Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Applying the genetic algorithm to optimization problems

H. Wan
Department of Computer Studies, Lingnan College, Hong Kong

Abstract
The application of the genetic algorithm to several problems in optimization is described. The performances in each of these experiments were compared with those of conventional methods. The pros and cons of using the genetic algorithm in the examples are also discussed.

INTRODUCTION

Since the genetic algorithm (GA) was first proposed by Holland[1], it has not been receiving too much attention for decades until the late 80s. It occurred at a time when connectionists were desperate for finding some quick and easy ways to determine the weights on neural networks (Ackley[2], Koza et al.[3]). Having been accepted by AI researchers, its effectiveness and resemblance to biological evolution is gaining more and more admiration and more and more research areas in which GA is applicable are being discovered.

The topic of optimization is at the center of operations research. For unconstrained optimization problems, variants of the steepest descent method play an important role. However, these methods rely heavily on the differentiability of the cost function; but in general, the cost function may even be discontinuous

over the search space. This is where a heuristic method like GA comes into play.

In this paper, the general method of GA is explained in Section 2. The experiment of applying GA to simple unconstrained optimization problems is illustrated in Section 3. A simple way of tackling the Traveling Salesman Problem is described in Section 4 and an experiment of using GA to determine the membership functions of a fuzzy controller is reported in Section 5. Section 6 gives a few comments on GA and some possible projects for the future are described in the last section.

GENETIC ALGORITHM

Holland, in his first book on GA, adapted Darwin's idea of natural selection to problems in optimization. His algorithm works on a population of chromosomes each of which, is a string of binary bits called *bit vectors* although some researchers suggest that other forms of chromosomes are more appropriate in some particular cases. Normally the dimension of the search space determines the length of the chromosomes. The process of searching is to find the best-fit chromosome. The layout of the bit vector of the best-fit chromosome determines the optimal solution to the objective function.

At the beginning, all chromosomes are chosen in a random way, eg. by using a random number generator to assign 1 or 0 to each of the elements in the bit vector. Each chromosome is evaluated and the set of chromosomes with the highest scores is selected. These chromosomes are allowed to crossover (Figure 1) with each other to produce a set of new chromosomes. The

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

less-fit chromosomes in the previous generation are discarded and replaced by the new chromosomes.

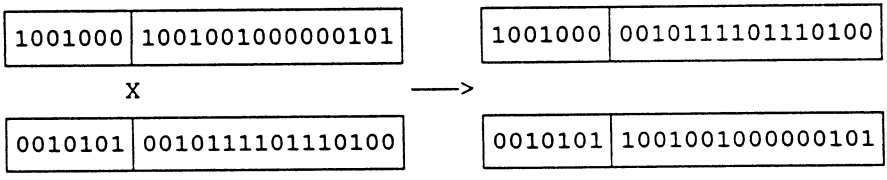

Figure 1. Crossover

Similar to the activities of chromosomes in living cells, transformations of bit vectors by processes like mutations (Figure 2) of genes can also take place on the newly formed chromosomes. (The best-fit chromosomes are left intact in order to save the good genes for the next generation). In more sophisticated systems, other forms of transformation have also been adopted (Hesser[4]).

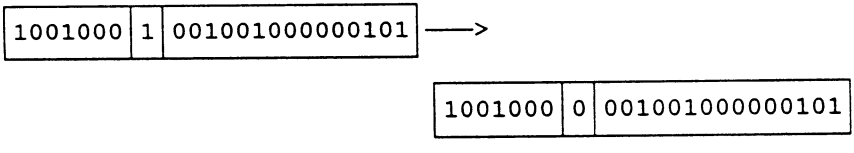

Figure 2. Mutation

When the next generation is ready, the degree of fitness of each chromosome is evaluated and then the process is repeated. The evaluation of chromosomes can be done simply by applying the values represented in the bit vectors to the objective function. Then a tiny sorting process is employed to single out the set of the fittest chromosomes before the next generation of chromosomes is produced.

The optimal solution is usually determined by going through a number of generations. However, the number of generations necessary to determine that the

most-fit chromosome is found is *a priori* unknown. This is one of the drawbacks of GA.

EXPERIMENT 1: UNCONSTRAINED OPTIMIZATION

An arbitrary set of quadratic functions are chosen, eg.

$$f1(x,y) = a1*x*x + a2*y + c1$$
$$f2(x,y) = b1*x*x + b2*y + c2 \quad (1)$$

where the coefficients a1, a2, b1, b2, c1, and c2 are known. The search space is $\mathbf{R}^2$. Hence, the bit vector should be long enough to encode two real numbers x and y. The precision of the solution depends on the number of bits, ie. the length of the chromosomes. But if more bits are used, the time spent on crossover and decoding (changing the bit vectors to usable real numbers) will increase.

The size of the population of chromosomes is also important. For the sake of simplicity, half of the population is replaced when one generation gives birth to the next generation. If there are more chromosomes in the population, a proportional number of chromosomes will be replaced in each generation, taking a longer time to generate (more) new chromosomes.

The percentage of chromosomes allowed to undergo mutations is also critical. In general, there should be less mutations at the beginning with the percentage increasing throughout the process. The idea complies with the general principle of stimulated annealing (Xu et al.[5]).

In this GA experiment, problem (1) above was tried with arbitrary coefficients. The initial generation was a 200-chromosome population. A solution was usually reached after no more than 200 generations. Compared

with conventional numerical methods, this GA does not guarantee that the optimal solution is always found. However, while using GA, problems like differentiability, singularity of the Hessan matrix and propagation errors no longer exist. Besides, problems like multiplicity of zeros can easily be dealt with.

EXPERIMENT 2: TRAVELLING SALESMAN PROBLEM

The travelling salesman problem (TSP) is basically a constrained optimization problem. Given a set of cities and the distances between these cities, the problem is to find the shortest route to visit each of the cities where 1) each city should have been visited by one and only one time, and 2) the route should start and finish at a given city. Though algorithmic approaches like the divide-and-conquer or greedy methods are explained in every book on algorithms, the problem becomes excessively complicated when the number of cities is larger than 10.

In my experiment, the chromosomes were no longer composed of bits, instead, they were taken to be integer strings. Each of the cities under consideration was given an identification number. The city numbers made up the chromosomes. Since the starting and finishing points of the route are the same, the leading and trailing integers in each chromosome should be the same. Therefore, each chromosome represented a particular route. The fitness of the chromosome was measured by the length of the route it encoded.

However, a crossover in the traditional way was not allowed this time. The only transformation used was to relocate the integers (ie. genes). Other activities like

inversion (cutting a small part of a chromosome and pasting it back to where it was, but in the reverse direction) was also applied. A large number of genes of the better-fit chromosome were relocated within the chromosome to produce new chromosomes for the next generation.

In the experiment, with 100 chromosomes and an arbitrary adjacent matrix for 10 cities, most of the near-optimal solutions came about in less than 100 generations. But when a 20-city problem was considered, a population of the same size might take up to 1000 generations to give a fairly optimal solution. On a few occasions, the solutions obtained from GA were not as good as what greedy method could produce.

A better result could be obtained if I used a modified process of crossovers. As Liepins et.al.[6] suggest, for two chromosomes of the form

A = 9 8 4 5 6 7 1 3 2 10
B = 8 7 1 2 3 10 9 5 4 6

if the crossover took place from the 4th gene to the 6th gene,

A = 9 8 4 | 5 6 7 | 1 3 2 10
B = 8 7 1 | 2 3 10 | 9 5 4 6

then each gene has to be changed by the permutations (5, 2), (6, 3) and (7, 10) so that the offsprings will be

A' = 9 8 4 | 2 3 10 | 1 6 5 7
B' = 8 10 1 | 5 6 7 | 9 2 4 3

Hence the first condition is maintained. With such a crossover, a population of 100 chromosomes could reach a near-optimal solution for a TSP of 40 cities in less than 400 generations.

EXPERIMENT 3: FUZZY CONTROL

A fuzzy control is the implementation of a fuzzy inference process. A fuzzy inference process consists of a set of rules (in a *rule base*). For a single-output system, a general format of the rule base is (following Tsuchiya[7]):

Rule 1:	IF Xa = Aa_1	THEN Y = B(a,1)
Rule 2:	IF Xa = Aa_2	THEN Y = B(a,2)
:	:	:
Rule j:	IF Xa = Aa_j	THEN Y = B(a,j)
Rule j+1:	IF Xb = Ab_1	THEN Y = B(b,1)
:	:	:
Rule 2j:	IF Xb = Aa_{2j}	THEN Y = B(b,2j)
:	:	:
Rule k:	IF Xs = As_{ks} AND Xt = At_{kt}	THEN Y = B(st,kskt)
:	:	:
Rule n:	IF Xz = Ab_{zj}	THEN Y = B(z,zj)

where X is the variable of the antecedent and Y is the variable of consequent; A and B are the labels for the antecedent and consequent clauses. These labels may be one of the following:

NL	(negative large)
NM	(negative medium)
NS	(negative small)
ZR	(nearly zero)
PS	(positive small)
PM	(positive medium)
PL	(positive large)

The general format above allows any one antecedent variable to take any possible label (eg. rules 1 to j) as well as any logical combination of variables (eg.

rule k). For the sake of simplicity, the membership function of each of these variables (Xa, .., Xz, Y) may take any triangular shape. The following figure shows all the possibilities:

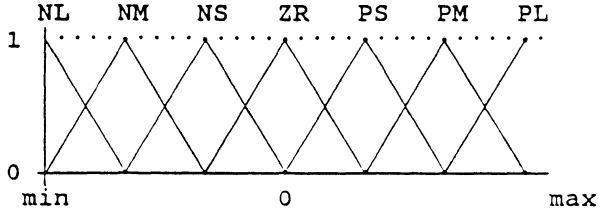

Figure 3

In general, it is easier to know which factors (ie. Xa, .., Xz) are related to the output than to determine how much these factors are related to the output. Hence for each of the rules in the rule base, B can take one of the seven possible forms as its membership functions.

GA can help to determine which one of the seven possible forms is the most appropriate. As suggested by Tsuchiya[7], three bits are used to decode the membership function (a gene of the form 000 indicates that the rule is inapplicable, denoted by O):

PL	PM	PS	ZR	NS	NM	NL	O
111	110	101	100	011	010	001	000

For a rule base of n rules, the length of the chromosomes must be 3n. This experiment started with a sales record of a certain item for 100 days. Each record contained the following fields:

Rec	sales	no. of customers	weather	temperature	day
5	6	12	rainy	18°C	Tue

Labels in the record were summarized as follows:

	antecedent variables	label
1	number of customers	< 10
2		10-14
3		15-19
4		20-24
5		25-29
6		> 29
7	weather	shiny
8		cloudy
9		rainy
10	Temperature	< 9
11		10-14
12		15-19
13		20-24
14		25-29
15		>29
16	day	Mon
17		Tue
18		Wed
19		Thur
20		Fri
21		Sat
22		Sun

22 rules were chosen. Then a collection of 100 chromosomes was generated. The bit vector of each chromosome represents a complete rule base. These chromosomes were evaluated by testing with the recorded data. The better-fit chromosomes were chosen to cross-over and mutate to produce the next generation. After a predetermined number of generations, the final set of rules was determined and could be taken as the contents of the rule base. Once the rule base was established, it could be used to forecast further sales.

COMMENTS

The simplicity of GA is its best advantage over other

heuristics. The algorithm can be implemented very easily and be applied to problems of various natures. Since the next generation of chromosomes is composed of the best-fit chromosomes and the descendants of the best-fit chromosomes, the best genes are kept in the population (by the best-fit chromosomes) and at the same time, better genes may be produced, hopefully, through crossovers or mutations (in the newly generated chromosomes).

Although the near-optimal point can usually be reached in a very short time, GA is easily trapped by local optimal points, suffering from the same defects as other heuristic searches. The likelihood that it falls into a local optimal point depends on the topology of the search space (Ackley[2]). But unfortunately, there is no way to discover the topology of the search space before the search for the optimal point begins.

When the performance of GA was not satisfactory, I normally incorporated a certain degree of stimulated annealing (SA) into the GA process. SA allows totally new chromosomes to be generated in the process of evolution (Xu et al.[5]). Hopefully, a trajectory towards a local optimal point may be diverted towards a more global optimal point by newly generated chromosomes.

The power of GA is further obscured by using a uni-processor machine. The longest time spent on each generation is in the evaluation of all the chromosomes. It is done in a serial manner on a uni-processor computer. Should it be implemented on a parallel machine, the time spent in each generation would be less affected by the increase in size of the population

of chromosomes; and the optimal point could be reached in a much faster time.

However, among all heuristics used in the discipline of operations research, GA is one of the simplest. It elegantly combines the trial-and-error method and the principle of biological evolution. Since crossovers and mutations are the only processes by which new points in the search space are explored, the rate of convergence depends on how many times these two processes are performed in each generation. However, the time spent in performing these transformations is insignificant if it is compared with the time taken to calculate the fitness of the chromosomes.

CONCLUSION

Several experiments using GA are reported in this paper. Although GA has been applied to simple problems like experiments 1 and 2 for some time, the possibility of using GA to determine the membership function of a fuzzy inference system is only a recent finding. In experiment 3, all membership functions were assumed to have a triangular shape. In more sophisticated systems in which membership functions of different forms are considered (most are usually taken as piecewise linear functions), GA might also be applicable for the determination of these functions. But then more bits will be needed to encode the function and the population of chromosomes will become more complicated. The possibility of applying GA to such fuzzy inference systems remains unexplored.

References

1. Holland, J.H. Adaptation in Natural and Artificial Systems, Ann Arbor: University of Michigan Press, 1975.
2. Ackley, D.H. A Connectionist Machine for Genetic Hillclimbing, Kluwer Academic Publishers, 1987.
3. Koza, J.R. & Rice, J.P. Genetic Generation of Both the Weights and Architecture for a Neural Network, IEEE, II-397, 1991.
4. Hesser, J. & Manner, R. An Alternative Genetic Algorithm, Schwefel, H.-P. & Manner, R. (Ed.) Parallel Problem Solving from Nature, Springer-Verlag, 1990.
5. Xu, L & Oja, E. Improved Simulated Annealing, Boltzmann Machine, and Attributed Graph Matching, Almedia, L.B. & Wellekens, C.J. (Ed.) Lecture Notes in Computer Science: Neural Networks, Spring-Verlag, 1987.
6. Liepins, G.E. & Hilliard, M.R. Genetic Algorithms Applications to Set Covering and Travelling Salesman Problems, Brown, D.E. & White, C.C. (Ed.), Operations Research and Artifical Intelligence: The Integration of Problem-Solving Strategies, Kluwer Academic Publishers, 1990.
7. Tsuchiya, T. et al. A Learning Fuzzy Rule Parameters using Genetic Algorithm, Proceedings of the 8th Fuzzy System Symposium, Hiroshima, 1992.