GENETIC ALGORITHMS WITH AN APPLICATION TO
NONLINEAR TRANSPORTATION PROBLEMS

MATTHEW HOBBS

ISOR, VICTORIA UNIVERSITY
WELLINGTON

Abstract

In this paper I will describe the theory and implementation of Genetic Al­
grithms. This is followed by a description of an attempt to solve the nonlinear
transportation problem using a genetic algorithm approach, (Michalewicz, Vi­
gnaux, & Hobbs [1990]). This is a continuation of earlier research in which a
system was built for the standard linear transportation problem (Vignaux &
Michalewicz [1990]). Results of this technique are promising.

1 Genetic Programming

Much of problem solving in Operations Research involves optimization using one of
a variety of techniques. The most extensive algorithm in practice is that of Linear
Programming. Like LP, most algorithms apply only to relatively small problem
domains. Some algorithms guarantee an optimal solution while others are heuristics
and aim to find an approximate or good solution. This still leaves a wide range of
problems for which there are no useful algorithms.

In general, algorithms take advantage of particular structural properties of the
problem in order to solve it. This approach fails for most complex problems. In
may cases the information may be little more than the ‘payoff’ of any particular
solution (e.g. cost) and the expression of the solution (e.g. a production schedule).
In this case the procedure for payoff optimization is unclear. If we have a given
solution, with a determined payoff, how do we know if it is optimal? How do we
determine what part of the solution expression should be changed if it is not? With
only the one solution we can usually answer none of these questions. The key is to
incorporate comparisons with other solutions into the algorithm

Searching of the type described above are usually probabilistic search tech­

The obvious
disadvantage is the time costs involved (when the feasible set is of any size). A more sensible approach is to somehow bias the random search to home in on the best solutions, quickly. An area where random search can be improved is in using the history of the search to select new (trial) solutions. The question becomes one of asking how do we most efficiently use the search history to produce the trial solutions?

Simulated Annealing is an algorithm whereby the history at any stage in the algorithm is the current solution. A trial solution is chosen by modifying the current solution in some way (mutation) and the new current solution is chosen probabilistically between the trial and current solutions, according to their relative payoff.

In Simulated Annealing the search history used at any step is only the current solution, previous solutions are discarded. Genetic Algorithms use the idea that previous solutions should not be discarded immediately and that useful information can be obtained by maintaining groups of solutions as the search history. The primary problem is how to combine the pool of solutions to create new (better) solutions.

The concept of maintaining a pool of solutions in order to generate new trial solutions is analogous to that of natural evolution with genetic code being the expression of an individual solution. New solutions are created by combining code from existing solutions (crossover). The optimizational tendency occurs when current solutions are discarded from the pool (e.g. die) according to lack of payoff. Algorithms that include ideas such as these are called Genetic Algorithms (Holland [1975], Davis [1987], Goldberg [1989]).

A typical Genetic Algorithm starts with a population of randomly generated solutions (the initial population) and repeatedly applies genetic operators modeled on natural genetic processes (e.g. crossover, mutation) to the population. Consider this extract from Davis [1987]:

"... the metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The ‘knowledge’ that each species has gained is embodied in the makeup of the chromosomes of its members. The operations that alter this chromosomal makeup are applied when parents reproduce; among them are random mutation, inversion of chromosomal material, and crossover - exchange of chromosomal material between two parents’ chromosomes."

Theory

Any run of a genetic algorithm is essentially a simulation of the evolution of a set of solutions exposed to constraints and performance (the environment). Those individuals (solutions) better suited to the environment, in general, will have a greater chance of survival and therefore have a greater chance of producing offspring. Although this may seem initially as a hill-climbing process, the algorithm (at the same time) maintains population diversity within the search space. The ability to maintain diversity enables the algorithm to escape local optima and ‘leap tall buildings’ without too much ado. Conversely, near a global optimum, the algorithm would have difficulty ‘homing in’ to the precise solutions, due to the blindness of the random mutations.
The expression of a solution (chromosome) for various problems will vary considerably but can be considered as a collection of elements (genes). Ideally we would like to be able to treat each gene separately, for optimization purposes. This is not usually feasible due to the often highly nonlinear or constrained nature of the problem. The only information we have may be the payoff of the chromosome as a whole. A feature of the genetic algorithm is in that although it appears to involve only competition between chromosomes, due to the nature of the genetic operators (e.g. crossover) - that manipulate genes directly - in fact competition occurs at the gene level. This makes sense when we consider that new solutions are created with only a subset of genes differing from their parent(s).

A useful term to use is that of a schema - see Holland [1975] or Goldberg [1989]. A schema (in this context) can be considered as a 'similarity template' for chromosomes defining a set of fixed genes. For example fixing gene number 3 at 'yes' and gene number 7 at 'no' (and leaving all other genes free) defines a schema for the chromosomes. To be more accurate than the previous paragraph, competition is on the schema level. Although a genetic algorithm processes only \( n \) structures per generation, it also processes of the order of \( n^3 \) schemata. Holland has named this important result implicit parallelism.

It is generally accepted that for any problem to be attempted by a genetic algorithm there must be five components, as follows.

1. A genetic representation of solutions to the problem (chromosome),
2. A way to create random solutions (for the initialization),
3. An evaluation function (the 'environment'), rating solutions by 'fitness'.
4. Genetic operators to alter child composition during reproduction, and
5. Values for the parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.)

For a variety of reasons most applications of the genetic algorithm have used binary (bit string) representations for the solutions (chromosomes). But the 'implicit parallelism' result is not restricted to bit strings (see Antonisse [1989]). The application in this paper uses a matrix of real numbers. Although it is true that any structure that can be put on computer can be represented as a binary vector (string), the difference is that the schema for the richer structure are much more relevant, and there are much less of them.

Implementation

A genetic algorithm will typically be of the following form.

1. create an initial set of random solutions (the initial population);
2. evaluate the solutions (see how good they are);
3. put the better solutions into a set:
There are two main genetic operators, \textit{mutation} and \textit{crossover}. The mutation operator arbitrarily alters one or more components of a selected structure—this increases the variability of the population (introducing new - or lost - schema). In general, each gene of each chromosome in the population undergoes a random change with a probability equal to the mutation rate. The crossover operator combines the features of two parent structures to form two similar offspring. It operates by swapping corresponding segments of a chromosome representing the parent solutions (with the objective of combining good schemata to produce better schemata - with more fixed genes).

When problems are found to be too specialized or complex for the standard algorithms it is necessary to take a heuristic approach to the problem. Genetic algorithms can be a powerful tool in such cases due to the fact that it can combine the information from various solutions (and at the same time, maintain the diversity required to try to 'span' the search space). Currently there is no programming language specific to this problem domain so systems have had to be set up in other languages as available. It should be noted that when the parameters to the problem are changed (e.g. a different problem of the same type) the programs set up will require very little change. For example if the objective function (goal) of the problem changes it is only necessary to change the evaluation procedure and no other. Such systems are more flexible to changes than most. For example a problem being solved using Linear Programming is fine until nonlinearity is introduced, in which case another algorithm is required. Genetic algorithms can compete very effectively against standard nonlinear algorithms on nonlinear problems. They may be best used in combination with nonlinear algorithms for such problems. For example using a nonlinear algorithm to home-in on the local optima.

There are a number of applications for which genetic algorithms have been applied including: physical design of circuit boards; travelling salesman (node-covering) problem; combinatorial problems in general; game strategy; nonlinear problems: optical design; keyboard configuration; machine learning; simulation of evolution; and gas pipeline optimization. In fact any problem that meets the five properties listed previously can be attempted by a genetic algorithm.

2 A Genetic Algorithm for the Nonlinear Transportation Problem

In testing the use of the genetic algorithm on the linear transportation problem (see Vignaux & Michalewicz [1989],[1990]) it is possible to compare its solution with the known optimum found using the standard algorithm and therefore to determine how efficient or otherwise the genetic algorithm is in absolute terms. Once we move to nonlinear objective functions, the optimum may not be known. Testing is reduced to comparing the results with those of other nonlinear solution methods that may themselves have converged to a local optimum.
The genetic algorithm will be compared with the GAMS nonlinear system (with MINOS optimizer) as a typical example of an industry-standard efficient method of solution. This system, being essentially a gradient-driven method, found some of the problems set up difficult or impossible to solve. In these cases modifications to the objective functions were made so that the method could at least find an approximate solution. The genetic system itself was written by Z. Michalewicz in the 'C' programming language.

The parameters required include (as well as the problem description): number of iterations, population size, mutation and crossover rates, and random number starting seed.

The behaviour of nonlinear optimization algorithms depend markedly on the form of the objective function. It is clear that different solution techniques may behave differently. In the experiments the overall objective function is the sum of the arc objective functions, thus there were no cross terms. Six different arc objective functions were used including: step function (with 5 equal steps); discretely changing slope (3 zones of a particular gradient); square; square-root; a function with a peak; and a linearly increasing sine function. They were used in conjunction with five problem structures in which the supply and demand vectors and the corresponding parameter matrix were randomly generated. The solutions to these problems are unknown.

The objective function for the transportation problem was thus of the form

\[ \sum f(x_{ij}) \]

where \( f(x) \) is one of the six arc objective functions.

Experiments and results

The population size was fixed at 40. The mutation rate was 20% with the proportion of mutation-1 being 50%, and the crossover rate was 5%. Problems were run for 10,000 generations.

The systems were tested on the six functions for five randomly generated cost matrices. See the second graph page for a graphical display of the results.

The genetic system was run on SPARC station/SUN terminals while GAMS was run on an Olivetti 386 with math co-processor. Although time comparisons between the two machines are difficult to make it should be noted that in general GAMS finished each run well before the genetic system. An exception is case A (in which GAMS evaluates numerous arc-tangent functions) when the genetic algorithm took no more than 15 minutes to complete while GAMS averaged at about twice that. For cases A,B, and D, where the GAMS parameter meant that multiple runs had to be performed to find the best GAMS solution, the genetic system overall was much faster.

Conclusions

The transportation problem was chosen as it provided a relatively simple convex feasible set. This means that it is easier to ensure feasibility in the solutions. The
procedure was then to look at the effects that the objective function alone has on the solving of the problem.

For the class of practical problems the genetic system is, on average, better than GAMS by 10% in case A and by 11.6% in case B. For the reasonable functions the results were different. In case C (the square function), the genetic system performed worse by 7.6% while in case D (the square-root function), the genetic system was worse by only 0.6%, on average. For the other group of functions the genetic system is dominant. The genetic system resulted in improvements of 32.9% and 55.1% over GAMS, averaging over the five problems.

This demonstrates the superiority of the genetic method over other systems which are very often limited to certain classes of problem functions.

GAMS did well on the smooth/monotonic (reasonable) functions. It is these cases where the gradient measurement techniques are most apt. In case C GAMS bettered the genetic system with much less cost in solving time.

For the practical problems, the gradient techniques have difficulty "seeing around the corner" to new zones of better costs. The genetic algorithm, taking a more structural approach, is able to jump between zones readily, resulting in much better solutions.

For the other problems, although they are both smooth, have significant structural features. Like the practical problems, but even more so, the genetic system did much better the GAMS.

REFERENCES


