SPE 26367

Design Applications of Genetic Algorithms

E.R. Jefferys, Conoco Inc.

This paper was prepared for presentation at the 68th Annual Technical Conference and Exhibition of the Society of Petroleum Engineers held in Houston, Texas, 3-6 October 1993.

Abstract

The paper outlines the basis of Genetic Algorithm (GA) optimisation and discusses the areas where these powerful computer based search procedures can best be used. Applications include many kinds of engineering design, financial optimisation, scheduling and finding rules to describe data.

An interface between a GA package and the design engineer's problem specification has been developed. The system is simple yet powerful, permitting the engineer to use sophisticated software on a PC without becoming an optimisation expert. Real applications to TLP column design optimisation and scheduling are presented to illustrate the power and flexibility of the method and the simplicity of the interface.

Future additional application areas are suggested.

1 Introduction

Genetic Algorithms (GA) are a computer based search procedure which can be used for a wide range of optimisation and search problems, including those which are very difficult to handle by more conventional techniques. GAs are particularly suited to problems with non-linearities, discontinuities or integer parameters, all of which can cause traditional optimisation schemes to fail.

Applications include many kinds of engineering design and financial optimisation, scheduling and finding rules to describe data. Suitable problems are those with a solution which can be described by a set of numbers and evaluated by a computer programme to give a single measure of 'goodness'. The GA proposes a population of solutions, choosing parameters from user specified ranges, evaluates the solutions and combines the best ones to generate more and usually better candidates. Most of the computational cost usually lies in the evaluation of the worth of solutions. Typically 2000 evaluations are required for a small problem with 5 independent variables and more for larger problems.

2 Genetic Algorithms

2.1 What is a genetic algorithm?

It is a fast computer based SEARCH technique, based on the Darwinian theory of Evolution (ref.1). A population of solutions evolves towards a user defined optimum. GAs find GOOD (but not necessarily optimal) solutions to any problem described by a moderate number of parameters. A major advantage is that they handle DISCRETE parameter values such as number of wells, number of casing types etc. Typical applications in the oil business include:

Optimise TLP structures (cost + weight)
Optimise road/drill pad layout
Optimise casing program
Schedule production
Find rules from data

GAs perform a fast and efficient (but not necessarily perfect) search among a potentially huge number of solutions. A simple pattern matcher demonstration

typically looks at 2000 possibilities before it finds the 1 correct solution in 4.29 billion. This is rather better than a random search!

GA optimisation is faster & better than manual design iteration in spite of GAs total ignorance of the problem; GE use GAs in Expert System based integrated design and find that the higher computational cost is more than paid for by manhour savings (ref.2).

2.2 How do GAs work?

A population of solutions evolves under intense selective pressure. Good solutions combine (breed) while poor solutions tend to be killed off. Good characteristics propagate from one generation to the next while poor ones are weeded out.

In nature 'good' solutions display reproductive success; the 'good' qualities are defined by environment so, for instance a Cheetah's speed is useless in swamps but crucial on the plains, thick fur vital at the poles but pointless in the hot jungle and large weight is a handicap out of water but an asset if it can be supported hydrostatically.

Natural evolution is slow as there is a long time from birth to breeding and animals are very complex systems; the number of bits of information in the genetic code of a simple bug is of the order of the number of characters in the encyclopedia Britannica, while we are typically going to code solutions with 50 bits of information. In nature, there is often diffuse selective pressure as the weakest do not always get weeded out (and a good thing too for some of us!).

Computer evolution is fast as problem evaluation is typically quick and a generation may take minutes, not years. A small number of variables suffices to code the solution to a problem and there is intense selective pressure; we can weed out the poor solutions ruthlessly.

2.3 Detailed Solution

The user writes a program to evaluate the value of one solution.

Parameters to be varied are specified as:
(Type Name Minimum value, maximum value, number of bits.)

Thus (REAL Thickness 1.5 2.5 5) would specify a (nominally continuous) variable named 'Thickness' which takes any of $2^5 = 32$ values between 1.5 and 2.5 inclusive, or as an integer range which can also be used to access a look up table of irregular values.

Thus (INTEGER rings 3 6) specifies an integer variable called 'Rings' which can take the values 3,4,5 or 6.

There are 2^n values for each variable which is coded within the GA as a string of 0's and 1's and hence a whole solution is expressed as a string of 0s & 1s and any string of the correct length represents a valid solution.

If we had only three variables, the problem specification files would be:

Type	Name	Min	Max	Nbits	Example	Binary
INTEGER	rings	3	6	2	5	10
REAL	thick	1.2	2.7	4	1.9	0111
INTEGER	Nstf	5	20	4	10	0110

The GA generates (say) 100 RANDOM binary strings of correct length (10), each of which represents a viable (if, perhaps undesirable) solution eg:

1001110110 maps to (5, 1.9, 10)
0010110111 maps to (3, 2.2, 12)
0100011111 maps to (4, 1.2, 20)
.
.

At the start of the optimisation, the GA translates binary strings to input parameters for the users computer program and calls the program to evaluate the worth of each possible solution. Once this is complete, the GA reads the file of merits and ranks them from worst to best. It then picks pair of solutions randomly, but with a bias towards the best.

Each 'gene' (string of 0's and 1's) is cut at a random place and the parts are swapped to generate new solutions

Thus:
Parents: 1001111101 & 0010110011
Cut between 3 and 4 to generate
Children: 100 ¦ 0110011 & 001 ¦ 1111101

An equal number of solutions from the old set are eliminated, again chosen randomly, but with a bias towards the worst. The 'children' replace the eliminated solutions and are then evaluated in their turn. Once again, the merits are sorted and the generations repeat until a specified number of individuals have been evaluated or the solution has reached a specified performance.

This technique combines parts of good solutions so one of the two will likely have the best features of each! This is similar to the human design approach. The method finds good solutions and the population converges to 1 or more optima. The process is quick if evaluation is quick.

2.4 What is special about GA optimisation?

Ordinary optimisation is typically based on 'hill climbing', where the changes in parameters are determined by the direction of steepest ascent. This approach falls down when there are canyons between allowable parameters (eg integer values only). In some cases, cliffs in the search space (constraints which cannot be crossed) also cause problems. The GA uses ALL discrete values and therefore discretises continuous values. Thus it handles integers well.

A family or families of solutions develop so that there may be radically different solutions available at the end of a run, permitting the engineer to choose on the basis of perhaps hard to codify criteria such as ease of fabrication, maintainability or familiarity.

The GA optimisation is not specialised to a particular problem; indeed it is totally ignorant of the characteristics of the problem so it cannot be tuned. Quick development of a solution package is balanced by lower computational efficiency - but computers keep increasing in performance.

GAs typically don't stick on local optima as the population samples a huge volume of the potential search space - although the population can converge to a good but not optimal solution if the selection is too brutal.

GA solutions are less likely to be misleading as the problem need not be bent to fit the solution method and they are robust and do not have problems with strange constraints or discontinuities.

They provide good solutions to 'hard' problems, defined as those with no GUARANTEED solution better than simple enumeration of ALL possible solutions.

2.5 When is GA optimisation not useful?

Hard problems with lots of variables cause all methods to fail! Easy problems with known solutions such as Linear Programming and Continuous Quadratic problems should be solved using conventional methods with fast efficient existing code. Problems with very expensive evaluation & well behaved solutions such as (perhaps) FE analysis may be better solved by hill climbing techniques.

Some hard to code problems are difficult to attack as the cross over breeding must give a valid solution. Fortunately most engineering problems are not of this type.

2.6 How do you use GAs for your problem?

Decide what you want to optimise! Cost? Weight? Schedule? Perhaps some combination of these with constraints. In the example which follows, we minimise the weight of a TLP column subject to a strength constraint - otherwise the minimum weight would converge to zero! You must decide what varies and what is fixed by external requirements; the more flexibility, the more likely is a truly excellent solution.

Write a program to evaluate any solution efficiently; for maximum speed, a Fortran or C evaluator may be linked directly with the GA (the package Conoco used was written in C).

All data are transferred via accessible ascii files, typically residing on RAM disc for speed, thereby eliminating many of the C-Fortran interface conflicts which can arise with argument passing.

A less computationally efficient but more independent interface was implemented where control is transferred to the evaluator at system level by a DOS system call, which eliminates all mixed language linking problems. Using this approach, it is easy to use any compiled or interpreted language, or, indeed a spreadsheet but this last option would not be particularly efficient.

The analysis program reads GA proposed parameters from an ascii text file with a specified structure and any auxiliary parameters from files to be defined by the user. The program loops over the GA specified values, writing a value to the output file for each case. In addition the user must specify the parameters to be optimised to the GA using the simple format used in the explanation above.

3 Example Problems

The two example problems were coded in Fortran and tested to obtain some hands on experience.

3.1 Stiffened Shell Optimisation

The objective is to choose design parameters to minimise the weight of a column of an offshore structure, subject to the constraint that it should bear a design radial pressure and axial load. This structure is a cylinder of variable skin thickness, stiffened by a number of rings of fixed design but with variable (equal) spacing. Additional stiffening is provided by a variable number of equally spaced stringers, which run the length of the structure and are welded to the outer skin. These stiffeners are of T beam form with variable thickness and width for both the web and the flange. Other parameters such as column radius, yield stress of the material etc are assumed fixed in the optimisation.

The evaluator program calculates the weight of the structure from the fixed variables and the parameters proposed by the GA and evaluates the capacity of the structure to carry the required loads. If the capacity is inadequate, the weight is penalised in proportion to the deficit in capacity. The adjusted weight is the output of the evaluator program. The capacity evaluator resulted from the Model Code for Reliability Based Design JIP.

The optimiser typically ran for 12 minutes, executing 5 experiments with 2000 evaluations in each. The best structure weighed about 92% of the Jolliet structure when required to resist the Jolliet axial collapse load with an associated 3 bar pressure. This is a fairly encouraging result since problem setup was less than 8 manhours and the problem could have been re-run in 12 minutes if any of the 'fixed' parameters (eg column diameter) had been varied, as occasionally happens in design.

3.2 Scheduling problem

While there are many effective specialised scheduling

algorithms, this application is presented to illustrate the wide range of GA applications. In addition, there is the option so solve coupled configuration (design) and scheduling problems.

Here the objective is to maximise the net present value (NPV) of a production system. Three products of equal value can be extruded by machines which are available in three capacities, with specified capital costs. A set amount of production is required for each product and a schedule must be set on a monthly basis. There are money and capacity costs in switching from one product to another on any machine.

The problem is coded for the GA using a 2 bit variable to specify machine type, with type 0 indicating no machine and no production. Up to three machines are coded into the problem since it is clear that the required capacity is achievable with this number. Product A is arbitrarily (but without loss of generality) assumed to begin production on the first day of the month while the start dates for products B and C are 5 bit GA variables. If a product begins and ends production on the same day, no changeover costs are charged and no production takes place. Any production above annual requirements is not valued - ie it is assumed that machines can be switched off without cost. Production is not forced to the annual requirement, but product value is large compared to machine cost so maximising NPV tends to generate solutions near to the requirement.

This problem had been investigated in a study of the 'simulated annealing' method of optimisation and typically took an hour to run on a VAX. The GA achieved very similar configurations and schedules after about 10 minutes on the PC.

One unique feature of the GA is that the best solution found may have an even better one very nearby and this occurred here. Production would run for one day on a machine, incurring substantial changeover costs. Ordinary optimisation methods are good at finding local optima but poor at global ones; the GA is the reverse so it is worth perturbing a GA solution locally to see if improvement is possible.

4 Application Areas

Now the programme is validated, other applications are being investigated.

- Floating structures;
  Shape optimisation to minimise hydrodynamic forces.
- Composite downhole tubing design;
  Choice of materials and structure to optimise performance.
- Customer supply depot allocation;
  Choice of supply depots to minimise distribution costs and satisfy capacity constraints.
- Crude blending optimisation;
  Maximise the value added by blending complementary crudes to fit refinery specifications.
- Portfolio optimisation;
  Choose projects to maximise return under a total capital constraint.

5 Conclusions

Robust optimisation methods can automate design and analysis problems previously amenable only to cut and try 'hand' analysis. Significant improvements in the quality of the solution are common and engineering time can be reduced significantly, particularly for frequently repeated analyses. The discipline of defining criteria and constraints for optimisation can bring focus to the design effort.

GAs are a powerful tool which can have a great effect on many areas of the petroleum business. However, they require some support to introduce them to potential users.

The discipline of viewing a problem as an optimisation may take some time to become familiar. It may be difficult to evaluate an option to give a single measure of worth, yet if this is not done, explicitly or implicitly, there is ultimately no way to choose between competing options. GAs are helpful in that constraints and criteria do not have to be bent to fit the problem solving methodology and can be specified naturally by users.

Optimisation can only help those who know what they want.

6 References

1. Genetic Algorithms in Search, Optimization and Machine Learning, D.E. Goldberg, Addison-Wesley, 1989.
2. Mechanical Engineering, February 1992, pp ?? - ??, ASME