k - x^*\|^2$, 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 minimizes 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 minimizes 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 build up 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 minimizes is found exactly within just n iterations. For nonlinear problems, performance 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 no smooth or when derivatives are impossible to find, but it is unfortunately often used when one of the algorithms above would be more appropriate.

2.3.1.1. 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 discredited. In solving a system of nonlinear equations, we seek a vector such that f(x)=0 where x is an 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 minimizes 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 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 ith column requires the partial derivative of f with respect to the ith variable, while the solution of the linear system (1.1) requires order n^3 operations 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 β.

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 .

- Trust Region and Line-search Methods.
- Truncated Newton Method.
- Broyden's Method.
- Tensor Methods.
- Homotopy Methods.

2.3.1.2. Nonlinear Least Squares

The nonlinear least squares problem has the general form

$\min\{r(x) : x \in IR^n\}$, Where r is the function defined by $r(x) = \frac{1}{2}\|f(x)\|_2^2$ For some vector-valued function f that maps R^n to R^m.

Least squares problems often arise in data-fitting applications. Suppose that some physical or economic process is modeled by a nonlinear function ϕ that depends on a parameter vector x and time t. If b_i is the actual output of the system at time t_i, then the residual $\phi(x,t_i) - b_i$ measures the discrepancy between the predicted and observed outputs of the system at time t_i. A reasonable estimate for the parameter x may be obtained by defining the ith component of f by $f_i(x) = \phi(x,t_i) - b_i$,

and solving the least squares problem with this definition of f.

From an algorithmic point of view, the feature that distinguishes least squares problems from the general unconstrained optimization problem is the structure of the Hessian matrix of r. The Jacobian matrix of f, $f'(x) = (\partial_1 f(x),...,\partial_n f(x))$,a be used to express the gradient of r since $\nabla r(x) = f'(x)^T f(x)$.similarly, $f'(x)$ is part of the Hessian matrix $\nabla^2 r(x)$ since $\nabla^2 r(x) = f'(x)^T f'(x) + \sum_{i=1}^{m} f_i(x)\nabla^2 f_i(x)$.To calculate the gradient of r, we need to calculate the Jacobian matrix $f'(x)$. Having done so, we know the first term in the Hessian matrix $\nabla^2 r(x)$ without doing any further evaluations. Nonlinear least squares algorithms exploit this structure.

In many practical circumstances, the first term $f'(x)^T f'(x)$ in $\nabla^2 r(x)$ is more important than the second term, most notably when the residuals $f_i(x)$ are small at the solution. Specifically, we say that a problem has small residuals if, for all x near a solution, the quantities $|f_i(x)|\|\nabla^2 f_i(x)\|$, $i = 1,2,...,n$ are small relative to the smallest eigenvalue of $f'(x)^T f'(x)$.

- Gauss-Newton Method
- Levenberg-Marquardt Method
- Hybrid Methods
- Large Scale Methods
- Techniques for solving L.S. problems with constraints

- Notes and References

2.3.2. Constrained Optimization

2.3.2.1. Linear Programming

Software for linear programming (including network linear programming) consumes more computer cycles than software for all other kinds of optimization problems combined. There is a proliferation of linear programming software with widely varying capabilities and user interfaces. The most recent survey of linear programming software for desktop computers carried out by OR/MS Today (19 (1992), pp. 44-59) gave details on 49 packages!

The basic problem of linear programming is to minimize a linear objective function of continuous real variables, subject to linear constraints. For purposes of describing and analyzing algorithms, the problem is often stated in the standard form $\min\{c^T x : Ax = b, x \geq 0\}$, where $x \in IR^n$ is the vector of unknowns, $c \in R^n$ is the cost vector, and $A \in R^{m \times n}$ is the constraint matrix. The feasible region described by the constraints is a polytope, or simplex, and at least one member of the solution set lies at a vertex of this polytope.

The simplex algorithm, so named because of the geometry of the feasible set, underlies the vast majority of available software packages for linear programming. However, this situation may change in the future, as more software for interior-point algorithms becomes available.

2.3.2.2. Nonlinearly Constrained Optimization

The general constrained optimization problem is to minimize a nonlinear function subject to nonlinear constraints. Two equivalent formulations of this problem are useful for describing algorithms. They are, $\min\{f(x) : c_i(x) \leq 0, i \in \tau, c_i(x) = 0, i \in \varepsilon\}$ (1.1) where each c_i is a mapping from R^n to R, and τ and ε are index sets for inequality and equality constraints, respectively; and $\min\{f(x) : c(x) = 0, l \leq x \leq u\}$, (1.2) where c maps R^n to R^m, and the lower- and upper-bound vectors, l and u, may contain some infinite components.

2.3.2.3. Bound-Constrained Optimization

Bound-constrained optimization problems play an important role in the development of software for the general constrained problem because many constrained codes reduce the solution of the general problem to the solution of a sequence of bound-constrained problems. The development of software for this problem, which we state as $\min\{f(x): l \le x \le u\}$, is also important in applications because parameters that describe physical quantities are often constrained to lie in a given range.

Algorithms for the solution of bound-constrained problems seek a local minimizes x^* of f. The standard first-order necessary condition for a local minimizes x^* can be expressed in terms of the binding set $B(x^*) = \{i : x_i^* = l_i, \partial_i f(x^*) \ge 0\} \cup \{i : x_i^* = u_i, \partial_i f(x^*) \le 0\}$ at x^* by requiring that $\partial_i f(x^*) = 0, i \notin B(x^*)$

There are other ways to express this condition, but this form brings out the importance of the binding constraints. A second-order sufficient condition for x^* to be a local minimizes of the bound-constrained problem is that the first-order condition hold and that $w^T \nabla^2 f(x^*) w > 0$ for all vectors w with $w \ne 0, w_i = 0, i \in B_A(x^*)$, where $B_A(x^*) = B(x^*) \cap \{i : \partial_i f(x^*) \ne 0\}$ is the strictly binding set at .

Given any set of free variables F, we can define the reduced gradient and the reduced Hessian matrix, respectively, as the gradient of f and the Hessian matrix of f with respect to the free variables. In this terminology, the second-order condition requires that the reduced gradient be zero and that the reduced Hessian matrix be positive definite when the set F of free variables consists of all the variables that are not strictly binding at x^*. As we shall see, algorithms for the solution of bound-constrained problems use unconstrained minimization techniques to explore the reduced problem defined by a set F_k of free variables. Once this exploration is complete, a new set of free variables is chosen with the aim of driving the reduced gradient to zero.

The NEOS SERVER also has a bound-constrained minimization facility to solve these problems remotely over the Internet.

2.3.2.4. Network Programming

Network problems arise, as the name indicates, in applications that can be represented as the flow of a commodity in a network. The resulting programs can be linear or non-linear; however, we only discuss the linear case. Due to the network structure of the model, we can develop fast techniques for solving these problems.

For example, assume that you are the manager of a company that has different production lines in different locations. The goods produced by your company (in these different locations) are shipped to the distribution centers. In order to simplify the problem, let us say that there are two production lines and two distribution centers. The cost of shipping a unit of product from a production center to each distribution center is known. We also assume that we know the demand at each center and the production level of each production line. In other words, we have a table of the form:

	Distr.1	Distr.2	supply
Line1	5$/unit	8$/unit	30 units
Line2	7$/unit	9$/unit	25 units
Demand	20 units	35 units	

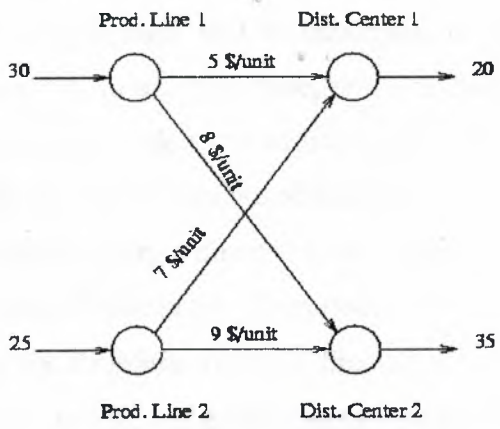
You want to minimize the cost of shipping your product to the different distribution centers while meeting the demand of the customers. Remember, your company produces items or units that cannot be broken down into fractions (cars for example); i.e., some of the decision variables representing your shipping problem must be integer.

One can build a mathematical model for solving the previous problem. An integer variable, t_{ij}, indicates the number of items that need to be shipped from location i to distribution center j. Mathematically, our model is

min	$5\tau_{11}$	$+8\tau_{12}$	$+7\tau_{21}$	$+9\tau_{22}$	
s.t.	$\tau_{11},$	$+\tau_{12},$			$=30$
			$\tau_{21},$	$+\tau_{22}$	$=25$
	$\tau_{11},$		$+\tau_{21},$		$=20$
		$\tau_{12},$		$+\tau_{22}$	$=35$
	$\tau_{11},$	$\tau_{12},$	$\tau_{21},$	τ_{22}	≥ 0

And all variables are integer. The first two constraints indicate that production centers 1 and 2 supply exactly 30 and 25 units of the product, respectively. The last two constraints indicate that we must meet the demand of each of the distribution lines. Note that we can formulate the first two constraints as $\leq$ constraints and the last two constraints as $\geq$ constraints; that is, we can produce at most 30 and 25 units and we do not want to exceed the demand of 20 and 35 units. However, since the total supply is equal to the total demand, it is clear that any feasible solution must satisfy all constraints as equality constraints (if the total demand is not equal to the total supply, we can add dummy production lines or distribution centers to the problem to make it balanced).

Note that only two entries appear in the column of a decision variable and that all the coefficients are equal to 1 (or -1 if we multiply a constraint by -1). In general, this constraint-matrix structure arises in optimization problems that can be modeled as directed network problems. For example, the previous model has the following network representation:



This is why, in such models, the constraint matrix is called the node-arc incidence matrix. It can be shown that every non-singular square sub-matrix of the node-arc incidence matrix is triangular and has a determinant that is either 1 or -1. Given this property along with integer demand and supply vectors, there is an integer optimal solution for the previous problem. That is, we can ignore the integer restriction on the variables and solve our problem as a linear program. As a matter of fact, the special structure of this class of problems permits developing special algorithms for solving these problems efficiently.

Network problems cover a large number of applications. Here, we describe some of them:

- **Transportation Problem.** We have a commodity that can be produced in different locations and needs to be shipped to different distribution centers. Given the cost of shipping a unit of commodity between each two points, the capacity of each production center, and the demand at each distribution center, find the minimal cost shipping plan. Note that the example that we described above is a transportation model.
- **Assignment Problem.** There are n individuals that need to be assigned to n different tasks. Each individual is assigned to one job only and each job is performed by one person. Given the cost that each individual charges for performing each of the n jobs, find a minimal cost assignment. Clearly, this problem is a special case of the transportation problem. Many techniques are developed for solving this class of problems since it is often encountered in application (for example, relaxing the tour constraint in a traveling salesman problem results in an assignment problem).
- **Maximum Value Flow.** Given a directed network of roads that connects two cities and the capacities of these roads, find the maximum number of units (cars) that can be routed from one city to another. Here, the constraints are the equilibrium or balance equations at each node (or road intersection); i.e., flow of the cars into a node is equal to the flow of the cars out of that node.
- **Shortest Path Problem.** Given a directed network and the length of each arc in this network, find a shortest between two given nodes.
- **Minimum Cost Flow Problem.** Given a directed network with upper and lower capacities on each of its arcs, and given a set of external flows (positive or negative)

that need to be routed through this network, find the minimal cost routing of the given flows through this network. Here, the cost per unit of flow on each arc is assumed to be known.

The NEOS SERVER operates a linear network optimization facility containing the codes NETFLOW and RELAX-IV to solve these problems remotely over the Internet.

2.3.2.5. Stochastic Programming

All of the model formulations that you have encountered thus far in the Optimization Tree have assumed that the data for the given problem are known accurately. However, for many actual problems, the problem data cannot be known accurately for a variety of reasons. The first reason is due to simple measurement error the second and more fundamental reason is that some data represent information about the future (e.g., product demand or price for a future time period) and simply cannot be known with certainty. We will discuss a few ways of taking this uncertainty into account and, specifically, illustrate how stochastic programming can be used to make some optimal decisions.

1. Recourse

The fundamental idea behind stochastic linear programming is the concept of recourse. Recourse is the ability to take corrective action after a random event has taken place. A simple example of two-stage recourse is the following:

- Choose some variables, **x,** to control what happens today.
- Overnight, a random event happens.
- Tomorrow, take some recourse action, **y,** to correct what may have gotten messed up by the random event.

We can formulate optimization problems to choose **x** and **y** in an optimal way. In this example, there are two periods; the data for the first period are known with certainty and some data for the future periods are stochastic, that is, random.

2. Example.

You are in charge of a local gas company. When you buy gas, you typically deliver some to your customers right away and put the rest in storage. When you sell gas, you take

it either from storage or from newly-arrived supplies. Hence, your decision variables are 1) how much gas to purchase and deliver, 2) how much gas to purchase and store, and 3) how much gas to take from storage and deliver to customers. Your decision will depend on the price of gas both now and in future time periods, the storage cost, the size of your storage facility, and the demand in each period. You will decide these variables for each time period considered in the problem. This problem can be modelled as a simple linear program with the objective to minimize overall cost. The solution is valid if the problem data are known with certainty, that is, if the future events unfold as planned.

More than likely, the future will not be precisely as you have planned; you don't know for sure what the price or demand will be in future periods though you can make good guesses. For example, if you deliver gas to your customers for heating purposes, the demand for gas and its purchase price will be strongly dependent on the weather. Predicting the weather is rarely an exact science; therefore, not taking this uncertainty into account may invalidate the results from your model. Your ``optimal" decision for one set of data may not be optimal for the actual situation.

3. Scenarios

Suppose in our example that we are experiencing a normal winter and that the next winter can be one of three scenarios: normal, cold, or very cold. To formulate this problem as a stochastic linear program, we must first characterize the uncertainty in the model. The most common method is to formulate scenarios and assign a probability to each scenario. Each of these scenarios has different data as shown in the following table:

Scenario	Probability	Gas Cost ($)	Demand (units)
Normal	1/3	5.0	100
Cold	1/3	6.0	150
Very Cold	1/3	7.5	180

Both the demand for gas and its cost increase as the the weather becomes colder. The storage cost is constant, say, 1 unit of gas is $1per year. If we solve the linear program for each scenario separately, we arrive at three purchase/storage strategies:

- Normal - Normal

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	0	0	500
2	100	0	0	500

- Total Cost = $1000
- Normal - Cold

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	0	0	500
2	150	0	0	900

- Total Cost = $1400
- *Normal - Very Cold*

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	180	180	1580
2	0	0	0	0

- Total Cost = $1580

We do not know which of the three scenarios will actually occur next year, but we would like our current purchasing decision to put is in the best position to minimize our expected cost. Bear in mind that by the time we make our second purchasing decision, we will know which of the three scenarios has actually happened.

4. Formulating a Stochastic Linear Program

Stochastic programs seek to minimize the cost of the first-period decision plus the expected cost of the second-period recourse decision.

Min $e^T x + E_\omega Q(x,\omega)$

Subject to $Ax = b$

$x \geq 0$ Where

$Q(x,\omega) = \min\ d(\omega)^T y$

Subject to $T(\omega)x + W(\omega)y(w) = h(\omega)$

The first linear program minimizes the first-period direct costs, $\mathbf{c^T x}$ plus the expected recourse cost, $Q(x,\omega)$ over all of the possible scenarios while meeting the first-period constraints, **Ax = b**

The recourse cost **Q** depends both on **x,** the first-period decision, and on the random event, ω the second LP describes how to choose $y(\omega)$ (a different decision for each random scenario ω). It minimizes the cost $\mathbf{d^T y}$ subject to some recourse function, **Tx + Wy = h.** This constraint can be thought of as requiring some action to correct the system after the random event occurs. In our example, this constraint would require the purchase of enough gas to supplement the original amount on hand in order to meet the demand.

One important thing to notice in stochastic programs is that the first-period decision, **x,** is independent of which second-period scenario actually occurs. This is called the nonanticipativity property. The future is uncertain and so today's decision cannot take advantage of knowledge of the future.

5. Deterministic Equivalent

The formulation above looks a lot messier than the deterministic LP formulation that we discuss elsewhere. However, we can express this problem in a deterministic for by introducing a different second-period **y** variable for each scenario. This formulation is called the deterministic equivalent.

Min $e^T x + \sum_{i=1}^{N} p_i d_i^T y_i$

Subject to $Ax = b$

$$T_i x + W_i y_i = h_i, i = 1,\ldots,N$$

$$x \geq 0$$

$$y_i \geq 0$$

where **N** is the number of scenarios and p_i is th e probability of the scenario's occurrence.

For our three-scenario problem, we have

$$\begin{array}{llllll} \text{Min} & e^T x & + p_1 d_1^T y_1 & + p_2 d_2^T y_2 & + p_3 d_3^T y_3 & \\ \text{s.t.} & Ax & & & & = b \\ & T_1 x & + W_1 y_1 & & & = h_1 \\ & T_2 x & + & W_2 y_2 & & = h_2 \\ & T_3 x & + & & W_3 y_3 & = h_3 \end{array}$$

$$x, y_i \geq 0$$

Notice that the nonanticipativity constraint is met. There is only one first-period decision, **x,** whereas there are **N** second-period decisions, one for each scenario. The first-period decision cannot anticipate one scenario over another and must be feasible for each scenario. That is, $\mathbf{Ax = b}$ and $\mathbf{T_i x + W_i y_i = h_i}$ for $\mathbf{i = 1,\ldots,N}$ Because we solve for all the decisions, **x** and $\mathbf{y_i}$ simultaneously, we are choosing **x** to be (in some sense) optimal over all the scenarios.

Another feature of the deterministic equivalent is worth noting. Because the **T** and **W** matrices are repeated for every scenario in the model, the size of the problem increases linearly with the number of scenarios. Since the structure of the matrices remains the same and because the constraint matrix has a special shape, solution algorithms can take advantage of these properties. Taking uncertainty into account leads to more robust solutions but also requires more computational effort to obtain the solution.

6. Comparisons with Other Formulations

Because stochastic programs require more data and computation to solve, most people have opted for simpler solution strategies. One method requires the solution of the

problem for each scenario. The solutions to these problems are then examined to find where the solutions are similar and where they are different. Based on this information, subjective decisions can be made to decide the best strategy.

7. Expected-Value Formulation

A more quantifiable approach is to solve the original LP where all the random data have been replaced with their expected values. Hopefully in this approach we will do all right on average. For our example then, we consider the (expected value) problem data to be

Year	Gas cost ($)	Demand
1	5.0	100
2	6.167	143.33

Solving this problem gives the following result:

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	143.33	143.33	1360
2	0	0	0	0

Cost = $1360.00

Let's compute what happens in each scenario if we implement the expected value solution:

Scenario	Recourse Action	Recourse Cost	Total Cost
Normal	Store 43.33 excess @ $1 per unit	43.33	1403.33
Cold	Buy 6.67 units @ $6 per unit	40	1400
Very Cold	Buy 36.67 units @ $7.5 per unit	275	1635

minimizing over a number of scenarios and, as a result, sacrifices the minimum cost for each scenario in order to obtain a robust solution over all the scenarios.

2.4 Discrete Optimization

In many applications, the solution of an optimization problem makes sense only if certain of the unknowns are integers. Integer linear programming problems have the general form

$\min\{c^T x : Ax = b, \quad x \geq 0, \quad x \in Z^n\}$(1.1) where Z^n is the set of n-dimensional integer vectors. In mixed-integer linear programs, some components of x are allowed to be real. We restrict ourselves to the pure integer case, bearing in mind that the software can also handle mixed problems with little additional complication of the underlying algorithm.

Integer programming problems, such as the fixed-charge network flow problem and the famous traveling salesman problem, are often expressed in terms of binary variables. The fixed-charge network problem modifies the minimum-cost network flow paradigm by adding a term $f_{ij} y_{ij}$ to the cost, where the binary variable y_{ij} is set to 1 if arc (i, j) carries a nonzero flow x_{ij}; it is set to zero otherwise.

In other words, there is a fixed overhead cost for using the arc at all. In the traveling salesman problem, we need to find a tour of a number of cities that are connected by directed arcs, so that each city is visited once and the time required to complete the tour is minimized. One binary variable is assigned to each directed arc; a variable x_{ij} is set to 1 if city i immediately follow city j on the tour, and to zero otherwise.

2.5. Multi-Objective Optimization

Most realistic optimization problems, particularly those in design, require the simultaneous optimization of more than one objective function. Some examples:

- In bridge construction, a good design is characterized by low total mass and high stiffness.
- Aircraft design requires simultaneous optimization of fuel efficiency, payload, and weight.

- In chemical plant design, or in design of a groundwater remediation facility, objectives to be considered include total investment and net operating costs.
- A good sunroof design in a car could aim to minimize the noise the driver hears and maximize the ventilation.
- The traditional portfolio optimization problem attempts to simultaneously minimize the risk and maximize the fiscal return.

In these and most other cases, it is unlikely that the different objectives would be optimized by the same alternative parameter choices. Hence, some trade-off between the criteria is needed to ensure a satisfactory design.

Multicriteria optimization has its roots in late-nineteenth-century welfare economics, in the works of Edge worth and Pareto. A mathematical description is as follows:

$$\min_{x \in C} F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} \qquad \ldots (MOP)$$

Where n >= 2 and

$$C = \{x : h(x) = 0,\ g(x) \leq 0,\ a \leq x \leq b\}$$

Denotes the feasible set constrained by equality and inequality constraints and explicit variable bounds. The space in which the objective vector belongs is called the objective space and image of the feasible set under **F** is called the attained set.

The scalar concept of ``optimality" does not apply directly in the multiobjective setting. A useful replacement is the notion of Pareto optimality. Essentially, a vector $x^* \in C$ is said to be Pareto optimal for (MOP) if all other vectors $x \in C$ have a higher value for at least one of the objective functions $f_i(\cdot)$, or else have the same value for all objectives. Formally speaking, we have the following definition:

A point $x^* \in C$ is said to be (glob ally) Pareto optimal or a (globally) efficient solution or a non-dominated or a non-inferior point for (MOP) if and only if there is no $x \in C$ such that $f_i(x) \leq f_i(x^*)$ for all $i \in \{1, 2, \ldots, n\}$, with at least one strict inequality.

Pareto optimal points are also known as efficient, non-dominated, or non-inferior points.

We can also speak of locally Pareto optimal points, for which the definition is the same as the one just given, except that we restrict attention to a feasible neighborhood of x^*. That is, if $B(x^*, \delta)$ denotes a ball of radius δ around the point

We can also speak of locally Pareto optimal points, for which the definition is the same as the one just given, except that we restrict attention to a feasible neighborhood of x^*. That is, if $B(x^*, \delta)$ denotes a ball of radius δ around the point x^*, we require that for some $\delta > 0$, there is no $x \in C \cap B(x^*, \delta)$ such that

$$f_i(x) \leq f_i(x^*), \qquad \text{for all } i = \{1, 2, \ldots, n\}$$

with at least one strict inequality.

Typically, there is an entire curve or surface of Pareto points, whose shape indicates the nature of the tradeoff between different objectives.

2.5.1. Solution Techniques

The multiobjective problem is almost always solved by combining the multiple objectives into one scalar objective whose solution is a Pareto optimal point for the original MOP. Most algorithms have been developed in the linear framework (i.e. linear objectives and linear constraints), but the techniques described below are also applicable to nonlinear problems.

2.5.2. Minimizing Weighted Sums of Functions

A standard technique for MOP is to minimize a positively weighted convex sum of the objectives, that is,

$$\sum_{i=1}^{n} \alpha_i f_i(x), \qquad \alpha_i > 0, \quad i = 1, 2, \ldots, n.$$

It is easy to prove that the minimizes of this combined function is Pareto optimal. It is up to the user to choose appropriate weights. Until recently, considerations of computational expense forced users to restrict themselves to performing only one such minimization. Newer, more ambitious approaches aim to minimize convex sums of the objectives for various settings of the convex weights, therefore generating various points in the Pareto set. Though computationally more expensive, this approach gives an idea of the shape of the

Pareto surface and provides the user with more information about the trade-off among the various objectives. However, this method suffers from two drawbacks. First, the relationship between the vector of weights and the Pareto curve is such that a uniform spread of weight parameters rarely produces a uniform spread of points on the Pareto set. Often, all the points found are clustered in certain parts of the Pareto set with no point in the interesting ``middle part" of the set, thereby providing little insight into the shape of the trade-off curve. The second drawback is that non-convex parts of the Pareto set cannot be obtained by minimizing convex combinations of the objectives (note though that non-convex Pareto sets are seldom found in actual applications).

2.5.3. Homotopy Techniques

Homotopy techniques aim to trace the complete Pareto curve in the bi-objective case (n=2). By tracing the full curve, they overcome the sampling deficiencies of the weighted-sum approach. The main drawback is that this approach does not generalize to the case of more than two objectives. For more information, see Rao and Papalambros [7] and Rakowska, Haftka, and Watson [6].

2.5.4. Goal Programming

In the goal programming approach, we minimize one objective while constraining the remaining objectives to be less than given target values. This method is especially useful if the user can afford to solve just one optimization problem. However, it is not always easy to choose appropriate ``goals" for the constraints. Goal programming cannot be used to generate the Pareto set effectively, particularly if the number of objectives is greater than two.

2.5.5. Normal-Boundary Intersection (NBI)

The normal-boundary intersection method uses a geometrically intuitive parameterizations to produce an even spread of points on the Pareto surface, giving an accurate picture of the whole surface. Even for poorly scaled problems (for which the relative scaling on the objectives are vastly different), the spread of Pareto points remains uniform. Given any point generated by NBI, it is usually possible to find a set of weights such that this point minimizes a weighted sum of objectives, as described above. Similarly,

it is usually possible to define a goal programming problem for which the NBI point is a solution. NBI can also handle problems where the Pareto surface is discontinuous or non-smooth, unlike homotopy techniques. Unfortunately, a point generated by NBI may not be a Pareto point if the boundary of the attained set in the objective space containing the Pareto points is no convex or `folded' (which happens rarely in problems arising from actual applications).

NBI requires the individual minimizes of the individual functions at the outset, which can also be viewed as a drawback.

NBI was developed by Das and Dennis ([8], [1]). A public domain Mat lab 4.2 implementation of NBI.

2.5.6. Multilevel Programming

Multilevel programming is a one-shot optimization technique and is intended to find just one ``optimal" point as opposed to the entire Pareto surface. The first step in multilevel programming involves ordering the objectives in terms of importance. Next, we find the set of points $x \in C$ for which the minimum value of the first objective function is attained. We then find the points in this set that minimize the second most important objective. The method proceeds recursively until all objectives have been optimized on successively smaller sets.

Multilevel programming is a useful approach if the hierarchical order among the objectives is of prime importance and the user is not interested in the continuous trade-off among the functions. However, problems lower down in the hierarchy become very tightly constrained and often become numerically infeasible, so that the less important objectives have no influence on the final result. Hence, multilevel programming should surely be avoided by users who desire a sensible compromise solution among the various objectives.

CHAPTER.3 A SIMPLE GA OPTIMIZATION ALGORITHM

3.1. The GA Algorithm

The GA algorithm used in the following example is based almost exactly on the description given on the previous page. The population size will be 4, and strings of bits of length 5 will be used. A crossover probability of 0.6 is assumed and a mutation probability of 0.001. With such a low chance of mutation, it does not occur in the following example.

3.1.1. The Optimization Problem

The problem is simply stated. Find the maximum value of the following function:

$$y = -x^2 + 8x + 15$$

In order to make things easy for us, we will assume that the maximum is between 0 and 25 (the actual maximum is at x=4) and that the maximum is an integer value.
So with this knowledge in hand, we must now choose a coding scheme for the string. In binary, we can represent integer values in the range [0..31] with a 5 bit string. Some examples of strings in a population may be :

String	Decoded Value
00001	1
00101	5
10110	22

Finally, we must decide on a fitness function, which will give the relative fitness values. Now the simplest method to employ here is to use the decoded x value to calculate the y coordinate and use the y coordinate as the fitness rating. Then, the fitness value for string i, as a percentage, will be the y value at i divided by the sum of all the y values for every string.

$$\text{Fitness Value}_i = \frac{f_i}{\Sigma f}$$

Fitness Function for String i in the String Population

For example, say we have a function $y=x^2$ and we are trying to find the maximum value of the function between [0...31]. Then the following strings would have the relative fitness's indicated below:

String	x Value	f(x)	Relative Fitness
00101	5	25	0.04
01101	13	169	0.25
10110	22	484	0.71

In reality, since the value of the function we want to minimise can take on negative values, the fitness function is slightly more complex than the one used above. However, in essence, the two remain equivalent.

3.1.2. Running the GA : Results

The first Iteration:

Firstly, we need to create a random population of strings. Say we start with the following:

String Population
00010
00111
10110
01011

Now we perform selection. The fitness value of each string is calculated and the strings are selected the following number of times:

String	x Value	f(x)	Relative Fitness Value	Nr of Selections
00010	2	27	0.35	1
00111	7	22	0.34	2
10110	22	-293	0.008	0
01011	11	-18	0.30	1

With these selections, our mating pool now looks like this:

String Population
00111
00010
00111
01011

Finally, the crossover probabilities need to be calculated (two crossovers need to be performed to create a new population of two). The GA calculates that it should perform splicing twice on two sets of randomly selected genes. Crossover performs the following to create the new population:

Mating Pool Strings	New Population
0001|0 0011|1	00011 00110
01|011 00|010	01010 00011

So, at the end of the first iteration, our new population looks like the following:

String Population	x Value
00011	3
00110	6
01010	10
00011	3

So, even after one iteration, with no knowledge except for the relative fitness value, the GA has begun to quickly converge on the optimum value of 4. This is startling, considering the GA knows nothing about the problem space in which it searches. It is effectively blind. Yet, just by examining a measure of goodness, having a large number of points to examine simultaneously and having a large amount of randomization thrown in, the GA efficiently searches the problem space for possible answers.

3.2. Schemata: The Building Blocks of GAs

3.2.1. Why do Genetic Algorithms Work the Way They Do?

GAs seems to perform wonders in practice, but many demand solid mathematical proof that search algorithms perform to the required expectations. What will follow will only be the simplest of introductions to the mathematical foundations of GAs. For a more comprehensive coverage, see (Holland, 1975) and (Goldberg, 1989).
In order to put GAs on some firm theoretical footing, the idea of notions, or building blocks, needs to be introduced. The basic idea behind building blocks is that very fit individuals in a population pass on high performance notions to their children. These notions take the form of substrings. It stands to reason that strings with a high fitness value must contain a substring that is a primary cause of such a high fitness. Thus, even though crossover may splice the string into two, there may be a good chance that the highly fit substring is passed on to the children. These highly fit substrings are known as building blocks.

3.2.2. Schemata

Schemata are templates of strings that describe similarity between certain sets of strings. In order to define a schema, the following alphabet is used:

Binary Alphabet	Meaning
0	Binary 0 in String
1	Binary 1 in String
*	Don't Care Term

The alphabet {0, 1,*}, can be used to represent any pattern of binary substrings we wish. For example:

Schema	Matching Strings
*1111	01111
	11111
*010*0	101010
	001010
	101000
	001000
**1*0	00100
	.....

Now, say we were experimenting with the GA from the previous optimization example. It is fairly obvious from the first iteration that the schema 00*** may produce high performance substrings. There are other schemata that can be hypothesized from this GA. So, much of the power of a GA revolves around its ability to process these building blocks in such a way as to use them to create fitter and fitter strings. Remember that Holland has suggested that for n strings in a population, n^3 substrings, and hence schemata, are usefully processed per iteration.

It is also useful to look at how our basic GA operators affect the processing of schemata. In reproduction, the effect is obvious. Fitter strings get selected for the mating pool more than weaker strings, and thus fitter schemata have a greater chance of being involved in the creation of the next generation than their weaker counterparts.

Crossover has a huge impact on the GA building blocks. Obviously, every time crossover occurs, there is a chance that useful schemata might be destroyed by the splicing process. This is one of the main reasons why crossover should not be performed 100% of the time. Look at the following two schemata. We define the length of a schema to be the distance between the first and last specific string position.

Schemata	Length
0*	0
1**1**	3
*0*1*1	4

Remember the way single point crossover functions. It is intuitive to see that useful schemata of long length are far more easily disrupted than schemata of short length.
We define the order of a schema to be the number of fixed positions in the schema.

Schema	Order
*0***	1
10*0**	3
111**0	4

The higher the order of a schema, the more specific the schema becomes. Obviously, 100*1 is far more specific than **0**. Again, with crossover, it is obvious that schemata with small orders have a less likely chance of being disrupted than schemata with high orders.
The affect of mutation on schemata is not difficult to determine. A high mutation rate will badly disrupt schemata, which is why a very low mutation rate is always advised for most GAs.

3.3. Advanced GA Operators

3.3.1. The Operators

Here's a list of some of the more advanced techniques being used and experimented with today:

- Dominance & Diploidy.
- Inversion & Reordering.
- Niche & Speciation.
- The Islands Model.
- Spatial Mating.

3.3.2. Uses of GAs in the Real World

Although the previous pages dealt with GAs solely as a optimization technique, there are a huge diversity of fields using GA technology for all sorts of different applications. Listed below is just a small sampling of the staggering number of applications that put GAs to use:

1. Criminal Suspect Recognition.
2. Music Composition.
3. Construction and Training of Neural Networks.
4. Scheduling Problems (The Traveling Salesman Problem).
5. Games Playing.
6. Prisoner's Dilemma.
7. Earthquake Epicenter Detection.
8. Structural Optimization.
9. Function Optimization.
10. Database Query Optimization.
11. Aircraft Design.
12. Determination of Protein Structures.

3.3.3. Criminal Suspect Recognition

One of the most novel (and copyrighted) uses of GAs to date has been created by Caldwell and Johnson (Caldwell, 1991). This system is used to help witnesses reconstruct facial depictions of criminals. This system is based on the fact that a witness

may be able to easily identify a suspect visually, but the ability to describe and recall a face in order to make a sketch is much more difficult.

The system has a large library of basic facial features. For example, the system contains images of noses, foreheads, ears, etc. The system contains the following building blocks used to create faces.

Facial Building Blocks
Foreheads
Eyes and the Separation Distances
Noses
Mouths
Chins

The system uses a 35 bit binary string to encode the features, and creates an initial population of 20 strings (faces). The witness studies the 20 faces, ranking each one (from 0 to 9), which serves as the fitness value. Then, a new generation is created using selection, crossover and mutation.

This GA has one feature, however, not present in others. The witness can lock a facial feature if he or she desires. Thus, if the eyes look okay, the witness can prevent the eyes from changing in future generations. This dramatically limits the problem space that the GA is required to search. The researchers have reported that convergence on a specific face can occur in as little as 20 generations, which is impressive considering that the GA has to search a problem space consisting of over 34 billion different faces.

3.3.4. Music Composition

Horner and Goldberg (1989), produced a paper describing a GA they had developed for composing music. This system used a GA to take an initial music pattern as input and then use the GA to gradually transform this input into a specified output pattern. The important point to note here is that the system has a series of starting notes which it must transform into a known series of final notes. The operations to be performed on the input pattern were discovered using a GA.

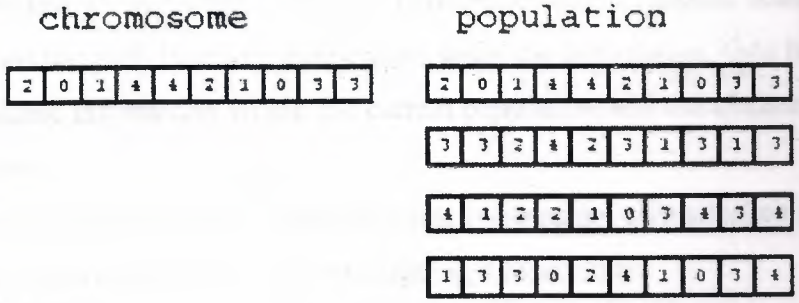
The GA did not work with the musical notes themselves. Instead, it cleverly worked with the transformational operations. So, for example, a substring in one of the GAs strings may represent an operation to "add a note", turning an initial four note pattern into a five note one. There were a number of transformational operators handled by the GA.

The GA in question used binary strings, with certain binary patterns mapping to transformational operations. The length of the strings was directly related to the length of the music to be produced. The GA used mutation, selection and crossover.

CHAPTER 4. OPTIMIZATION OF STRUCTURING ELEMENTS BY GENETIC ALGORITHMS

Optimization is an important issue in image and signal processing. The cost function can be modeled to define a multi-dimensional performance surface. The easiest way to find the set of best solutions is to solve the problem for all possible parameters and retain the ones giving the best result, a method that is termed full-search method. The set of parameters is called solution vector and in many cases it is not possible to compute all solution vectors because their number is simply too large. In these cases, iterative methods are better suited.

In the present case, it is not possible to use full-search algorithms. First of all, the search space is incredibly large: consider a (small) support of a mask of 5 $\delta_1(x)$ 5 pixels for the definition of 2^{25}. The total number of possible arbitrary shapes is (Figure 4.1)



(Figure 4.1)

And even much larger for bigger image supports. It is therefore out of question to explore the entire set of possible solution vectors. The use of coarse to fine strategies can sometimes be helpful in such situations, when a coarser representation of the problem is possible, which is not the case case.

4.1. Basic principles

It is not the intention of this chapter to derive the formal framework for GAs, but rather to provide the basic principles so that the algorithm can be understood. The inquiry for

robust search has made genetic algorithms become fundamentally different from classical algorithms. The differences are based on four principles:

1. GAs uses a coded representation of the parameters, not the parameters themselves.
2. GAs search from a population of solution vectors, not a single solution vector.
3. GAs exclusively use values of the function under study, and do not consider auxiliary information, such as the derivative.
4. GAs use probabilistic transition rules, not deterministic rules.

The function parameters - or the ``living being'' - are represented by a structure called chromosome. Genetic algorithms manipulate chromosomes to profit from and exploit similarities between different performing chromosomes. GAs optimizes a population of chromosomes, unlike other methods that optimize only a single solution vector. The probability to select a false solution is reduced by considering several solution vectors of high performance. GAs remains highly general because their optimization is directly based on the function values. There are no limitations set to continuous and derivable functions. Transition rules of GAs are stochastic and not deterministic as in many other algorithms. Yet, there remains an important difference between GAs and random search algorithms, where decisions uniquely based on pure chance guide the exploration. GAs benefits largely form the available information within the current population and use chance only to guide their exploration.

As in any optimization procedure, three associated objects are characteristic for GAs:

1. The environment of the system undergoing optimization.
2. The adaptive plan which determines successive structural modifications in response to the environment.
3. A measure of the performance of different chromosomes in the environment.

The first point and the third point are given by the problem and the task of the GA is to control the mixture of operators that affect the system undergoing optimization. Thus, the workings of the system are conveyed in the adaptive plan which determines what chromosomes arise in response to its environment. A given chromosome performs differently in different environments, it is more or less fit, and it is the adaptive plan's task to produce chromosomes which perform ``well'' (are fit) in the environment confronting it.

4.2. Query Handling as a Complex Optimization Problem

Among all relational operators the most difficult one to process and optimize is the join. The number of alternative plans to answer a query grows exponentially with the number of **join**s included in it. Further optimization effort is caused by the support of a variety of join methods (e.g., nested loop, hash join, merge join in Postgres) to process individual **join**s and a diversity of indices (e.g., r-tree, b-tree, hash in Postgres) as access paths for relations.

The current Postgres optimizer implementation performs a near-exhaustive search over the space of alternative strategies. This query optimization technique is inadequate to support database application domains that involve the need for extensive queries, such as artificial intelligence.

The Institute of Automatic Control at the University of Mining and Technology, in Freiberg, Germany, encountered the described problems as its folks wanted to take the Postgres DBMS as the backend for decision support knowledge based system for the maintenance of an electrical power grid. The DBMS needed to handle large **join** queries for the inference machine of the knowledge based system.

Performance difficulties in exploring the space of possible query plans created the demand for a new optimization technique being developed.

In the following we propose the implementation of a Genetic Algorithm as an option for the database query optimization problem.

4.2.1. Genetic Algorithms (GA)

The GA is a heuristic optimization method which operates through determined, randomized search. The set of possible solutions for the optimization problem is considered as a population of individuals. The degree of adaption of an individual to its environment is specified by its fitness.

The coordinates of an individual in the search space are represented by chromosomes, in essence a set of character strings. A gene is a subsection of a chromosome which encodes

the value of a single parameter being optimized. Typical encodings for a gene could be binary or integer.

Through simulation of the evolutionary operations recombination, mutation, and selection new generations of search points are found that show a higher average fitness than their ancestors.

According to the "comp.ai.genetic" FAQ it cannot be stressed too strongly that a GA is not a pure random search for a solution to a problem. A GA uses stochastic processes, but the result is distinctly non-random (better than random).

Structured Diagram of a GA:

```
-------------------------

P(t)     generation of ancestors at a time t
P''(t)   generation of descendants at a time t

+=========================================+
|>>>>>>>>>>>  Algorithm GA  <<<<<<<<<<<<<|
+=========================================+
| INITIALIZE t := 0                       |
+=========================================+
| INITIALIZE P(t)                         |
+=========================================+
| evaluate FITNESS of P(t)                |
+=========================================+
| while not STOPPING CRITERION do         |
|   +-----------------------------------+ |
|   | P'(t)   := RECOMBINATION{P(t)}    | |
|   +-----------------------------------+ |
|   | P''(t)  := MUTATION{P'(t)}        | |
|   +-----------------------------------+ |
|   | P(t+1)  := SELECTION{P''(t) + P(t)} | |
|   +-----------------------------------+ |
|   | evaluate FITNESS of P''(t)        | |
|   +-----------------------------------+ |
|   | t := t + 1                        | |
+===+=====================================+
```

4.2.2. Genetic Query Optimization (GEQO) in Postgres

The GEQO module is intended for the solution of the query optimization problem similar to a *traveling salesman problem (TSP). Possible query plans are encoded as integer* strings. Each string represents the **join** order from one relation of the query to the next. E. g., the query tree

```
  /\
 /\ 2
/\ 3
4  1
```

is encoded by the integer string '4-1-3-2', which means, first join relation '4' and '1', then '3', and then '2', where 1, 2, 3, 4 are relids within the Postgres optimizer.

Parts of the GEQO module are adapted from D. Whitley's Genitor algorithm.

Specific characteristics of the GEQO implementation in Postgres are:

- Usage of a steady state GA (replacement of the least fit individuals in a population, not whole-generational replacement) allows fast convergence towards improved query plans. This is essential for query handling with reasonable time;
- Usage of edge recombination crossover which is especially suited to keep edge losses low for the solution of the TSP by means of a GA;
- Mutation as genetic operator is deprecated so that no repair mechanisms are needed to generate legal TSP tours.

The GEQO module allows the Postgres query optimizer to support large **join** queries effectively through non-exhaustive search.

4.2.3. Future Implementation Tasks for Postgre SQL GEQO

Work is still needed to improve the genetic algorithm parameter settings. In file `backend/optimizer/geqo/geqo_params.c`, routines `gimme_pool_size` and `gimme_number_generations`, we have to find a compromise for the parameter settings to satisfy two competing demands:

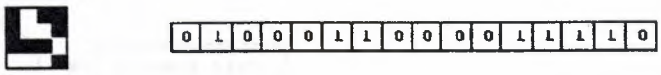
- *Optimality of the query plan*
- Computing time

4.3. Transposition to the optimization of the structuring element

Encoding of the arbitrary shaped structuring function $\delta_1(x)$ is rather straightforward.

Suppose $N \times M$ is defined on a $\delta_1(x)$ binary image mask. Because of its binary character, two alleles are sufficient in the chromosome representation (0 = black and 1 = white). The

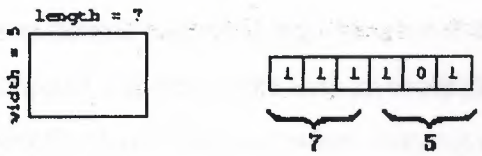
chromosome can be a vector representation of the image when it is scanned in direct video scan (or any other scan) (see Fig. a).



$\delta_1(x)$ chromosome

a) arbitrary shape

If some restrictions on are put, for example that only rectangular structuring elements are allowed, a different chromosome representation becomes possible. In this case, the length and the width can be expressed as a binary number which can be directly coded into the chromosome (Fig. b).



$\delta_1(x)$ chromosome

b) rectangular shape

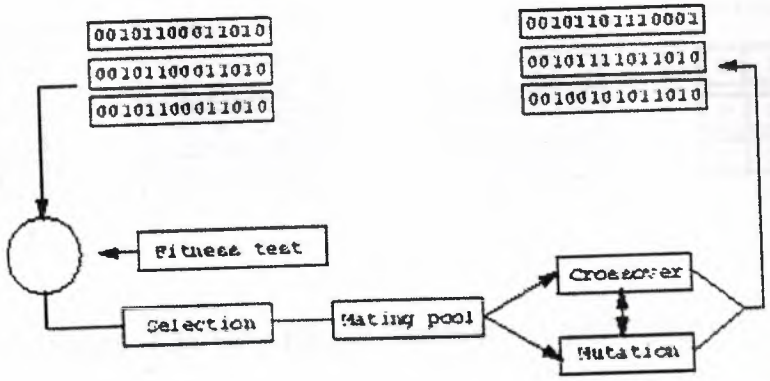


Figure: Encoding of a chromosome: a) arbitrary shape and b) rectangular shape

Other encoding solutions are based on different strategies, for example encoding into continuous vectors. Probably, there is no single technique that works best for all problems and a certain amount of art is involved in selecting a good encoding technique.

The evaluation function provides a fitness measure for a chromosome when applied to the problem to be solved. An evaluation function takes a chromosome as an input, decodes it into its natural representation, applies it to the problem and returns a number or a list of numbers that is a measure of the chromosome's performance on the problem to be solved. Evaluation functions play the same role in genetic algorithms that the environment plays in natural evolution. The interaction of a chromosome with an evaluation functions provides a measure of fitness that the genetic algorithms uses when carrying out reproduction. In the present case, a fitness measure is for example the number of skeleton points or the bit--requirements after entropy coding.

Given these initial components -- a problem, a way of encoding solutions to it, and an objective function that returns a measure of how good any encoding is -- the genetic algorithm carries out a simulated evolution dictated by the adapted plan as follows (Fig 4.2): At the beginning of each time period (generation), the plan's accumulated information about the environment resides in the finite population selected from the set of all attainable structures.

Chromosome	1	2	3	4	5	6	7	8	9	10
Fitness	7	3	2	9	1	5	8	6	4	5
Total	7	10	12	21	22	27	35	41	45	50

Figure (4.2): General scheme of a genetic algorithm

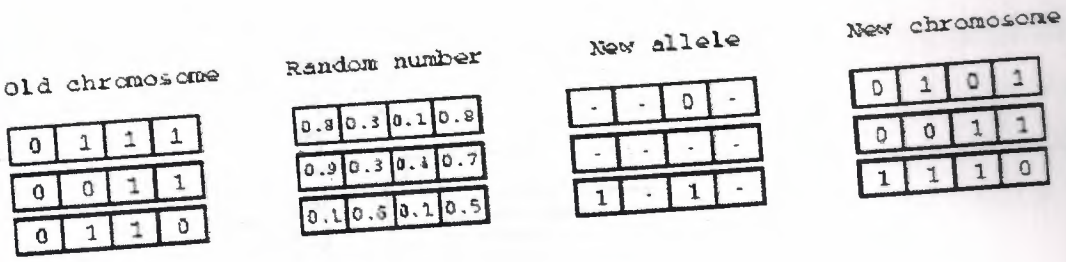


Figure (4.3): Example of a mutation with probability of

4.4.2. Crossover

Unlike mutation that is studied in most biological classes, crossover is not as well known. There are different possible crossovers, e.g. one--point crossover, two--point crossover.

For a crossover operation to take place, two parents are necessary and they will give two new children. A user defined probability value

Fig (4.4)

Determines if the operator is applied to a chromosome. The one-point crossover swaps parts of the two parent genes after a randomly selected point (see Fig.4.4).

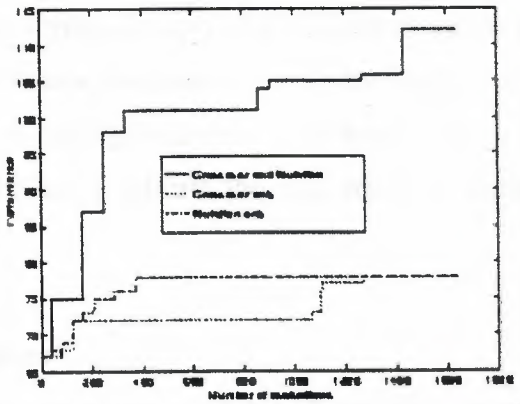


Figure: Example of a one--point crossover.

The two--point crossover operator acts by slicing the chromosome at two randomly selected places and interchanging the information between the two cuts. Hence it is possible to exchange a set of genes in the middle of a chromosome.

Crossover represents an important feature of natural evolution - namely the ability of a population of chromosomes to explore its search space in parallel and combine the best findings.

4.4.3. Evaluation of the importance of the genetic operators

Let us analyses the role of the two genetic operators, mutation and crossover. Consider an algorithm with no mutation operator, only a crossover operator. If by any chance an allele is not present in the initial population, there is no way for the crossover operator to recover this allele. It is possible only due to the mutation operator, which is able to locally change the chromosome's value. On the other hand, an algorithm without crossover operator results in a random sequence of structures drawn from the set of all possible structures, equivalent to a random search technique. It is the crossover operator that provides the necessary mechanism to exploit beneficial material originating from two different chromosomes.

4.5. Shape oriented crossover operator One-point shape crossover

*The one-point shape-crossover is composed of two complementary traditional one-*point crossovers of which one is chosen randomly. The first one is applied on the bit-string

representation of the structuring element and the second one is applied on the bit-string representation turned by π_2 in a clockwise sense. In this way, rows and columns are treated equally. One of the cuts in the figure is chosen randomly, e.g. in this example the vertical cut. The different shaded areas indicate the area which is exchanged between the two squares.

4.5.1. Cut-out crossover

The one-point crossovers are incapable of extracting an area within the structuring element. The cut-out crossover determines randomly two pairs of coordinates, defining the corner points $\vartheta 0^{\circ}$ and π_1 of a rectangle. Crossover consists in exchanging the genetic material within the rectangle.

4.5.2. Two-point shape crossover

The traditional two-point crossover cuts the bit-string representation of the chromosome at two positions and exchanges the information in between. As for the one-point crossover, these cuts translate into horizontally oriented cuts in the matrix representation of the structuring element. The two-point shape crossover consists of two traditional two-point crossovers of which one is applied on the structuring element which is turned by Figure (4.5)

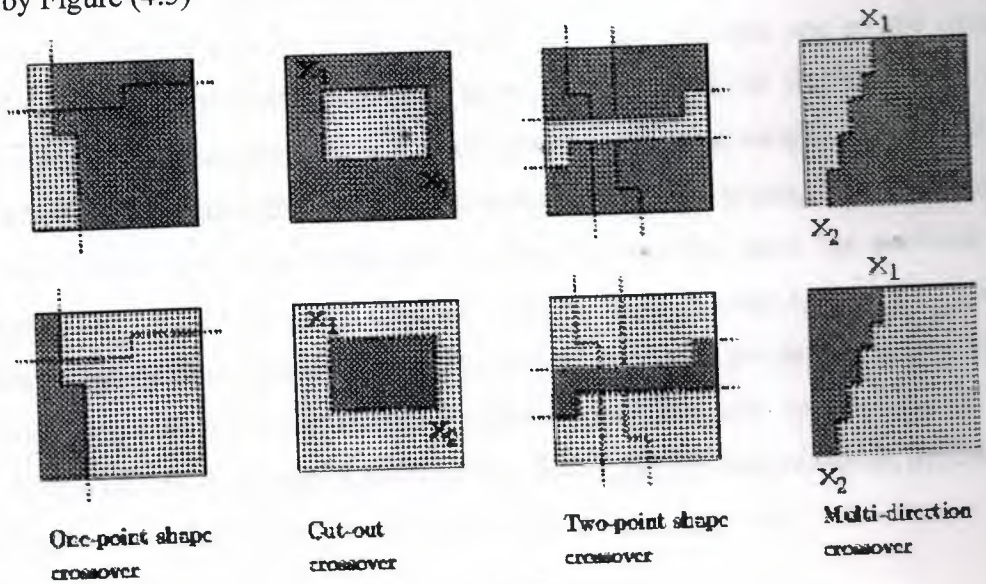


Fig (4.5)

In a clockwise sense. Randomly, the rotated or non-rotated representation determines the crossover site.

4.5.3. Multi-direction crossover

The multi-direction crossover lays a random line through the structuring element which defines the crossover site. The line is constructed by choosing randomly a point 5×5 on any of the four borders of the structuring element. The second point of the line is chosen randomly on any of the remaining three borders. With this operator, it is possible to exchange beneficial material that lies in the corners of the structuring element.

4.5.4. Universal crossover

Four shape-oriented crossovers have been presented. Easily, other configurations may be constructed. For instance, operators that cut out triangles have also been implemented. Of all the different operators, the choice of which one performs best is impossible. Indeed, for different optimization problems different shape-crossover operators may achieve the best result, as illustrated in the next paragraph. The reason is the following: in the case of a binary representation of a decimal number, each gene can be attributed to a weight corresponding to its binary position. Genes with similar weights constitute performant substrings and naturally, they are adjacent one to the other. An example earlier in the thesis Fig (4.6) gave an illustration. In the case of a matrix representation of a structuring element, every gene has the same weight. It is impossible to judge whether rectangular grouped genes, triangularly grouped genes, or genes grouped on rows or columns are more performant. It heavily depends upon the problem to be optimized. Thus, either one of the presented shape-crossovers may be best for a particular problem and all of them should be tried. The logical reaction is the definition of a universal crossover, proposed here, that randomly chooses one crossover operator out of all the possibilities, i.e. either the one-point shape crossover, the cut-out crossover, the two-point shape crossover of the multi-direction crossover is chosen randomly.

The influence of the mutation and crossover operator on the performance of the GA is studied. Both operators applied individually result in bad results, but their combination gives good results. Even though the crossover operator may rarely be directly responsible for the creation of a new more performing chromosome, it is extremely important for exploring the solution space by combining beneficial material between different chromosomes.

In the search for improved performance, the crossover operator should be adapted to the particular problem of optimization of two-dimensional structuring elements. Therefore, it is proposed to replace the traditional crossover operator by a shape-oriented crossover operator. The bit-string representation of a chromosome is replaced by a matrix representation. It allows treating columns and rows of a structuring element in an equal manner, which was not possible with the traditional one-point crossover defined on the bit-string representation. Improving the crossover operator has shown that no-one of the new defined shape-crossovers clearly demonstrated outstanding performances in the majority of the cases. The reason is that all genes in the representation of the structuring element have equal weights and it is impossible to decide what spatial configurations provide above average contributions. Nevertheless, experimental results show that a random combination of the various shape-crossover operators results in significant improvement of the performance of the GA. Such a combination adds genetic diversity to the algorithm and turns out to be very effective.

Various simulation results have demonstrated that the optimal structuring element can indeed be found by GAs in a time much shorter compared to full search approaches. It is also demonstrated that the number of skeleton points can be reduced by a factor of 2 for segmented images by using optimized structuring elements rather than a traditional Skeleton decomposition based on the chessboard mask.

$$I_i = -\mathrm{lb} P_i \qquad [\text{bits}] \tag{8.1}$$

An inherent problem in genetic algorithms is to know when to stop with the evolution. Different strategies can be employed, all of them being more or less heuristic. In the proposed application, the number of cycles that the GA is to run through has been fixed experimentally. Another solution could be the analysis of the evolution curve; if the rate of

increase in performance decreases with respect to a sliding window, it could indicate that the maximal performance is attained.

CHAPTER.5 APPLICATIONS OF GENETIC ALGORITHMS

5.1. Brief Overview

GAs was introduced as a computational analogy of adaptive systems. They are modeled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators such as mutation and recombination (crossover). A fitness function is used to evaluate individuals, and reproductive success varies with fitness.

The Algorithms

1. Randomly generate an initial population M(0)
2. Compute and save the fitness u(m) for each individual m in the current population M(t)
3. Define selection probabilities p(m) for each individual m in M(t) so that p(m) is proportional to u(m)
4. Generate M(t+1) by probabilistically selecting individuals from M(t) to produce offspring via genetic operators
5. Repeat step 2 until satisfying solution is obtained.

The paradigm of GAs described above is usually the one applied to solving most of the problems presented to GAs. Though it might not find the best solution. more often than not, it would come up with a partially optimal solution.

5.2. Who can benefit from GA

Nearly everyone can gain benefits from Genetic Algorithms, once he can encode solutions of a given problem to chromosomes in GA, and compare the relative performance (fitness) of solutions. An effective GA representation and meaningful fitness evaluation are the keys of the success in GA applications. The appeal of GAs comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. GAs is useful and efficient when

- The search space is large, complex or poorly understood.

- Domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space.
- No mathematical analysis is available.
- Traditional search methods fail.

The advantage of the GA approach is the ease with which it can handle arbitrary kinds of constraints and objectives; all such things can be handled as weighted components of the fitness function, making it easy to adapt the GA scheduler to the particular requirements of a very wide range of possible overall objectives.

GAs has been used for problem-solving and for modeling. GAs is applied to many scientific, engineering problems, in business and entertainment, including:

1. Optimization: GAs have been used in a wide variety of optimization tasks, including numerical optimization, and combinatorial optimization problems such as traveling salesman problem (TSP), circuit design [Louis 1993] , job shop scheduling [Goldstein 1991] and video & sound quality optimization.
2. Automatic Programming: GAs have been used to evolve computer programs for specific tasks, and to design other computational structures, for example, cellular automata and sorting networks.
3. Machine and robot learning: GAs has been used for many machine- learning applications, including classification and prediction, and protein structure prediction. GAs have also been used to design neural networks, to evolve rules for learning classifier systems or symbolic production systems, and to design and control robots.
4. Economic models: GAs has been used to model processes of innovation, the development of bidding strategies, and the emergence of economic markets.
5. Immune system models: GAs has been used to model various aspects of the natural immune system, including somatic mutation during an individual's lifetime and the discovery of multi-gene families during evolutionary time.
6. Ecological models: GAs have been used to model ecological phenomena such as biological arms races, host-parasite co-evolutions, symbiosis and resource flow in ecologies.

7. Population genetics models: GAs has been used to study questions in population genetics, such as "under what conditions will a gene for recombination be evolutionarily viable?"
8. Interactions between evolution and learning: GAs has been used to study how individual learning and species evolution affect one another.
9. Models of social systems: GAs has been used to study evolutionary aspects of social systems, such as the evolution of cooperation [Chughtai 1995], the evolution of communication, and trail-following behavior in ants.

5.3. GA on optimization and planning: Traveling Salesman Problem

The TSP is interesting not only from a theoretical point of view, many practical applications can be modeled as a traveling salesman problem or as variants of it, for example, pen movement of a plotter, drilling of printed circuit boards (PCB), real-world routing of school buses, airlines, delivery trucks and postal carriers. Researchers have tracked TSPs to study bimolecular pathways, to route a computer networks' parallel processing, to advance cryptography, to determine the order of thousands of exposures needed in X-ray crystallography and to determine routes searching for forest fires (which is a multiple-salesman problem partitioned into single TSPs). Therefore, there is a tremendous need for algorithms.

In the last two decades an enormous progress has been made with respect to solving traveling salesman problems to optimality which, of course, is the ultimate goal of every researcher. One of landmarks in the search for optimal solutions is a 3038-city problem. This progress is only party due to the increasing hardware power of computers. Above all, it was made possible by the development of mathematical theory and of efficient algorithms. Here, the GA approach is discussed.

There are strong relations between the constraints of the problem, the representation adopted and the genetic operators that can be used with it. The goal of traveling Salesman Problem is to devise a travel plan (a tour) which minimizes the total distance traveled. TSP is NP-hard (NP stands for non-deterministic polynomial time) - it is generally believed cannot be solved (exactly) in time polynomial. The TSP is constrained:

- The salesman can only be in a city at any time

- Cities have to be visited once and only once.

When GAs applied to very large problems, they fail in two aspects:

1. They scale rather poorly (in terms of time complexity) as the number of cities increases.
2. The solution quality degrades rapidly.

5.3.1. Failure of Standard Genetic Algorithm

To use a standard GA, the following problems have to be solved:

- A binary representation for tours is found such that it can be easily translated into a chromosome.
- An appropriate fitness function is designed, taking the constraints into account.

1. Non-permutation matrices represent unrealistic solutions, that is, the GA can generate some chromosomes that do not represent valid solutions. This happens:

- In the random initialization step of the GA.
- As a result of genetic operators (mutation and crossover).

Thus, permutation matrices are used. Two tours including the same cities in the same order but with different starting points or different directions are represented by different matrices and hence by different chromosomes, for example:

tour (23541) = tour (12354)

2. An proper fitness function is obtained using penalty-function method to enforce the constraints.

However, the ordinary genetic operators generate too many invalid solutions, leading to poor results. Alternative solutions to TSP require new representations (Position Dependent Representations) and new genetic operators.

5.3.2. Evolutionary Divide and Conquer (EDAC)

This approach, EDAC [Valenzuela 1995], has potential for any search problem in which knowledge of good solutions for sub problems can be exploited to improve the solution of the problem itself. The idea is to use the Genetic Algorithm to explore the space of problem subdivisions rather than the space of solutions themselves, and thus capitalize on the near linear scaling qualities generally inherent in the divide and conquer approach.

The basic mechanisms for dissecting a TSP into sub problems, solving the sub problems and then patching the sub tours together to form a global tour, have been obtained from the cellular dissection algorithms of Richard Karp. Although solution quality tends to be rather poor, Karp`s algorithms possess an attractively simple geometrical approach to dissection, and offer reasonable guarantees of performance. Moreover, EDAC approach is intrinsically parallel.

The EDAC approach has lifted the application of GAs to TSP an order or magnitude larger in terms of problem sizes than permutation representations. Experimental results demonstrate the successful properties for EDAC on uniform random points and PCB problems in the range 500 - 5000 cities.

5.4. GA in Business and Their Supportive Role in Decision Making

Genetic Algorithms have been used to solve many different types of business problems in functional areas such as finance, marketing, information systems, and production/ operations. Within these functional areas, GAs has performed a variety of applications such as tactical asset allocation, job scheduling, machine-part grouping, and computer network design.

5.4.1. Finance Applications

Models for tactical asset allocation and international equity strategies have been improved with the use of GAs. They report an 82% improvement in cumulative portfolio value over a passive benchmark model and a 48% improvement over a non-GA model designed to improve over the passive benchmark.

Genetic algorithms are particularly well-suited for financial modeling applications for three reasons:

1. They are payoff driven. Payoffs can be improvements in predictive power or returns over a benchmark. There is an excellent match between the tool and the problems addressed.
2. They are inherently quantitative, and well-suited to parameter optimization (unlike most symbolic machine learning techniques).

3. They are robust, allowing a wide variety of extensions and constraints that cannot be accommodated in traditional methods."

5.4.2. Information Systems Applications

Distributed computer network topologies are designed by a GA, using three different objective functions to optimize network reliability parameters, namely diameter, average distance, and computer network reliability. The GA has successfully designed networks with 100 orders of nodes.

GA has also been used to determine file allocation for a distributed system. The objective is to maximize the programs' abilities to reference the file s located on remote nodes. The problem is solved with the following three different constraint sets:

1. There is exactly one copy of each file to be distributed.
2. There may be any number of copies of each file subject to a finite memory constraint at each node.
3. The number of copies and the amount of memory are both limited.

5.4.3. Production/Operation Applications

Genetic Algorithm has been used to schedule jobs in a sequence dependent setup environment for a minimal total tardiness. All jobs are scheduled on a single machine; each job has a processing time and a due date. The setup time of each job is dependent upon the job which immediately precedes it. The GA is able to find good, but not necessarily optimal schedules, fairly quickly.

GA is also used to schedule jobs in non-sequence dependent setup environment. The jobs are scheduled on one machine with the objective of minimizing the total generally weighted penalty for earliness or tardiness from the jobs' due dates. However, this does not guarantee that it will generate optimal solutions for all schedules.

GA is developed for solving the machine-component grouping problem required for cellular manufacturing systems. GA provides a collection of satisfactory solutions for a two objective environment (minimizing cell load variation and minimizing volume of inter cell movement), allowing the decision maker to then select the best alternative.

5.4.4. Role in Decision Making

Applying the well established decision processing phase model of Simon (1960), Genetic Algorithms appear to be very well suited for supporting the design and choice phases of decision making.

- In solving a single objective problem, GA designs many solutions until no further improvement (no increase in fitness) can be achieved or some predetermined numbers of generations have evolved or when the allotted processing time is complete. The most fit solution in the final generation is the one that maximizes or minimizes the objective (fitness) function; this solution can be thought of as the GA has recommended choice. Therefore with single objective problems the user of GA is assisted in the choice phase of decision processing.
- When solving multi-objective problems, GA gives out many satisfactory solutions in terms of the objectives, and then allows the decision maker to select the best alternative. Therefore GAs assist with the design phase of decision processing with multi-objective problems.

GAs can be of great assistance for examining alternatives since they are designed to evaluate existing potential solutions as well to generate new (and better) solutions for evaluation. Thus GAs can improve the quality of decision making.

5.5. Learning Robot behavior using Genetic Algorithms

Robot has become such an prominent tools that it has increasingly taken a more important role in many different industries. As such, it has to operate with great efficiency and accuracy. This may not sound very difficult if the environment in which the robot operates remain unchanged, since the behaviors of the robot could be pre-programmed. However, if the environment is ever-changing, it gets extremely difficult, if not impossible, for programmers to figure out every possible behaviors of the robot. Applying robot in a changing environment is not only inevitable in modern technology, but is also becoming more frequent. This has obviously led to the development of a learning robot.

The approach to learning behaviors, which lead the robot to its goal, described here reflects a particular methodology for learning via simulation model. The motivation is that making mistakes on real system can be costly and dangerous. In addition, time constraints may limit the extent of learning in real world. Since learning requires experimenting with behaviors that might occasionally produce undesirable results if applied to real world. Therefore, as shown in the diagram, the current best behavior can be place in the real, on-line system, while learning continues in the off-line system.

Previous studies have shown that knowledge learned under simulation is robust and might be applicable to the real world if the simulation is more general (add more noise and

distortion). If this is not possible, the differences between the real world and the simulation have to be identified.

5.5.1. GAs' Role

Genetic Algorithms are adaptive search techniques that can learn high performance knowledge structures. The genetic algorithms' strength come from the implicitly parallel search of the solution space that it performs via a population of candidate solutions and this population is manipulated in the simulation. The candidate solutions represent every possible behaviors of the robot and based on the overall performance of the candidates, each could be assigned a fitness value. Genetic operators could then be applied to improve the performance of the population of behaviors. One cycle of testing all of the competing behaviors is defined as a generation, and is repeated until a good behavior is evolved. The good behavior is then applied to the real world. Also because of the nature of GA, the initial knowledge does not have to be very good.

5.5.2. Conclusion and Future Work

The system described has been used to learn behaviors for controlling simulate autonomous underwater vehicles, missile evasion, and other simulated tasks. Future work will continue examining the process of building robotic system through evolution. We want to know how multiple behaviors that will be required for a higher level task interact, and how multiple behaviors can be evolved simultaneously. We are also examining additional ways to bias the learning both with initial rule sets, and by modifying the rule set during evolution through human interaction. Other open problems include how to evolve hierarchies of skills and how to enable the robot to evolve new fitness functions as the need for new skill arises.

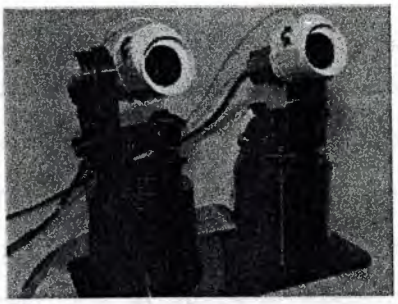
5.6. Genetic Algorithms for Object Localization in a Complex Scene

In order to provide machines with the ability to interact in complex, real-world environments, and sensory data must be presented to the machine. One such module dealing with sensory input is the visual data processing module, also known as the

computer vision module. A central task of this computer vision module is to recognize objects from images of the environment.

There are two different parts to computer vision modules, namely segmentation and recognition. Segmentation is the process of finding interested objects while recognition works to see if the located object matches the predefined attributes. Since images cannot be recognized until they have been located and separated from the background, it is of paramount importance that this vision module is able to locate different objects of interest for different systems with great efficiency.

5.6.1. GA parameters



The task of locating a particular object of interest in a complex scene is quite simple when cast in the framework of genetic algorithms. The brute force-force method for finding an object in a complex scene is to examine all positions and sizes, with varying degrees of occlusion of the objects, to determine whether the extracted sub image matches a rough notion of what is being sought. This method is immediately dismissed as it is far too computational expensive to achieve. The use of genetic methodology, however, can raise the brute-force setup to an elegant solution to this complex problem. Since the GA approach does well in very large search spaces by working only with a sample available population, the computational limitation of the brute-force method using full search space enumeration does not apply.

An experiment was actually carried out based on the following technique, GAs optimized for Portability and Parallelism developed at Michigan State University.

5.6.2. Conclusion and Future Work

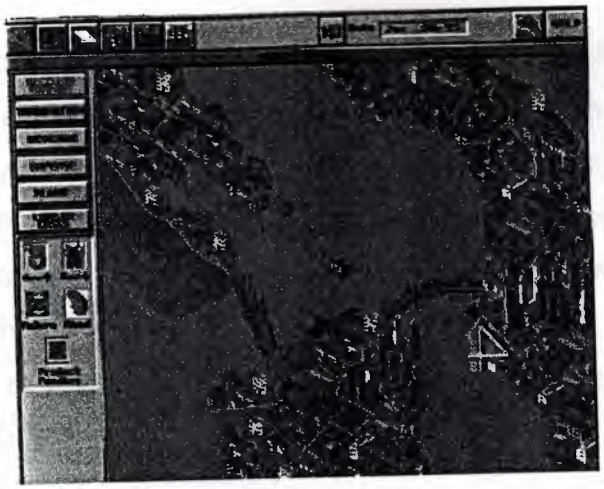
It has been shown that the genetic algorithm perform better in finding areas of interest even in a complex, real-world scene. Genetic Algorithms are adaptive to their environments, and as such this type of method is appealing to the vision community who must often work in a changing environment. However, several improvements must be made in order that GAs could be more generally applicable. Grey coding the field would greatly improve the mutation operation while combing segmentation with recognition so that the interested object could be evaluated at once. Finally, timing improvement could be done by utilizing the implicit parallelization of multiple independent generations evolving at the same time.

5.7. Artificial Life

Genetic algorithms are currently the most prominent and widely used computational models of evolution in artificial-life systems. These decentralized models provide a basis for understanding many other systems and phenomena in the world. Researches on GAs in a life give illustrative examples in which the genetic algorithm is used to study how learning and evolution interact, and to model ecosystems, immune system, cognitive systems, and social systems.

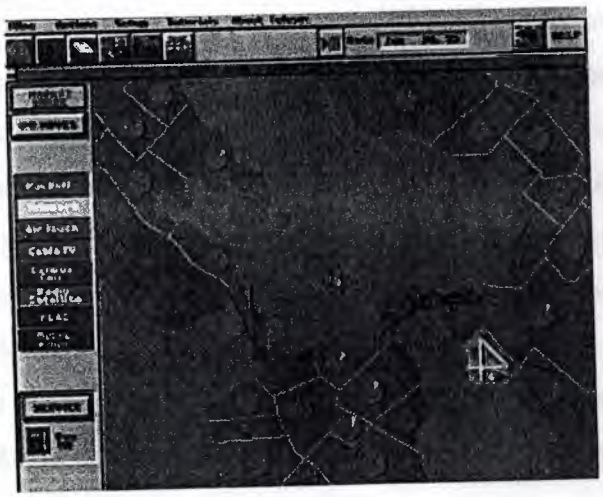
5.7.1. A life on Telecommunication

In the rapidly converging telecommunications industry, technology never stops changing. To assist telecom managers in adapting and prospering during this turbulent period, a business-simulation program, TeleSim, is developed, using artificial life approach. This training tool is designed to provide thought leadership and training for managers facing the challenges of a rapidly changing marketplace.



A TeleSim player acts as a manager in a telecommunications company and pilots the company through a simulated marketplace testing various scenarios and the impact on operations, competitor response and customer behavior. The player confronts with internal staff communications, regulatory penalties, natural disasters, and financial/ technological trade-offs similar to those that managers face in the real world.

In this virtual telecommunications marketplace, the TeleSim player faces seven competitors, which are modeled using adaptive agent technology. The competitive agents interact, adapt to each other, and to the decisions of the player. The competitors learn to execute the best strategic moves as they adapt to the ever changing environment. This emerging and evolving world involves convergence in technology as well as changes in the market and regulations, demonstrating some self-organizing behaviors.



Simulations let people experience and think through the complexity of the business situation and make experiments that they could not possibly do in the real world. People learn to make decisions and gain a better understanding of what present management has been doing. TeleSim allows for a more interactive computer-based approach to scenario development and strategic planning. TeleSim simulates telecommunications businesses, designed as a tool for business planning and management training. In TeleSim, the player learns to develop strategic plans to assess market opportunities and determine an organization's capability for pursuing the dynamics of its strategic direction.

5.8. Vision Intelligence for Precision Farming Using Fuzzy Logic Optimized Genetic Algorithm and Artificial Neural network

Agriculture in developed countries after the Industrial Revolution has tended to favor increases in energy input through the use of larger tractors and increased chemical and fertilizer application. Although this agricultural technology has negative societal and environmental implications, it has supported food for rapidly increasing human population. In western countries, "sustainable agriculture" was developed to reduce the environmental impact of production agriculture. At the same time, the global agricultural workforce continues to shrink; each worker is responsible for greater areas of land. Simply continuing the current trend toward larger and heavier equipment is not the solution. A new mode of thought, a new agricultural technology is required for the future. Intelligent robotic tractors are one potential solution (Noguchi, *et al.*, 1996, 1997). Sensors are an essential part of intelligent agricultural machinery. Machine vision, in particular, can supply information about current crop status, including maturity and weed infestations. The information gathered through machine vision and other sensors such as GPS can be used to create field management schedules for chemical application, cultivation and harvest. The purpose of the study is to develop an intelligent machine vision system for an agricultural mobile robot. The vision system developed is able to simultaneously detect crop rows and gather field information. The vision system uses a Genetic Algorithm (GA) optimized fuzzy logic decision-making system to classify crop and weed material. After segmenting out weeds,

the vision system can estimate the crop height and width using an artificial neural network (ANN).

5.9. Machine Vision Hardware

5.9.1. Vision System

The machine vision system developed uses a CCD camera, a frame grabber and a computer. A monochrome CCD camera serves as the image sensor for the vision system. A near infrared filter (800 nm) is installed on the camera to improve discrimination between the plant material and soil. The camera is mounted on the centerline of the test vehicle, with a 20 degree down angle. The test vehicle is a conventional 115-kW tractor (CASE-IH, 7220) with a modified computer controlled electro hydraulic steering system. In addition to machine vision, the navigation sensors on the vehicle included a geomagnetic direction sensor and DGPS (**Fig. 5.1**).

Fig.5.1 Overview of the test vehicle

Under ideal conditions, the real-time kinematics DGPS has achieved 20-cm or better accuracy.

5.9.2. Camera Calibration

The camera field of view was approximately 4 m x 20 m. Static camera calibration was used to develop a conversion from the camera coordinate system to the a ground coordinate system. The camera calibration method is briefly explained here. On the image

plane the pixel coordinates (U, V) can be represented by a set of homogeneous coordinates (u, v, t):

$$U = \frac{u}{t}, \; V = \frac{v}{t} \qquad (1)$$

If the z-coordinate of the field is on a known constant plane, the mapping of $(x, y, 1)$ in the field into (u, v, t) on the image plane becomes:

$$\begin{pmatrix} u \\ v \\ t \end{pmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad (2)$$

An inverse perspective transformation matrix, which can convert from the pixel coordinates (U, V) to the ground coordinates (x, y), can be represented as follows:

$$x = \frac{x'}{t'}, \; y = \frac{y'}{t'} \qquad (3)$$

$$\begin{pmatrix} x' \\ y' \\ t' \end{pmatrix} = \left[C^{-1}\right] \begin{pmatrix} U \\ V \\ 1 \end{pmatrix} \qquad (4)$$

The center of the front of the tractor was adopted as origin of the ground coordinate system in the study. The transformation matrixes were calculated using a least square method for nine points measured in the range of -2 m to 2m for x-axis, and 0 m to 10 m for y-axis. As the results, the following transformation matrixes were obtained:

1) Perspective transformation matrix:

$$C = \begin{bmatrix} 240.03 & 93.04 & 262.1 \\ -4.51 & -15.85 & 1058.1 \\ 0.0 & 0.37 & 1.0 \end{bmatrix} \qquad (5)$$

2) Inverse perspective transformation matrix:

$$C^{-1} = \begin{bmatrix} 0.4148 & -0.0039 & -104.6095 \\ -0.0003 & -0.2458 & 260.1519 \\ 0.0 & 0.0009 & 0.0345 \end{bmatrix} \quad \textbf{(6)}$$

The root-mean-square (RMS) conversion accuracy from the image coordinate to the ground coordinate was approximately 5.8 cm for the over all field-of-view.

5.9.3. Compression of Image Information

The machine vision system can quickly gather a large amount of information in a short period of time. To reduce the processing load and to decrease the computer memory requirements, the image was compressed by expressing the canopy area as a density within a 20-cm square. The upper image in **Fig. 5.2** was obtained thorough binalization of a raw image in the 4 m x 2m field-of-view. The lower image in **Fig. 5.2** shows the canopy area represented as a gray level calculated through the transformation mentioned above. Because the actual area of the each pixel is spatially different, the conversion coefficient from a pixel size to the actual area was calculated for each of the 20-cm squares.

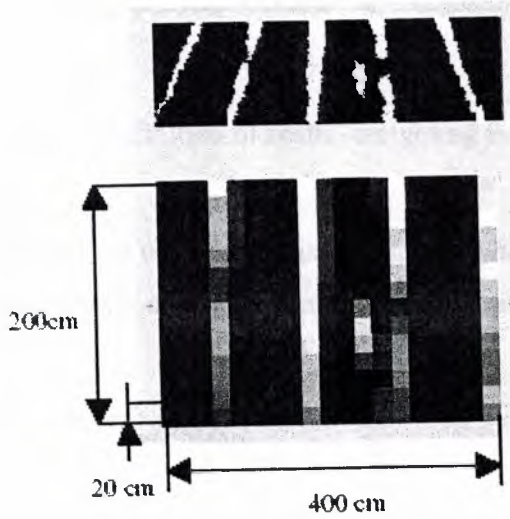


Fig.5.2 Conversion and compression of
The image information

5.10. Classification of Crop and Weed by A combination of Fuzzy Logic and Genetic Algorithm

5.10.1. Classifier of Crop and Weed Based on Fuzzy Logic

Fuzzy logic, which can deal with ambiguous information, was used as the crop and weed classifier in this study. Fuzzy logic can represent the solution in terms of a probability. Therefore, three fuzzy input parameters *x1* to *x3* were chosen to classify crop and weed areas in the field-of-view. Since crop rows are almost parallel with the *y*-axis of the field-of-view during the travel, we chose the average of the gray level for each row in the image as *x1*, the standard deviation of that as *x2*, and the spatial weighting factor as *x3*. As shown in **Fig. 5.3**,

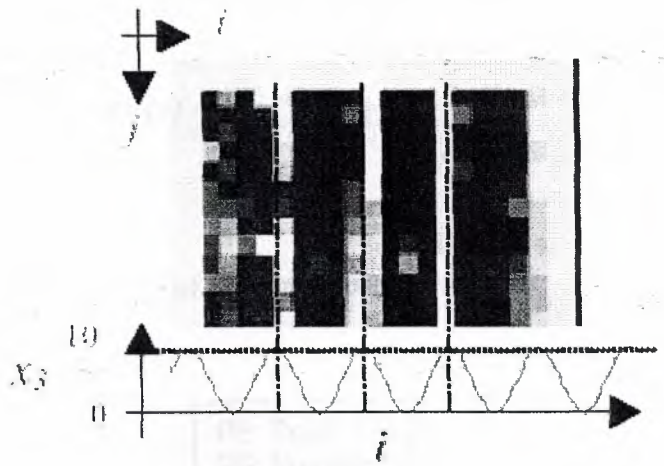


Fig.5.3 Definition of spatial weighting factor, x_3

The spatial weighting factor, *x3* utilized the fixed crop row width as priori information. The fuzzy logic input parameters are normalized and can be expressed as follows:

$$x_1 = \sum g_{i,j} / n_j \qquad (7)$$

$$x_2 = 100 \sqrt{\frac{\sum_j (g_j - x_1)^2}{n}} \Big/ x_1 \qquad (8)$$

$$x_3 = 50\sin(i-\tau)(2\pi/4.5)+50 \qquad (9)$$

The input and output membership functions and the fuzzy rules are shown in **Fig. 5.4**, **Fig. 5.5** and **Table3.1**.

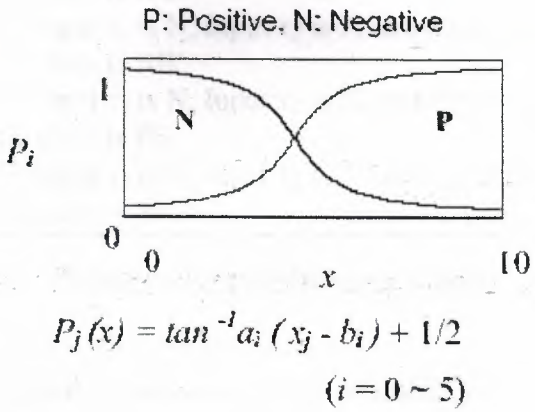


Fig.5.4 Input membership function

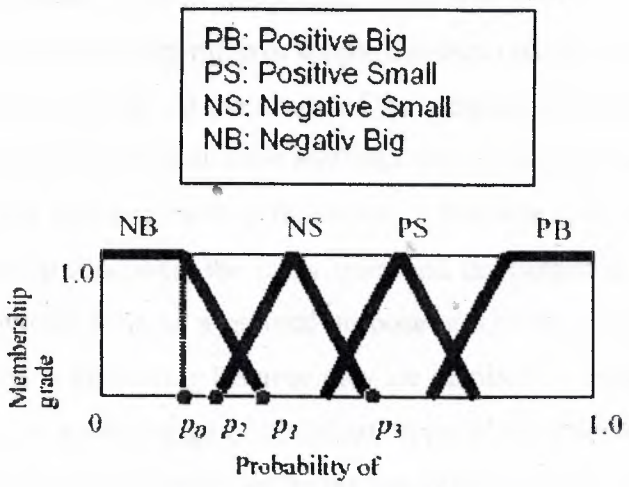


Fig.5.5 Output membership function

Rule 1: IF Input x_1 is P, Input x_2 is P, and Input x_3 is P
THEN the class is PS.
Rule 2: IF Input x_1 is P, Input x_2 is P, and Input x_3 is N
THEN the class is NS.
Rule 3: IF Input x_1 is P, Input x_2 is N, and Input x_3 is P
THEN the class is PB.
Rule 4: IF Input x_1 is P, Input x_2 is N, and Input x_3 is N
THEN the class is PS.
Rule 5: IF Input x_1 is N, Input x_2 is P, and Input x_3 is P
THEN the class is NS.
Rule 6: IF Input x_1 is N, Input x_2 is P, and Input x_3 is N
THEN the class is NB.
Rule 7: IF Input x_1 is N, Input x_2 is N, and Input x_3 is P
THEN the class is PS.
Rule 8: IF Input x_1 is N, Input x_2 is N, and Input x_3 is N
THEN the class is NS.

Table5.1 Temporally fuzzy rules.

The fuzzy logic was designed so as to output the probability of crop from the fuzzy logic classifier.

5.10.2. Optimization of Fuzzy Logic by GA

One of the disadvantages of fuzzy logic when compared to a conventional image classifier (such as the Kmeans algorithm or simple threshold based segmentation) is that the classification accuracy depends on the shape of the membership functions and the fuzzy rules built by a designer. Trial-and-error methods are frequently used to determine the fuzzy logic parameters and membership functions. A GA was used to optimize the fuzzy logic input membership functions, the fuzzy rules and the output membership functions. Holland (1975) proposed GAs as a general-purpose stochastic optimization method for search problems. GAs is interesting because they are inspired by biological evolution and they seem applicable to a wide range of optimization problems (Noguchi and Terao, 1997; Noguchi *et al.*, 1997). The data processed by the algorithm consists of a set (population) of strings that represent multiple points in a search space. A finite length string, in which each bit is called an allele, is defined as a solution (individual) having the objective function value of a point in the search space. The function to be minimized by the algorithm is

converted to a fitness value that determines the probability of the individual undergoing transitional operators. The operators are analogous to the biological terms of crossover, mutation and selection. In the study, coefficients, *ai* and *bi* (*i*= 0 to 5) in the input membership function and *p0* to *p3* in output membership function, which decide the shape of those, were coded in the individuals as shown in **Fig. 5.6**.

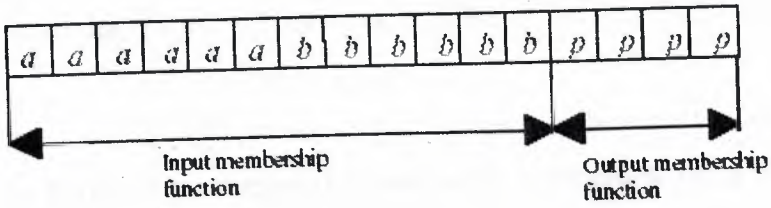


Fig.5.6 Coding of fuzzy parameters to a Chromosome.

The flowchart of the GA for optimizing the fuzzy logic classifier is shown in **Fig. 5.7**.

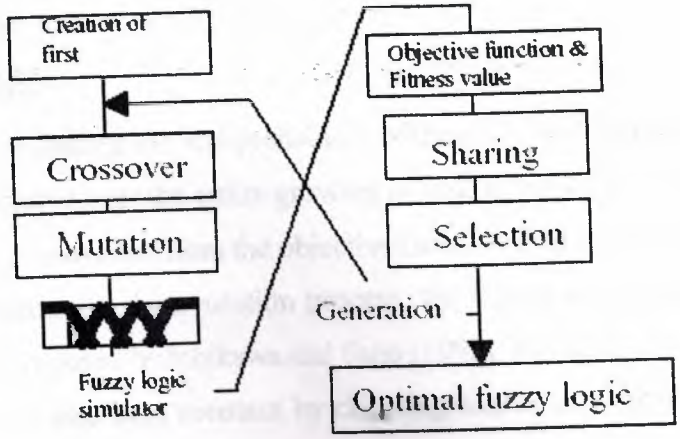


Fig.5.7 Flowchart of GA optimization of Fuzzy logic

Crossover was a random exchange of multiple alleles between the selected mating pairs, which created two offspring. Half of an individual's total number of alleles was exchanged during crossover. Mutation happened with a probability of 1.0 on selected individuals that had not crossed over. All the alleles were increased by a mutation width *De* that was

randomly chosen in the range of -10 to 10. After these transitional operators created new individuals, the survivors were selected from the double-sized tentative population that included the current and newly created individuals. Selection, an analog to natural selection, was conducted by spinning a simulated roulette wheel whose slots had different sizes proportional to the fitness values of the individuals and by an elitist preservation strategy in which the individual with the lowest objective function value was exceptionally chosen.

As shown in Eqn. (10), the adopted objective function was the error function summing up the squared error between the training data and the data calculated by each individuals for all acquired image data.

$$V(c_i) = \sum_{j=0}^{n} \left(\hat{p}_j - p(c_i, j) \right)^2 \qquad \textbf{(10)}$$

Here,

V : Objective function, P: Fuzzy logic probability,

C_i : i-th chromsome, n: Number of training data,

$\hat{p}$: Real probability.

The training data, including the real probability of the crop, was determined manually. For the image data gathered over the entire growing period, n was 1235. The fitness value f for each individual was converted from the objective function $V(u)$. In addition, to maintain the diversity of individuals in the evolution process, the fitness value was corrected using a sharing operation proposed by Ichikawa and Sano (1992). The size of the population for the next generation was also kept constant by choosing half of the tentative population. The results were obtained using the GA parameters of population size m= 20, the number of generations Ng= 500, the probability of crossover Pc = 0.7, and the probability of mutation Pm = 0.3. *William 82* soybeans were chosen as the test crop in the study. **Fig. 5.8** shows the GA optimization process for the fuzzy logic parameters.

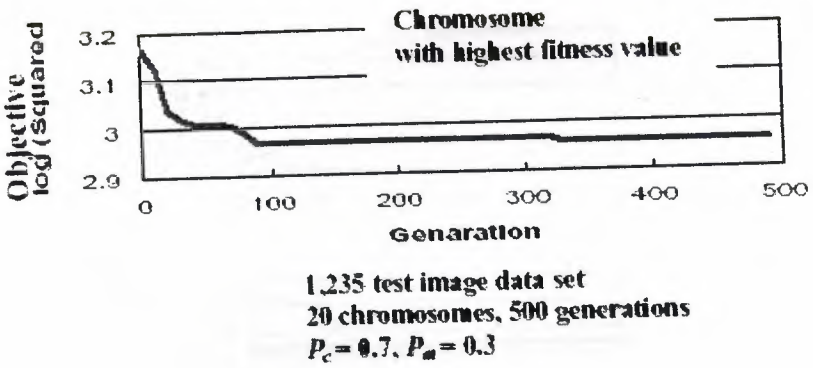


Fig.5.8 the GA/fuzzy logic optimization Process

The objective function value of the individual with the highest fitness value in each generation was represented in the figure. Because it was found that the objective function value decreased with the progress of the generation, and converged in certain value, it was seemed that the developed GA could find out the sub-optimal fuzzy logic.

5.10.3. Results using Fuzzy Logic optimized by GA

The input and output membership functions created by the GA are shown in **Fig. 5.9** and **Fig. 5.10**, respectively.

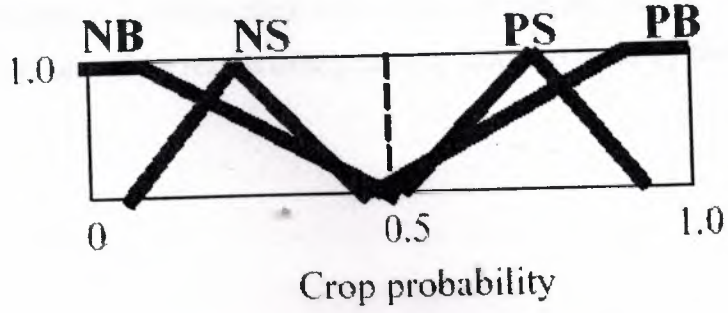


Fig.5.9 Output membership function created By GA.

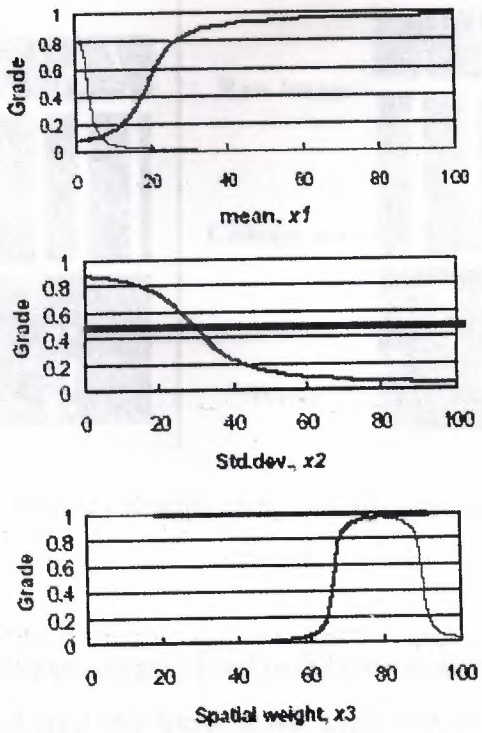


Fig.5.10 Input membership function create

By the GA

The shapes of the membership functions were quite different from fuzzy logic functions conventionally created by a human.

In particular, the GA created unsymmetrical membership seems unique in the general fuzzy logic. **Fig. 5.11** shows one of the results of segmenting crop and weed by the created fuzzy logic.

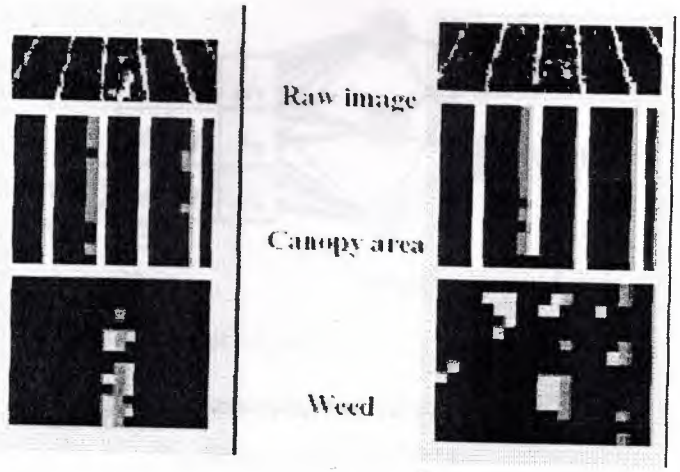


Fig.5.11 Results using fuzzy logic created
By GA

The top figure is a raw image and gray level in the second and the third figures indicate the existence probabilities of crop and weed. It was clear that the fuzzy logic could correctly distinguish crops and weeds in the figure. In fact, accuracy of the segmentation were investigated using the images acquired in six different growth stages covered the entire growing period to verify robustness of the created fuzzy logic. We confirmed that recognition accuracy did not relate with the crop growth, and the weed area could be almost perfectly segmented. It is fair to say that the method developed by combining the fuzzy logic and the GA was appropriate and effective.

5.11. Estimation of Crop Growth Using ANN

5.11.1. Construction of an ANN for crop growth prediction

Crop growth information is important for making fertilzer application decisions as well as for investigating spatial variation in overall yield. A tractor operator qualitatively observes crop conditions during travel and operations in the field; an agricultural mobile robot also has to detect crop growth parameters such as such as crop height and width. A crop prediction method was developed for machine vision. An ANN was utilized to find the relationship between the image pattern and the crop growth parameters. The construction of the ANN is shown in **Fig.5.12**.

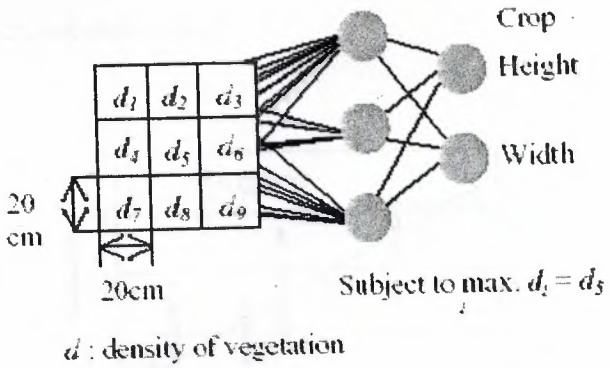


Fig.5.12 Construction of the crop growth Predictor using the NNA.

The ANN is three layers network, which was composed of an input layer, a hidden layer and an output layer. The gray levels of nine 20-cm square tiles were the inputs to the ANN. To identify the individual crop center, *d5*, which was the highest gray level in the neighbors, was adopted as a constraint condition. Training data were randomly chosen from the image data during entire growing period. A back propagation algorithm (BP) was adopted as the ANN training method. A set of 300 training samples were used to build the ANN.

5.11.2. ANN Prediction Accuracy

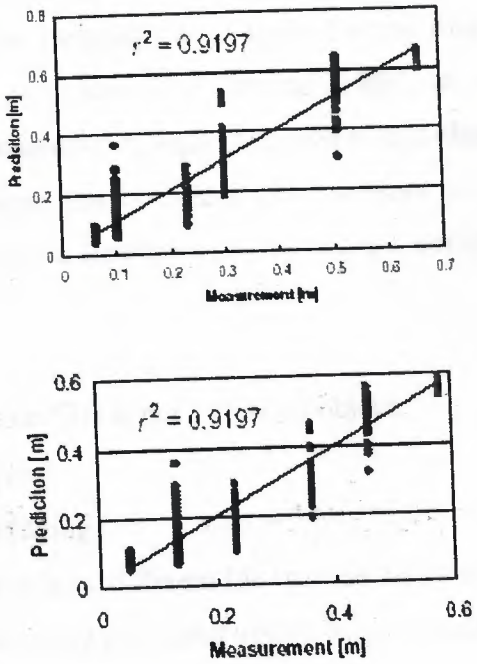


Fig.5.13 Accuracy of the NNA crop growth Predictor.

Fig.5.13 shows the ANN prediction accuracy for crop height and width using the training data. We investigated the relationship between the measured and predicted values through the entire growing period. The *r2* for both crop height and crop width was 0.92 for the training data and about 0.84 for the test data. The high correlation implies that an ANN – machine vision system can be used as a crop prediction sensor. Combining GPS and image data created spatial maps relating the crop height and width. Arc View (Environmental Systems Research Institute, Inc.) Geographic Information System (GIS) software was used to create the maps.

5.12. Artificial Life? Real Life? Are they interchangeable?

Artificial-life programmer claims that, with the help of the increasingly advanced technology, they will soon go beyond merely modeling or simulating living organisms and

actually create life. The claim is not simply that one could design an artificial life with the help of a computer, and then build it out of organic molecules. They claim that one could create living organism simply by programming a computer in a right way. If today's virus is not alive, tomorrows will be. Computer equivalent of worm, frog would soon be rampaging in the networks. This claim is known as "**strong A-life**", as opposed to "**weak A-life**". Obviously there are two schools of thought regarding to this claim.

Foes of **strong-A-life** argue that no matter how advance computer technology would become, life cannot be created simply by programming a computer. They put forward the arguments:

- A computer generated life is not a material object.
- It can not move about
- It is not capable of dying
- Same individual but have different life span on different machines.

A computer generated life is not a material object. When one talks about living organism, it is a kind of material object: something that takes up space and has a mass, a chemical composition, and other physical properties. Material objects are something that satisfy most proposed definitions of life: they take in matter, utilize its energy, and expel its remains in a less ordered form; they have well-defined boundaries; they can reproduce themselves with great accuracy; and so on. On the contrary computer generated life forms do not satisfy these definitions of life.

It can not move about. The only movements one could detect from computer generated organism are contraction and expansion.

It is not capable of dying. Computer generated life forms can not really die. Real organism is made up of molecules arranged in an extremely complex and delicate way. When an organism dies, the arrangement is destroyed. However, the life of artificial life ceases to exit when the machine on which it is running stops. But the program hardly dies since when the machine is turned back on, computer generated life "resurrects".

Same individual but have different life span on different machines. When a program, regarded as the artificial life, is run on two machine, the two instances of the program are supposed to have identical attribute and thus can be assumed to be the same individual.

However, clearly two instances could have different life span if one of the machine stops running before the other. This is absolutely nonsense in the context of real life. It literally means that one individual could die more than once.

Supporters of **strong a-life** obviously think otherwise. One of the more prominent supporter, Christopher Langton, writes that the artificial life created do not live in the medium as we know. It is in a virtual medium where they reside. He further argues that models built could be so real that they would cease to be models of life and become examples of life themselves. He claims that any definition or list of criteria broad enough to include all known biological life will also include certain classes of computer processs and therefore will have to be considered as "actually alive".

CONCLOISON

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^3 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^3 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] ***Genetic Algorithms in Engineering and Computer Science*** , edited by G.Winter, J.Periaux & M.Galan, published by JOHN WILEY & SON Ltd. in 1995.

[2] [Louis 1993] ***Genetic Algorithms as a Computational Tool for Design***, by Sushil J. Louis, in August 1993

[3] ***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)

[4] ***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.

Obtained : Central Library of Imperial College (4 Life Sciences 575.116.12 KOR)

[5] ***Learning Robot Behaviours using Genetic Algorithms*** , by ALAN C.Schultz. Navy Center for Applied Research in Artificial Intellignece.

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

[7] ***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

[8] **http://www.cs.qub.ac.uk/~M.Sullivan/ga/ga3.html**

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

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

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

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