Java based Distributed Genetic Programming on the Internet

Fuey Sian Chong
School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK
F.S.Chong@cs.bham.ac.uk http://studentweb.cs.bham.ac.uk/~fsc/

Abstract

A distributed approach for parallelising Genetic Programming (GP) on the Internet is proposed and its feasibility demonstrated with a distributed GP system termed DGP developed in Java. DGP uses Java Servlets to handle communication between distributed machines and a population pool to facilitate migrations. It also has an interactive user interface for controlling a run and graphical displays of the evolution process. DGP is run successfully across the world over the Internet on heterogeneous platforms without any central co-ordination. The run results and the outcome of an experiment to determine DGP's performance are reported together with a description of DGP.

1 INTRODUCTION

Parallelising GP on the Internet is an ideal way to realise the increase in computing power required to solve larger and harder problems. Existing work on parallel GAs include the global, coarse-grained and fine-grained models [3], [4] and [6]. Parallel GPs have been implemented on a network of transputers [1]. This paper discusses our distributed approach to parallelise GP on the Internet and describes DGP. DGP allows anyone with access to the Internet to harness the huge number of unused CPU cycles it contains to evolve programs. The use of Java enables distributed systems to be implemented on heterogeneous hardware and software on the Internet. DGP is run successfully across the world over the Internet. Possible motivations for Internet users to participate in the run include altruism, fun, learning about GP, general interest and more tangible benefits such as gaining publicity through contributing CPU cycles towards a shared system and the ability to run their own GP experiments on DGP.

2 THE DGP SYSTEM

The implementation consists of several sequential genetic programs, termed DGP clients, each with its own

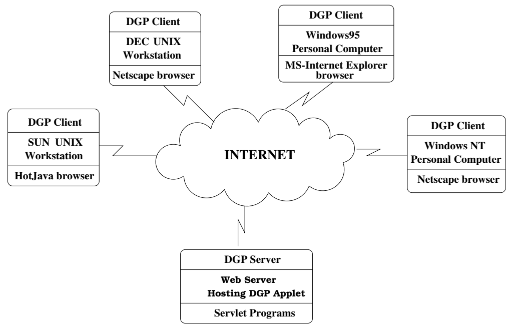

Figure 1: Four workstations of different types work asynchronously over the Internet on a common GP problem using a distributed population, communicating occasionally via Servlet programs.

population deme, running asynchronously over the Internet and communicating with each other through the DGP server which is the web server hosting the applet, to solve a common problem (see Figure 1).

2.1 POPULATION POOL STRUCTURE

Migrations among demes occur through the common population pool located in the server machine. The population pool stores migrants exported to it for subsequent imports. Machines with different speeds can therefore join and leave the GP run dynamically.

Each deme has a single buffer within the population pool to contain the migrants it exports and from which other demes retrieve their migrants. Figure 2 depicts the structure of the population pool, a Java `Hashtable` object. An individual buffer in the population pool can be identified by its "ClientID", a unique name made up of the IP address of the DGP client machine and the value of a counter object which keeps track of the number of times DGP is run. Each entry contains a symbolic string representation of the individual, the individual object itself and an **access list** which is a Java `Vector` object. The access list keeps track of the ClientID of runs which have already been sent the individual associated with the list. It prevents a deme from receiving the same individual from the same pop-

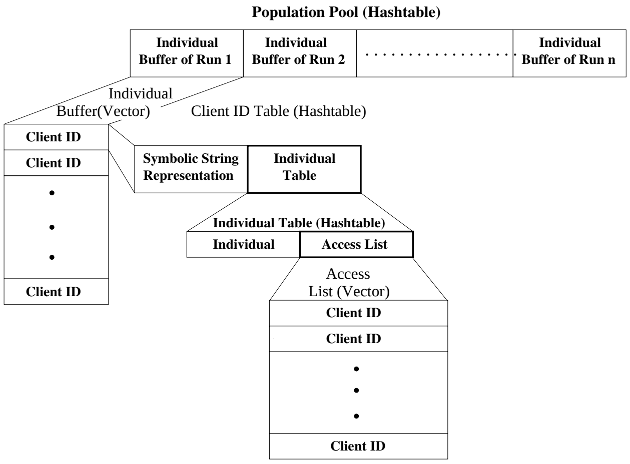

Figure 2: Structure of the Population Pool

ulation more than once during different migrations. The existence of duplicate individuals in the population pool does not improve the genetic variety in a population and wastes computational efforts [7].

2.2 MIGRATION STRATEGY

The overall algorithm of DGP is explained with a flowchart in Figure 3. To facilitate communication, server applications, written as Servlets, were developed for execution by the DGP server. These servlets are the **SeedServlet**, **PopServlet** and **StatServlet**.

At the beginning, when a run is started with the *Start* button, the DGP client invokes the **SeedServlet** servlet and sends its GP settings to it for processing. The SeedServlet servlet in turn returns a unique randomiser seed and a run ID to the applet client. The run ID is used in every communication with the web server and uniquely identifies each run to the servlet programs at the server. The randomiser seed is used for the creation of the initial random population and fitness cases and the selection of crossover points and individuals for crossover and reproduction. Unique randomiser seeds ensure that independently created populations are not identical to each other.

The DGP client then evolves its population using crossover and reproduction operations. After completing a user specified number of generations, migration of individuals occurs. The DGP client invokes the **PopServlet** servlet to deposit a user specified number of the best solutions that have not been exported before into a population pool. It also sends user specified information about its run id and the number of migrants it wishes to import. Individuals that have already been sent will be tagged with "Sent" flags that prevent them from being exported again. Similarly, imported migrants will be tagged with "Sent" flags and prevented from being sent back to the population pool. Selected migrants are duplicated before being sent to the population pool.

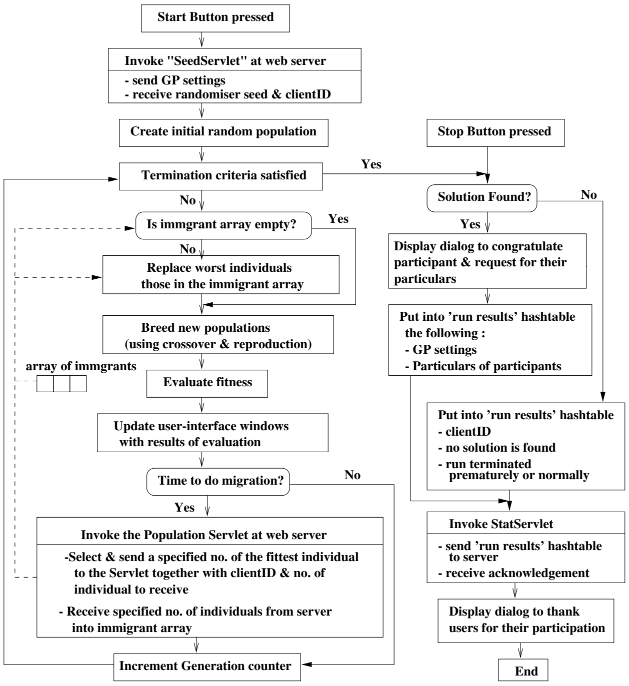

Figure 3: Overview of the DGP algorithm

At the web server, the PopServlet servlet responds to the DGP client's request for migrants by randomly selecting, from the population pool, a number of buffers equivalent to the number of migrants requested. It then retrieves one individual from each buffer using the Last-In-First-Out (LIFO) method. A random selection of buffers will cause all buffers to be selected with identical probability. As a result, the search does not concentrate on specific regions of the solution space but allows different demes to explore different solution spaces. The use of the LIFO method to retrieve individuals from their buffers is "greedy" in nature in that the most recently received individual, usually of higher fitness is imported by a deme to help achieve convergence. The less fit ones are then accepted into the population at subsequent generations to provide variety, if not convergence. In addition, selecting from more buffers achieves more variety in genetic materials than selecting more individuals from a single buffer because individuals in the same buffer are deposited by the same deme.

Upon receipt of the migrants, the DGP client replaces the worst individuals in its population with these migrants. Replacement takes place regardless of whether the worst individuals in the population are fitter than the migrants. This is to introduce genetic variety into the population.

A new population is then bred using crossover and reproduction operations. The whole process repeats until a solution is found or the target number of generations has been reached. The run can also be prema-

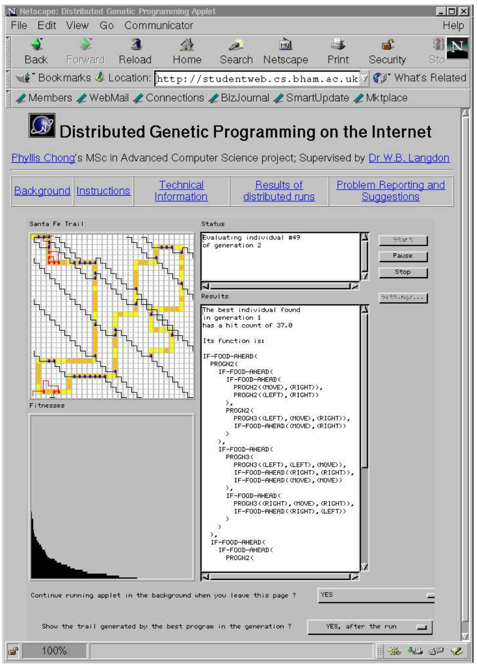

Figure 4: User interface panel of DGP

turely terminated with the *Stop* button. Upon termination, the **StatServlet** is then invoked and the best individual or solution is sent to it together with the GP settings of the run in which a solution is found, the fitness of the best individual and the generation number in which the solution is found or in which the run terminates.

2.3 FEATURES OF DGP

DGP is built to support a variety of problem domains. Two different applications, Symbolic Regression and Artificial Ant [7], have been successfully implemented on the DGP platform. New problems can be supported simply by implementing problem specific Java methods and specifying designated problem names to DGP.

DGP also allows users to take part in a run without being connected to the Internet all the time. This is achieved by designing the DGP applet to work offline as a Java application. Users only need to connect to the Internet during start up of runs to obtain randomiser seeds for creating initial populations and during termination of runs to distribute results. During migrations, users can choose whether to reconnect to the Internet to distribute individuals. Running DGP offline also enables users to terminate runs prematurely, save the states of runs to disk and resume the runs later.

2.4 USER INTERFACE

The web page for DGP's user interface can be found at `http://studentweb.cs.bham.ac.uk/~fsc/DGP.html`. Figure 4 shows the main web page.

The user interface controls a run and allows users to inspect and dynamically modify migration and run control parameters. Normally, the browser will suspend the applet thread when the user leaves or scrolls off the applet page and will resume execution when the user returns to the applet page. However, users can run DGP in the backgrouond when browsing another web page so that idle CPU cycles can be used more efficiently. The user interface shows the progress of the run and a symbolic representation of the best program found in each generation along with its fitness and number of hits. It also displays a histogram of the number of hits of all individuals in a population and graphical presentations of programs generated by GP.

3 WORLD WIDE TESTS

Participation was solicitated from the GP mailing list to run DGP to solve the Artificial Ant problem. A total of 237 runs of DGP were conducted across 11 countries on the Internet. Results proved the feasibility of DGP, with trouble free operation and without central co-ordination. A total of 33 correct programs were found out of 102 complete runs.

4 PERFORMANCE OF DGP

An experiment was conducted to study and compare the performance of DGP to another three approaches. The three approaches are the conventional standalone GP involving one large population and the distributed GP application studied in our previous work, set up in both "coordinated" and "uncoordinated" modes. The two modes differ in whether coordination and efforts were expended to keep the workload and processing speeds of machines involved in the experiment the same throughout the experiment.

The distributed GP program developed in our previous work was based on the coarse-grain model used by [8] and was written as a Java application to run on a network of UNIX workstations and PCs. All machines are interconnected and communication between them takes place directly in a "peer-to-peer" way without having to go through another node for coordination. Machines participating in the run have to be pre-determined before the run begins so that all machines know who to communicate with.

4.1 EXPERIMENTAL SETUP

All four approaches were set up to perform 50 runs, each with 50 generations. The standalone conventional GP was set up identically to the corresponding experiment described in [7] and was run on a single workstation with a single large population size of 900. Each of the other three distributed GP approaches has evolutions carried out simultaneously on three workstations, each with a population size of 300.

The performances of different approaches were compared using their probabilities of success at each generation of their runs. The probability of success for

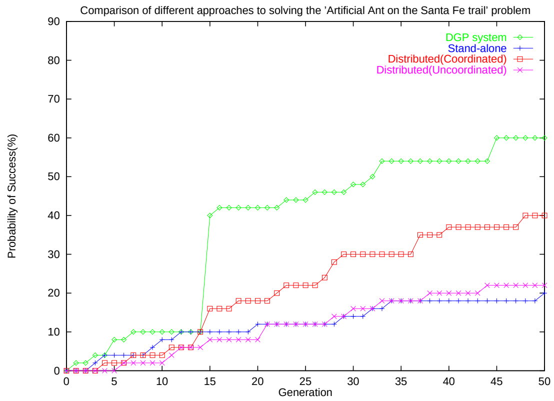

Figure 5: Performance curves for the Santa Fe trail problem comparing DGP with standalone GP, the Coordinated distributed GP and the Uncoordinated distributed GP

a particular generation is the cumulative probability that, by generation i, at least one program in the population causes the ant to collect all 89 pieces of food along the Santa Fe trail within the allowed time. For example, if 10 out of 50 runs are successful by generation 20, the cumulative probability of success is 20%.

4.2 ANALYSIS OF RESULTS

Figure 5 shows that DGP's probability of success is higher than those of other approaches in almost all generations. In particular, the number of solutions generated out of the 50 runs by the DGP system, the standalone GP, the coordinated and uncoordinated distributed GPs are 30, 10, 20 and 11 respectively.

It is interesting to note that the probability of finding a solution for DGP increased sharply from 10% in generation 14 to 40% in generation 15 when the first migration took place. Similar patterns of increase were also evident in the other two distributed approaches with migration. However, their increases were not as high as that of DGP. The higher probability of success for DGP is mainly due to the use of a population pool to store migrants exported to it for subsequent imports. In this way, all demes have the opportunities to import highly fit migrants from the population pool to produce better offsprings no matter when they start or end their evolution. Once a solution is found and exported to the population pool, the probability of importing a solution from the population pool in the first migration is very high. This explains why the probability of finding the correct program increases sharply from 10% in generation 14 to 40% in generation 15 when the first migration occurs. For our previous two approaches, exported individuals can only be imported by demes which are evolving concurrently with the exporting demes. This means that a slow machine that is left evolving on its own when other machines have completed their runs, will not be able to receive any migrants and will in fact, run as a standalone population. We observed that the probabilities of success for all distributed approaches generally increased when migrations took place after generations 14, 29 and 44. This demonstrates the advantage of migration where an exchange of individuals is essential to improve the quality of solutions generated [2].

5 CONCLUSION

Experimental results prove the feasibility of DGP and suggest its potential to solve larger and harder problems. New problems can be supported by implementing problem specific modules to DGP.

Further details can be found in [5]. The source code for DGP is available via anonymous ftp from site `ftp.mad-scientist.com` and directory `/pub/genetic-programming/code/DGPsrc.tar.gz`.

Acknowledgements

My gratitude goes to my supervisor, W.B.Langdon for his criticisms and ideas, Hansueli Gerber for his standalone GP source code on which my code is based and all those who have participated in the running of DGP on the Internet.

References

[1] D. Andre and J. R. Koza. Parallel genetic programming on a network of transputers. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 111–120, 1995.

[2] P. S. Berthold Kröger and O. Vornberger. Parallel genetic packing on transputers. In *Parallel Genetic Algorithms: Theory and Applications*. 1993.

[3] E. Cantú-Paz. A survey of parallel genetic algorithms. Technical report, University of Illinois, 1997.

[4] M. L. C.H. Pettey and J. Grefenstette. A parallel genetic algorithm. In *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 1987.

[5] F. S. Chong. Java based distributed genetic programming on the internet. Technical Report CSRP-99-7, University of Birmingham, 1999.

[6] M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In *Parallel Problem Solving from Nature*. 1991.

[7] J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. 1992.

[8] R. Tanese. Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.