Evolving simple fault-tolerant routing rules using genetic programming

S.H. Shami, I.M.A. Kirkwood and M.C. Sinclair*

Dept. of Electronic Systems Engineering, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ.

Tel: 01206-872477; Fax: 01206-872900; Email: mcs@essex.ac.uk

Monday 2$^{\text{nd}}$ June 1997

Abstract

A novel approach to solving network routing and restoration problems using the genetic programming (GP) paradigm is presented, in which a single robust and fault-tolerant program is evolved which determines the near-shortest paths through a network subject to link failures.

Introduction: The aims of this Letter are to demonstrate the principle of applying genetic programming (GP) [1] to find the shortest or near-shortest path route through simple networks subject to link failures, to assess the approach on a number of test networks, and to explore whether multi-population GP could provide improved results. Traditional centralised methods, such as the simple Dijkstra's Algorithm [2] can be used to identify the shortest path through a static network. Should links fail, Dijkstra's Algorithm could be re-applied to the faulty network and new shortest paths obtained. Alternatively, a distributed restoration algorithm, such as Grover's [3], could be applied. However, the single centralised program described in this paper, found by GP and composed of problem-specific functions, is not in competition with Dijkstra's Algorithm or any other traditional centralised routing algorithms, but rather was 'blindly' evolved to find the near-shortest paths in the given network subject to link failures, and thus is fault-tolerant and robust [4].

*Author to whom correspondence should be addressed

In Fig. 1 a notation of the form `p(q)` is used to identify the links in a network where `p` is the link number and `q` is the length of that link. For example in Network 1 of Fig. 1, Node 3 is connected to Node 4 via Link 6 whose length is 2 units. Now consider the application of GP to, say, Network 1. Our problem then is to evolve a single program that can route information from any node back to Node 0, when there can be up to one link failure. The routes found will be the shortest or near-shortest paths. In other words, a single program is to be able to find simultaneously the shortest or near-shortest path trees through eight networks: the network when all links are fault-free, and seven networks when each of the seven links, in turn, is broken.

This is the first known application of GP to either a telecommunications network routing or restoration problem.

Problem Representation: GP usually describes its solution by means of LISP *s*-expressions which can be re-drawn as parse trees containing the problem-specific functions and terminals used. Our novel problem-specific function, used to evolve a solution to the problem, is as follows:

```
(IF-CUR-GO W X Y Z) =
  X, if the Current Node is Node W,
     Node W and Node X are directly
     connected, and Node X has not been
     visited twice before.
  Y, if the Current Node is Node W,
     Node X and Node W are not directly
     connected, or Node X has been
     visited twice before.
  Z, if the Current Node is not Node W.
```

For example, in Network 1, if the information to be routed was at Node 3, the `Current Node`, the function `(IF-CUR-GO 3 1 4 2)` would route the information from Node 3 to Node 1 provided Node 1 has not been visited twice before, else it would be routed to Node 4. This next node then becomes the `Current Node`. Our Function Set [1] is then {`IF-CUR-GO`} and the Terminal Set comprises the nodes {`0,1,2,3,4`}. The driving force behind all evolutionary algorithms is the problem-specific fitness function. In this application, the fitness of a potential solution was obtained by summing the lengths of the paths from Nodes 1, 2, 3 and 4 back to Node 0 in each of the eight possible networks (Network 1, and the seven variants of Network 1 with one link broken); clearly, in this case, the smaller the fitness measure, the better is the solution.

To assess the results, we define a new fitness measure, *Dijkstra fitness* (f_D) which is the difference between the fitness of the solution found by our GP run (f) and the true optimum (found by Dijkstra's Algorithm [2]), which we call s, *i.e.* $f_D = f - s$. Thus zero Dijkstra fitness corresponds to the true optimum, indicating that the value found by that GP run is equal to s (given under each network in Fig. 1). In order to have a fitness measure which is more representative for comparing several networks, we use another measure which we call P_A (percent above the optimum) and define it as $P_A = (f_D/s) \times 100$.

Use of Multiple Populations: We decided to test our approach on a set of five networks, as well as explore the use of multi-population GP (MP). Our implementation used `lilgp` (v1.02) [5], which is clearly documented, regularly upgraded, and supports multiple populations with arbitrary exchange topologies. We used two distinct MP techniques called MP-Ring Architecture (MP-R) and MP-Injection Architecture (MP-I) [6]. In the ring architecture (Fig. 2) each sub-population chooses its five best individuals and sends it to the next sub-population in the ring. Each sub-population takes the individuals sent to it, and uses them to replace its five worst members. This is done after a specified number of generations. On the other hand, the injection architecture is a hierarchical arrangement. In the three sub-population example shown, sub-populations 1 and 2 both send their five best solutions (total ten) to sub-population 3. Sub-pupulation 3 replaces its ten worst individuals with those received from the donors.

Results: For each network fifteen runs were carried out, five for each of the three techniques: namely uni-population GP (UGP), MP-R and MP-I. Each run was given 100 generations for uniformity. A population size of 600 was used for all runs. All other control parameters used were Koza's default parameters [1]. Fig. 3 shows the Dijkstra fitness (f_D) for each generation using the three techniques on Network 2 with the same random seed. The delay in convergence using MP can be clearly seen. Table 1 shows the overall performance of GP on the five networks. The paradigm performed extremely well on Network 3 with the GP result being only 0.57% above the optimum. It was under 3% for Network 1, and under 10% for Network 2. For Networks 4 and 5 the performance of GP is certainly poor. For four of the five networks, the best results were obtained by all three techniques; for Network 2, however, MP-I obtained the best result.

The robustness of some of the best evolved solutions was examined by testing their ability to find near-shortest path routings in their target network, but this time with more than one link failure. In general, we found them able to produce near-shortest path routes in the majority of (connected-network) cases with two or even three failures.

Conclusions & Further Work: The above results demonstrate that genetic programming (GP) has some limited potential, for small networks, to evolve near-optimal fault-tolerant routing rules which are robust enough to be able to solve a high proportion of multiple link failures. Overall, though, this approach lacks adequate performance even for modest-sized networks. In addition, our comparison of uni-population and multi-population GP is arguably inconclusive (*cf.* [6]). Currently, the first and third authors are investigating evolving distributed software agents for telecommunications network routing and restoration, rather than centralised routing rules, in an effort to obtain more scalable results.

Acknowledgements: Initial experiments on Network 1 were undertaken by the second author as part of an M.Sc. in Telecommunication and Information Systems and subsequently continued by the first author as part of a Ph.D. project, both supervised by the third author. The second author was sponsored by the RAF, and the first by the Government of Pakistan.

References

[1] KOZA, J.R.: ‘Genetic Programming: On the Programming of Computers by Means of Natural Selection’ (MIT Press, 1992)

[2] DIJKSTRA, E.W.: ‘A note on two problems in connexion with graphs’, *Numerische Mathematik*, 1959, **1**, pp. 269–271

[3] GROVER, W.D.: ‘The Selfhealing Network: A fast distributed restoration technique for networks using digital cross-connect machines’, IEEE Global Conf. on Communications 1987, pp. 1090–1095

[4] KIRKWOOD, I.M.A., SHAMI, S.H., & SINCLAIR, M.C.: ‘Discovering simple fault-tolerant routing rules using genetic programming’, Intl. Conf. on Artificial Neural Networks and Genetic Algorithms, Norwich, UK, April 1997.

[5] ZONGKER, D. & PUNCH, B.: ‘lil-gp 1.0 User’s Manual’ (Michigan State University, 1995)

[6] PUNCH, B., ZONGKER, D. & GOODMAN, E.: ‘The royal tree problem, a benchmark for single and multiple population genetic programming’ in ANGELINE, P.J. & KINNEAR, K.E., Jr. (Eds): ‘Advances in Genetic Programming Volume 2’ (MIT Press, 1996)

List of Tables

List of Figures

Network number	Target (s)	Best fitness	Best P_A	Technique responsible
1	345	353	2.32	all three
2	622	684	9.97	MP-I
3	176	177	0.57	all three
4	375	603	60.8	all three
5	476	581	22.1	all three

Table 1: *GP results for the five networks*

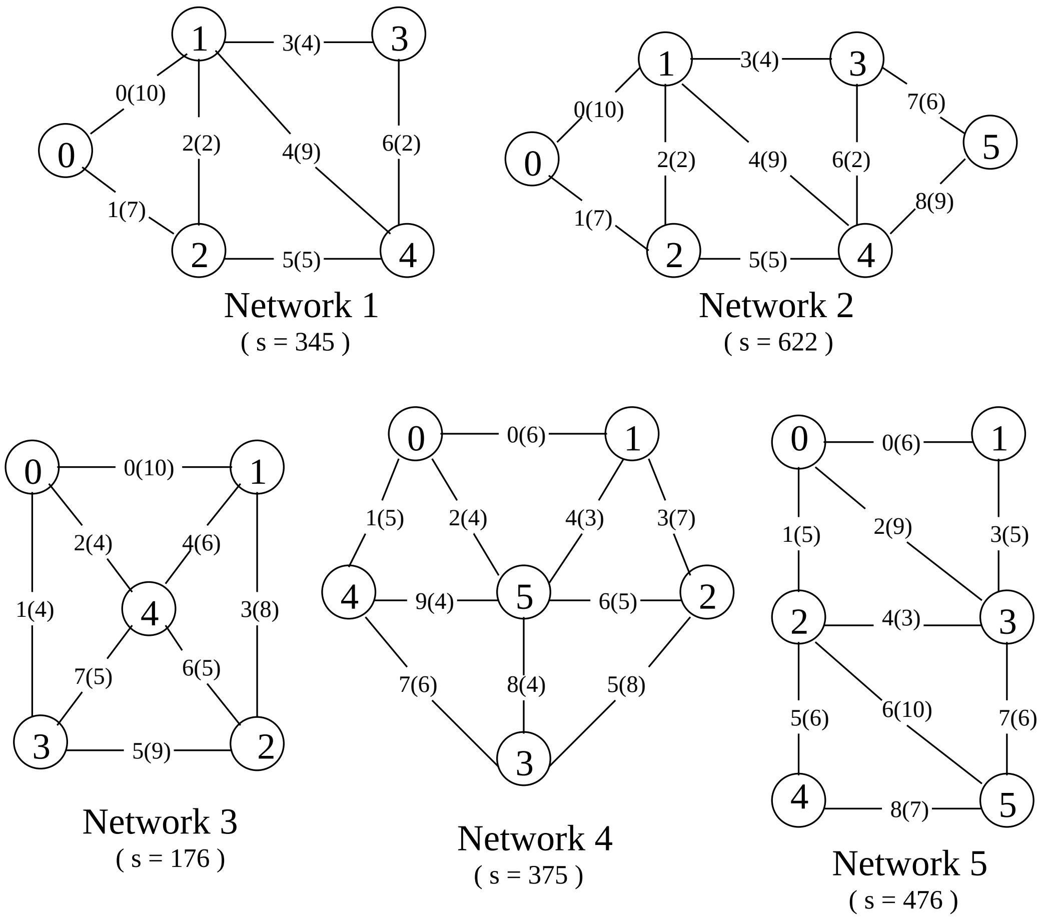
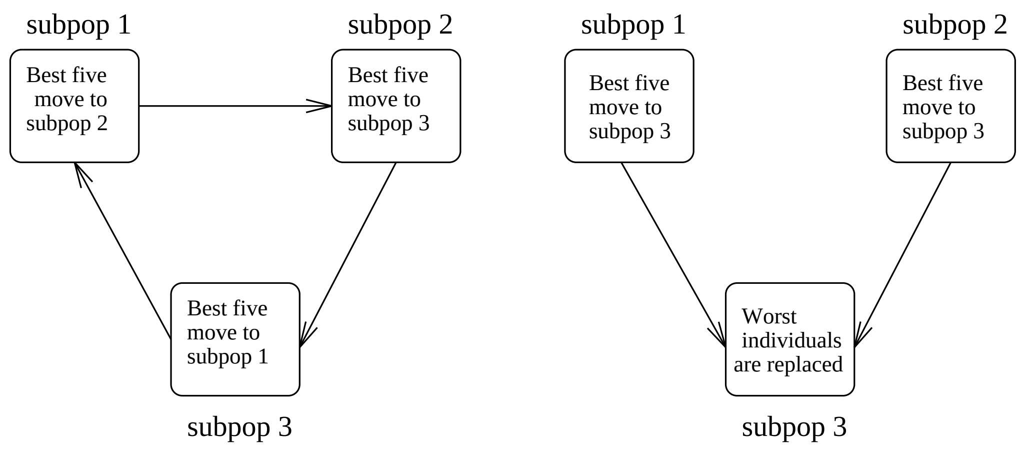


Figure 1: *The test networks*


subpop 1
Best five move to subpop 2
subpop 2
Best five move to subpop 3
Best five move to subpop 1
subpop 3
subpop 1
Best five move to subpop 3
subpop 2
Best five move to subpop 3
Worst individuals are replaced
subpop 3

(a) Ring Architecture (three subpops) (b) Injection Architecture (three subpops)

Figure 2: *Multi-population (two distinct techniques)*

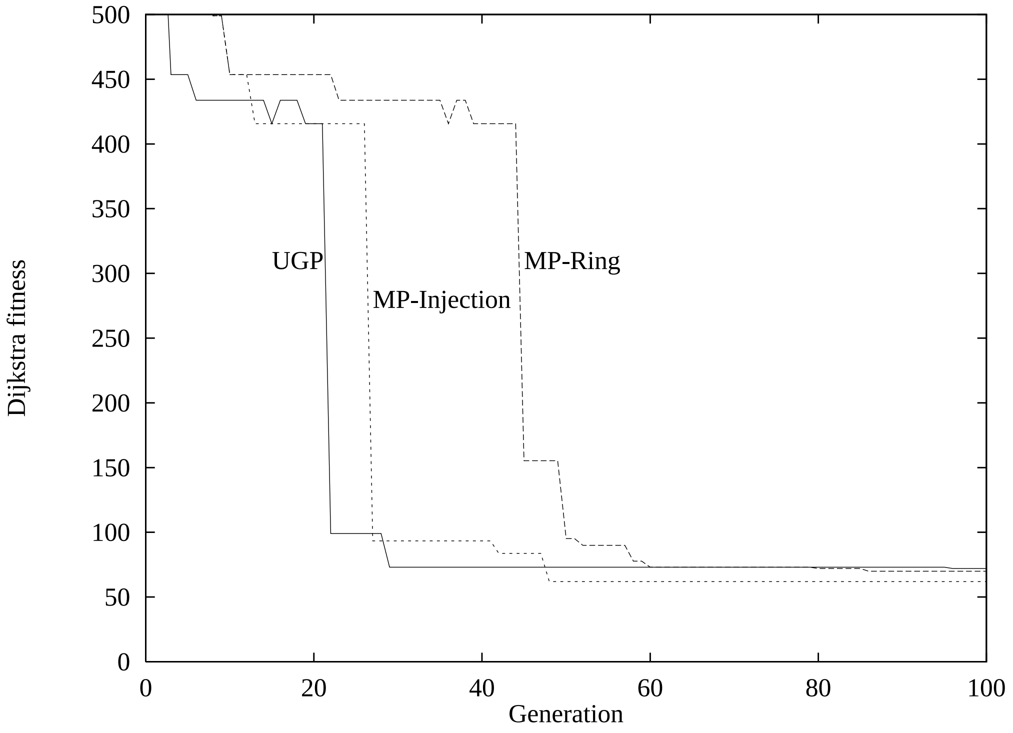


Figure 3: *The three techniques on Network 2*