Routing optimization by concurrent genetic algorithms

A. Genco[a,b], S. Lopes[b], G. Lo Re[b], M. Tartamella[b]
[a]*Dipartimento di Ingegneria Elettrica, Università di Palermo, Viale delle Scienze, 90128 Palermo, Italy*
[b]*Centro Studi sulle Reti di Elaboratori, Consiglio Nazionale delle Ricerche, Viale delle Scienze, 90128 Palermo, Italy*

Abstract

A parallel implementation of genetic algorithms is applied to routing protocols with low bandwidth consumption. In particular, the paper discusses the (LSP) *link state packet* protocols.
The first part of the paper deals with network topology and transmission parameters, together with the structure for storing the network. The second part discusses the Genetic Algorithm implementation. To this end, it considers the generation of the initial population that is a subset of all the possible paths connecting couples of nodes. As far as the mating and mutation policy is concerned, a strategy is proposed that allows the algorithm to replace segments of the entire path.
The implementation is carried out in parallel, thus letting different populations to evolve separately. Subsets of different populations migrate periodically to avoid the populations to persist in some local minima. These are the equilibrium states, where no better path, with a lower cost, can be found for a given period.
As for conclusions, comparisons between the results of the sequential and distributed implementations of Genetic Algorithms are reported.

1 Introduction

The world of telecommunication is growing towards a digital cellular radio system. This is better known as PCN (Personal Communication Networks) or DCS (Digital Communication System) standard as developed by the European Telecommunication Standards Institute [7].

Because PCNs use a high radio frequency, it is necessary to cover the area with many small cells related to base stations (radio bridge) and controllers (interface with the Public Telephone System). Therefore, in the future, it will be necessary to manage networks with a huge number of telecommunication resources. It is hence opportune to study new routing algorithms that better fit networks with a very large number of nodes.

A routing protocol is devoted to forward data across the network from a source to a destination [1]. The most popular types are the *distance vector* and the *link state protocols* (LSP) [4]. In particular the LSP protocols maintain the list of the neighbouring nodes of a router, together with the cost for reaching them. On the basis of this information (the LSP database), each router can execute the Dijkstra's algorithm to find out the shortest path.

The LSP packet is transmitted to all the other routers, which are responsible for storing the most recently generated LSP. At this point, the goal of the routing protocol is to generate a forwarding database that will allow the router to select the neighbour which to send to packets for any destinations. OSPF (Open Shortest Path First) [5], for example, after creating the initial routing environment, uses a path cost evaluation that is variable on an interface-by-interface basis. Then it creates a routing update message only when there is a change in the routing environment. By this way, there is a substantial saving on the network bandwidth. As a counterpart, the price paid for a lower bandwidth consumption is the increase of CPU and memory usage. The main cause of this is the complexity of the algorithm that is used to build the routing environment. In particular, OSPF adopts the Dijkstra's algorithm to achieve the optimum solution.

When a router uses the Dijkstra's algorithm, it computes all the possible paths to reach a given destination in the internetwork. The collection of these paths is known as the Topological Database, so that, when a failure occurs, a router simply checks its Topological Database for the next best path to the destination.

The Dijkstra algorithm is a very powerful one for graphs of small and medium size. Its complexity is polynomial and therefore, increases about linearly with the size of the problem. However, when we deal with very large domains, the response time can be very long. This is also due to the fact that each search cannot exploit the previous ones. In particular, this happens when some changes occur in the link states. A link may go down or up, or it may be less or more loaded with transmissions.

Actually, the above routing protocols do not consider this last aspect. They only take into account a fixed cost for each link. Nevertheless, some strategies can be adopted to avoid performing too many routing computations. One of these consists in collecting some changes before starting a new search for the optimum path. However, this can downgrade the efficiency of the network.

A suitable option for very large networks can be to adopt heuristic algorithms. We will achieve a solution in a shorter time, even if this may be a sub-optimal one.

Transactions on Information and Communications Technologies vol 9, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

To this end, we investigated some natural algorithms such as Simulated Annealing and Tabu Search. In this paper we discuss the advantages and the implementation of a Genetic Algorithm (GA) [3] for routing problems.

2 Genetic approach for the shortest path problem

A genetic algorithm emulates biological evolutionary behaviours to solve optimization problems. A GA comprises a set of individual elements such as in a population, and a set of biologically inspired operators that are defined over the population itself. A GA operates by means of a simple cycle:

1) creation of population (a subset of all the possible solutions);
2) genetic manipulation to create a new population.
3) selection of the best solutions;

In a GA it is important to consider the dimension of the population, and the different strategies for mate, mutation, selection and migration. These are the tuneable parameters that make the algorithm obtain good results.

2.1 The network topology

In our implementation we considered a particular reticular network in the shape of a torus with a variable number of nodes, as reported in Fig. 1. This choice was for implementation reasons. One reason is to have a fixed number of adjacencies in every node. The other one is to have both co-ordinates without bounds. This assumption makes the domain more free and simple to be explored.

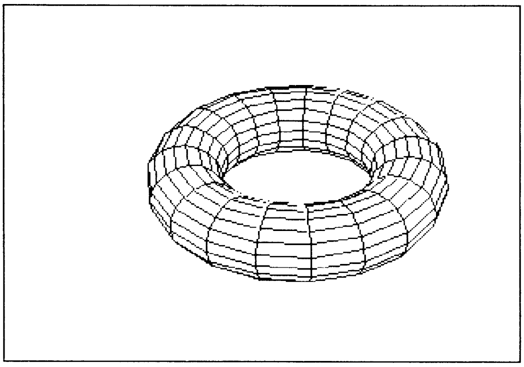

Fig. 1: The network topology

A randomly selected value was assigned to each link representing its cost. This criterion allows the highest degree of generality, but, in most cases, does not adhere to real situations. What happen in real networks is that a certain relationship exists between the number of hops and the connection cost.

For this reason the searches performed by of our GA implementation, rarely find out the optimum solution, in particular when the optimum path includes a number of nodes that are much more than those needed to cover the minimum geometrical distance. Some corrections could be done to build simulation cases more adherent to the reality, thus improving the efficiency of the method

2.2 Selecting the initial population

Each individual is a path that connects a source to a destination. It is represented as a sequence of steps towards one of the four neighbouring nodes. According to the grid definition it will be described as a chromosome with a chain of genes that are called: *up, down, left, or right.*

The algorithm creates the initial population randomly. However, the extraction is driven by a suitable distribution of probabilities. This is to avoid too winding paths. A greater preference is attached to the moves that more directly tend to the target. In particular the distribution adopted divides the probabilities between the two axes equally, but, within the same co-ordinate, it assigns the 70% of probability to the direction that reduces the geometrical distance from the destination.

Many trials were carried out varying the size of the population, according to the size of the network being explored. For instance, we started with a small network, just to verify the reliability of our implementation. For this case, populations ranging from 50 to 100 individuals were considered. The whole set of ranges investigated is reported in Table I.

Table I: population vs. network size

Network size (nodes)	Population range
900	50 ÷ 100
10,000	50 ÷ 200
40,000	50 ÷ 400
90,000	50 ÷ 1000

2.3 Mating and mutation strategy

According to the genetic paradigm, a population evolves with the changes introduced by the new individuals. They are originated by mates and genetic mutations. The first event type is performed by genetic algorithms with the crossover operator. This make two individuals generate a new one that inherits the genes from its parents.

In the case of an individual that represents one of the possible paths from a source to a destination, this is generated collecting two sub-paths that are taken from its parents. In particular, the operation consists in finding a node that is common to the two parents. Then the new individual is generated extracting the first sub-path from the first parent and the second sub-path from the other parent. The mating process generates a new population of the same size of the preceding one. This new population can completely replace the older one, or a selection can be performed to select the new individuals from both populations. With regards to the routing problem, the second strategy turned out to give better results.

As far as the mutation is concerned, this is performed at each generation selecting two random genes in each individual of a random sample. The size of this sample is previously fixed on a percentage basis, taking into account the main goal of a mutation that is to allow the search to escape from local minima. The smaller is the population size, the more frequent these occurrences are. Therefore, we adopted percentage values varying in the range [10,...,30], where the highest values have been used for small populations. The sub-path between the two nodes is rebuilt randomly, according to the same criterion adopted when generating the whole path.

We carried out several experiments changing the mutation strategy. One of these was to rebuild the sub-paths by means of the Dijkstra's algorithm. This strategy appeared to be very fast and effective, but only within the sub-paths. As a counterpart, the search of the global path turned out to fall in local minima, never allowing the search to escape from there. Other trials were carried out with a random extraction of the steps towards the destination node in the sub-path. Distributions of probability, that were different from the one adopted to build the global paths, were tested to the end of varying the locality of the sub-paths.

2.4 Selection strategy

The selection strategy is aimed to select the individuals that survive at each new generation. To this end, the individuals are listed in ascending order, according to the value of the objective function, that is the cost of the path. Then a random value is extracted in the range [1/n,1] wher n is the size of the population. Finally the simple function $y = (x^{-1})$, where x is the random value, is used to select the new individual in the above list.

As it can be noted, this strategy attaches a greater probability to the individuals at the top of the list. Nevertheless, the function allows the algorithm to select individuals of any position in the list, thus preserving the genetic characteristics of the whole population.

3 Parallel implementation

Since early implementations, Genetic Algorithms were conceived to run in parallel. In the natural reality, different populations can evolve separately with some migrations taking place randomly. This scheme can be applied to our case study directly, or some different policies can be adopted to simplify the implementation. For instance, our implementation performs the migration at each fixed period of generations.

The migration scheme is based on an oriented circular list where the populations are linked together. Therefore, a migration can occur in one direction from a population to its subsequent neighbour.

Many experiments were carried out to select the most suitable period at which to perform migrations. Good results were achieved fixing such a period at four generations.

The migrations are performed asynchronously. Each population sends its best individual, and does not wait for acknowledgement from the receiver. The receiver checks for arrivals at each generation of its own. When this happens, that population replaces its worst individual with the foreigner just arrived. As well as mutations, migrations act so as to let the populations escape from too stable situations, with all the individuals being very similar among them.

The asynchronous behaviour of communication makes it possible to implement the application with parallel systems that can be homogeneous or not. In particular, we carried out the experiments on a cluster of workstations that are connected by means of an Ethernet LAN. We also used the PVM (Parallel Virtual Machine) [6] software to arrange the parallel environment on the distributed system. This allows programs to include some primitives for data exchange and synchronisation in their C or Fortran source.

The processes that simulate the populations are arranged in a master-slave scheme, where the master is devoted to function as an I|O server and to create the slave processes.

According to the different processing rates of the used workstations, the master process performs a suitable allocation of processes among them, trying to distribute the working load equally. Then it waits for messages from the slave processes and append every incoming better result to the output file.

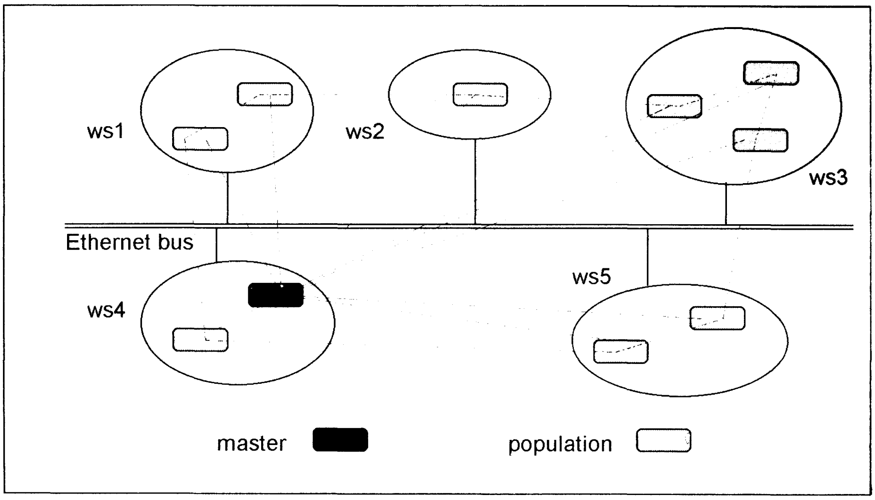

Fig. 2: The distributed environment

4 Experimental results

Many experiments were carried out considering networks of different size. A set of trials were conducted with a sequential implementation, and another set by means of the distributed environment.

Firstly, we ran the Dijkstra's algorithm to find out the optimum solution of each problem. As expected, Dijkstra performed faster and better than GA for small and medium problems (for instance till to 10,000 nodes). For larger problems it is not possible to perform direct comparisons, because Dijkstra gives the optimum in a long time, while GA gives an approximated solution in a shorter time. For instance, as for the 90,000 node network, Dijkstra achieves the optimum solution within 12 minutes, while GA gives its first solution after 59 seconds, but with an error of 50 %. If we let GA continue searching, after 10 minutes the error lowers till to 10 %.

Another consideration deals with the problem generation. This is performed on a random basis and generates spaces of solutions with an unpredictable complexity. Consequently, it may happen that small networks may be harder to be tackled than some larger ones.

Nevertheless, Table II shows some comparisons between the sequential and the distributed implementation of GA. The reported results are expressed as mean values, because there are big differences even among examples with the same network size.

Table II: GA results

problem size (nodes)	1st solution (% error - sec's)	sequential response time (10 % error)	distributed response time (7% error)
10,000	60 - 0.7	10	5
40,000	60 - 5	80	57
90,000	50 - 59	600	384

The second column of the table reports the percentage error of the first solution that is randomly generated by GA, together with the response time. The third column gives the mean value of the response time to achieve an approximation of 10 %. These experiments were carried out sequentially, that is executing only one process per example. The fourth column deals with the distributed implementation and reports the times taken to reach an approximation of 7%. As it can be observed, the distributed executions gave results that were always better than the sequential ones. Improvements were achieved for both execution time and result approximation.

5 Conclusions

The obtained results entitle us to believe that GA are a good option to tackle routing problems for large networks. As well as other heuristic methods, GA can give good approximations of the optimum solution in a time actually shorter than the one needed by a deterministic algorithm such as the Dijkstra's algorithm. Better results could be achieved adopting some suitable heuristic criteria that are capable of making the simulated networks better fit realistic ones.

Furthermore, execution time can be notably lowered by means of a parallel approach. This also succeed in finding out better approximations.

The presented simulations were carried out by using a distributed system. This was arranged by means of a workstation network and a free software such as PVM. The use of a heterogeneous distributed system was possible because of the asynchronous behaviour of the co-operating processes that simulate populations and their evolution.

Acknowledgements

This study has been supported by MURST, the Ministry of University and Research Scientific and Technological of Italy, and by CNR, the National Council of Researches of Italy.

References

1. Radia Perlman, Interconnections: Bridges and Routers, 1992, Addison Wesley
2. Clarck, Policy Routing in Internet Protocols, RFC1102, 1982
3. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, 1989, Addison Wesley
4. Mark Dickie, Routing in today's internetworks, 1994, VNR Communications Library
5. Moy, J., Open Shortest Path First, version 2, RFC 1247. Network Information Center, SRI International. Menlo Park, CA
6. G.A. Geist, V.S. Sunderam, Network-Based Concurrent Computing on the PVM System, Concurrency: Practice and Experience, VOL 4, N. 4, 1992
7. IBM, An Introduction to Wireless Technology, International Technical Support Centers.