Constrained Gas Network Pipe Sizing with Genetic Algorithms

Ian D. Boyd
boyd@bgers.co.uk
British Gas ERS
Research and Technology Division
Newcastle-upon-Tyne
NE99 1LH, England

Patrick D. Surry, Nicholas J. Radcliffe
{pds,njr}@epcc.ed.ac.uk
Edinburgh Parallel Computing Centre
King's Buildings
University of Edinburgh
EH9 3JZ, Scotland

Abstract

The application of a genetic algorithm to an industrial pipe-sizing problem is presented. The optimum sizes for the pipes in a given network with specified supply and demand requirements must be determined, subject to two additional constraints. The problem was used as a test case for the evolutionary computing language RPL2. The genetic approach is shown to produce better results than the existing industrial heuristic at the expense of longer run times.

1 Problem Specification

It is a frequent criticism of genetic algorithms and evolutionary computation more generally that published problems are usually contrived and unconstrained. (There are of course notable exceptions, for instance the work of Goldberg (1989) on gas network compressor optimisation.) This paper demonstrates how a relatively straightforward implementation of a genetic algorithm can find significantly better solutions than the standard heuristics actually used for a real, constrained pipe-sizing problem to offer actual cost reductions of about 4%.

The design of a gas network—e.g. to supply a new housing development—involves defining the layout of the network and, having done this, choosing the types of pipe to be laid. The layout is generally determined by such considerations as the routes of roads but the selection of the pipe types is tackled as a constrained optimisation problem. The important constraints on any design are that:

- the pipes selected should allow the customer demands to be met at or above a 'minimum design pressure'
- each pipe (other than those incident to a source) should have at least one upstream pipe of the same or greater diameter.

Pipes are produced in a range of discrete diameters and in a number of materials, and for a given material the cost per unit length of pipe is an increasing function of diameter.

The pressure drop along a pipe, for a fixed flow, is a decreasing function of diameter so larger diameters will generally give a more secure network. The problem in designing a network is therefore to select the diameters of pipes in such a way that they are large enough to provide security of supply but not larger than they need to be. The latter would amount to an over-design of the network, in that a cheaper design would have been adequate.

Real problems often involve some pipes whose diameter is fixed, typically because they have already been laid. Thus the pipe sizing of the network does not necessarily involve selecting the diameters of *all* of the pipes in the network.

In the particular problem being considered, the network contained 25 pipes, each of which could be selected from six possible sizes, giving rise to a search space of size $6^{25} \simeq 3 \times 10^{19}$. The pipes connect 25 nodes, 23 of which are (varying) demand nodes and two of which are pressure-defined source nodes (flow-defined source nodes may also be specified). The network is a real one, which was actually built (with pipe-sizes determined using the heuristic method discussed in section 4.1).

2 Genetic Representation and Operators

The representation used to represent the pipe sizes in the network is a variable cardinality integer string. A genome is a sequence of N integers $a_1 a_2 \cdots a_N$ with $a_i \in \{0, 1, \ldots, C_i - 1\}$, where C_i is the cardinality (number of alleles) of the ith gene. (The particular problem instance tackled happened to have $C_i = 6$ for each pipe but this is not generally the case.)

This representation was built as a generic library for *The Reproductive Plan Language 2*, RPL2 (Surry & Radcliffe, 1994). Several generic genetic operators were used to manipulate the representation. The evaluation function used is described in section 2.2.

2.1 Genetic Operators

Create Genome A random genome is created by uniformly choosing values for each gene from the allowable set of alleles.

Random Mutation The random mutation operator simply replaces each allele by a random allowable value with probability equal to the mutation rate r. It is possible that an allele could be replaced by the same value. In our work we used random mutation with rate 0.025, combined with creep mutation.

Creep Mutation operator replaces each allele a_i with $a_i + 1$ or $a_i - 1$ with probability $(C_i - 1) \cdot r$ where r is the mutation rate. Thus genes with high cardinalities are more likely to be modified than those with low numbers of allowable alleles. This biases the operator so that it takes the same number of trials (in expectation value) to mutate from the lowest allowable value to the highest (0 to $C_i - 1$). A cyclic flag indicates whether creeping directly from $C_i - 1$ to 0 (or vice versa) is allowed. For the pipe-sizing problem, we used non-cyclic creep mutation with rate 0.05.

This operator is more appropriate (and effective) in the context of the pipe-sizing problem since the alleles in this case correspond to an ordered list of pipe sizes, so that consecutive integers represent "close" pipe sizes.

N-Point Crossover A standard N-point crossover operator was developed which produces a single child from two parents. The operator selects N points in the string of integers, and forms a child by alternately copying sections from one parent and then the other, switching at each cross point. The operator is further parameterised to sometimes simply clone one of the parents rather than crossing them. Initial experiments made use of N point crossover with $2 \leq N \leq 5$ but it was found to be less effective than uniform crossover.

Uniform Crossover The parameterised uniform crossover (Spears & DeJong, 1991; Syswerda, 1989) was also used. The operator creates a single child from two parents, by choosing each gene in the child from one of the parents, selected randomly according to the bias parameter. The operator may also simply clone one of the parents, according to a second parameter. We used a bias of 0.6, and never allowed cloning.

Although this operator is cited as weaker than N-point crossover in preserving short schemata, this is relevant only if there is greater than average correlation between adjacent genes in the genome (strong linkage). In the case of the pipe-sizing problem, it is difficult in general to to define an ordering in which this is necessarily the case, making the uniform crossover operator appropriate for the problem. It is possible that a labelling derived by minimising the bandwidth of the network's connectivity matrix would increase the effectiveness of N-point crossover for the pipe-sizing problem, but this has not yet been explored.

A further distinction between N-point and uniform crossover is the distribution of genetic material in the child. Uniform crossover gives a binomial distribution of material from the two parents, while N-point crossover gives a uniform distribution (Eshelman *et al.,* 1989).

2.2 Evaluation Function

The evaluation function used determines the cost of a genome by summing the cost of the pipes making up the network.

The two constraints must be considered, however. Both the upstream pipe constraint and the minimum pressure constraint are implicit. Their satisfaction can only be determined by solving the non-linear gas flow equations in the network. (This defines the upstream direction and the pressure at each node in the network.) This means that a penalty function is really the only viable approach for handling these constraints, as it would be extremely difficult or impossible to construct genetic operators that respected them. Ideas from Richardson *et al.* (1989), Michalewicz & Janikow (1991) and Michalewicz (1993) were used to increase the penalty gradually as a function of generation number.

The form of the cost function used was

$$cost = \sum_{j=1}^{N_{pip}} l_j \cdot c(D_j) + \alpha n_{gen}^{k_1} \cdot (p_{des} - p_{min})^{k_2} + \beta n_{gen}^{k_3} \cdot \sum_j (D_j - D_k)^{k_4} \qquad (1)$$

where the first term is the cost of the pipes as a function of diameter, the second term is the minimum pressure constraint, and the final term is the upstream pipe constraint with summation over pipes j where there is no upstream pipe which is of greater or equal diameter and D_k is the diameter of the largest upstream pipe from pipe j.

Values were selected for constants α and β which normalised nominal values of the penalties to the same scale as the basic cost of the network. The various exponents were selected in order to make the penalties grow at roughly the same rate as networks became "worse" at satisfying the constraints (values of $k_2 = 0.5$ and $k_4 = 1.0$ were used). The annealing parameters k_1 and k_3 were subject to some experimentation, but 0.2 was found to be an effective value for both.

3 The Reproductive Plan

A fairly conventional reproductive plan was developed to tackle the pipe-sizing problem using RPL2 (Surry, 1993). The actual plan which yields the best solution is presented in section 5.

Elitism was used to preserve the best member of the population. Because the fitness of a genome depends on penalty functions that vary with the current generation number, the entire population must be reevaluated at each generation, prior to fitness-based reproduction.

Various forms of structured population structures were investigated during the problem, using the facilities provided by RPL2. A panmictic (unstructured) population was used in most initial experiments, in order to tune values of parameters, particularly for the fitness function. A fine-grained structure did not yield significant benefits, perhaps due to the relative simplicity of the problem. An island structure was also used, with the main benefit being the ability to run on multiple processors and try longer runs with larger populations. However, this also did not significantly improve performance.

4 Results

4.1 Heuristic

The current heuristic technique used by British Gas was applied to the problem, in order to compare its performance to the genetic approach. In fact the network was actually installed using the results obtained from the heuristic.

The heuristic determines a good configuration of pipe sizes by first assuming a constant pressure drop over the whole network and guessing some initial pipe sizes which will yield a valid network (satisfying the constraints) but not necessarily optimal. The heuristic proceeds by locally optimising this solution, repeatedly trying to reduce single pipe diameters while maintaining a valid network. Eventually this process terminates when no pipe size can be reduced while maintaining network validity.

The algorithm takes on the order of ten seconds on a 486 PC (25MHz) to reach its "optimal" configuration for the problem under study. A schematic (which does not represent differences in pipe lengths and source/demand requirements) of this solution appears in the left part of figure 1.

4.2 Genetic Algorithm

The genetic technique produced consistently good results, although it did not always converge to the same optimal solution. In most cases it found networks which were better, often significantly, than that determined by the heuristic approach. In almost all

cases the algorithm found a valid network by the end of the run (i.e. one in which the penalty terms were zero). Run times for typical populations of 100 networks through 100 generations were of the order of several minutes on a Sun SPARC 2 workstation. The best result was a network which was approximately 4% cheaper than the heuristic solution. A schematic of this network is shown in figure 1.

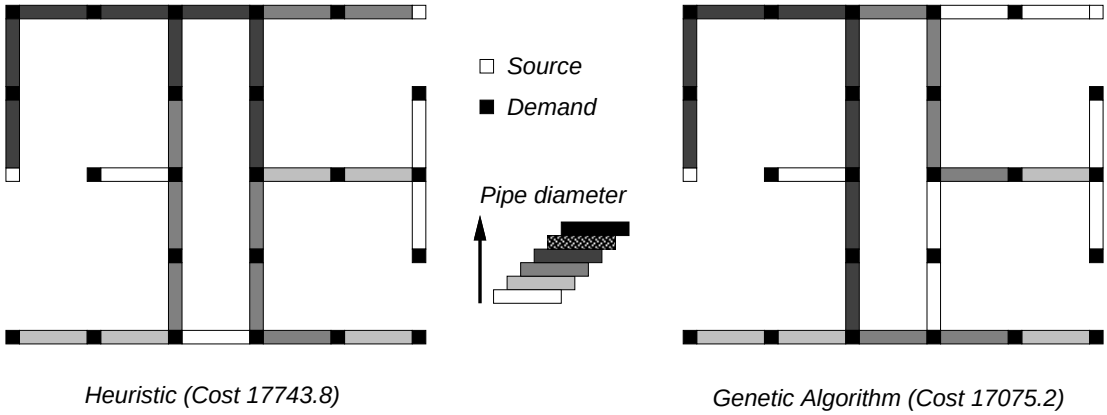

Figure 1: The genetic algorithm finds a solution approximately 4% better than that found by the heuristic technique. The networks are shown only schematically, so that the pipes are actually of different lengths. The demand and supply requirements are also different at each node. Both networks are valid in that they satisfy both the upstream-pipe and minimum pressure constraints. The genetic algorithm result is from a panmictic population of population size 100, running for 100 generations. A typical run time for the heuristic is a few seconds, as compared to a few minutes for the genetic algorithm.

5 The RPL2 Plan Used to Solve the Problem

```
plan(PipeSize)                  % Plan that produces the 17075 solution

use     StdInst, IntVC("data.dat"), Debug;

string  sFile;
bool    bMaxIsBest;
int     iCounter, nGenerations, i, nPopsize;
genome  gNew,gChild,gParentA,gParentB, gBest;
gstack  gsPop, gsCache;
real    rExp1, rExp2, rExp3, rExp4, rMinPressPen, rUpstreamPen, rMutateRate;

BGInit("network.dat");          % Initialisation section
sFile := "stdout";              % Direct output to the terminal
nPopsize := 100;                % Population size
nGenerations := 100;            % Number of generations
bMaxIsBest := FALSE;            % Minimise the cost function
rExp1 := 0.2;                   % exponent for n_gen in min press penalty term
rExp2 := 0.5;                   % exponent for min press penalty value
rExp3 := 0.2;                   % exponent for n_gen in upstream penalty term
rExp4 := 1.0;                   % exponent for upstream penalty value
rMinPressPen := 20.0;           % scale factor
rUpstreamPen := 2.0;            % scale factor
rMutateRate := 0.05;            % basic mutation rate
Randomize(6684271);             % pseudo random initialisation

for iCounter := 1 to nPopsize   % generate initial population
        gChild := RandomGenome();
        BGCostFunction(gChild, 1, rExp1, rExp2, rExp3, rExp4, rMinPressPen, rUpstreamPen);
        Push(gChild,gsPop);
endfor

for i := 1 to nGenerations
```

```
        Empty(gsCache);         % re-evaluate all solutions since penalty changes
        for iCounter := 1 to nPopsize
                gChild := Pop(gsPop);
                BGCostFunction(gChild, i, rExp1, rExp2, rExp3, rExp4, rMinPressPen, rUpstreamPen);
                Push(gChild, gsCache);
        endfor
        Swap(gsPop,gsCache);

        gNew := SelectRawBest(gsPop, bMaxIsBest);       % use elitism
        for iCounter := 1 to nPopsize - 1
                gParentA := SelectRawTournament(gsPop, bMaxIsBest, 2, 0.6, TRUE);
                gParentB := SelectRawTournament(gsPop, bMaxIsBest, 2, 0.6, TRUE);
                gChild := CrossUniform(gParentA, gParentB, 0.6, 1.0);
                MutateRandom(gChild,rMutateRate / 2.0);
                MutateCreep(gChild,rMutateRate,FALSE);
                BGCostFunction(gChild, i, rExp1, rExp2, rExp3, rExp4, rMinPressPen, rUpstreamPen);
                ReplaceRawTournament(gsPop, gChild, bMaxIsBest, 4, 0.6, TRUE, TRUE);
        endfor
        ReplaceRawWorst(gsPop, gNew, bMaxIsBest);

        PrintInt(i);
        gChild := SelectRawBest(gsPop, bMaxIsBest);
        BGCostFunction(gChild, i, rExp1, rExp2, rExp3, rExp4, -rMinPressPen, -rUpstreamPen);
        PrintGenome(gChild,sFile);
        StatsPrint(i,5,gsPop,sFile);
endfor
endplan
```

References

Davis, 1991. Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold (New York), 1991.

Eshelman *et al.*, 1989. Larry J. Eshelman, Richard A. Caruana, and J. David Schaffer. Biases in the crossover landscape. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.

Goldberg, 1989. David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley (Reading, Mass), 1989.

Michalewicz and Janikow, 1991. Zbigniew Michalewicz and Cezary Z. Janikow. Handling constraints in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1991.

Michalewicz, 1993. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag (Berlin), 1993.

Richardson *et al.*, 1989. John T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–195. Morgan Kaufmann (San Mateo), 1989.

Spears and De Jong, 1991. William M. Spears and Kenneth A. De Jong. On the virtues of parameterised uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236. Morgan Kaufmann (San Mateo), 1991.

Surry and Radcliffe, 1994. Patrick D. Surry and Nicholas J. Radcliffe. Rpl2: A language and parallel framework for evolutionary computing. Technical Report EPCC-TR94-10, Edinburgh Parallel Computing Centre, 1994.

Surry, 1993. P. D. Surry. RPL2 user guide. Technical Report EPCC-BG-PAP-RPL2-UG, Edinburgh Parallel Computing Centre, November 1993.

Syswerda, 1989. Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.