G2DGA: An Adaptive Framework for Internet-based Distributed Genetic Algorithms

Johan Berntsson
School of Software Engineering and Data Communications
Queensland University of Technology, QLD 4001, AUSTRALIA
j.berntsson@qut.edu.au

ABSTRACT

The Internet is different from traditional parallel computing environments, and Distributed Genetic Algorithms (DGAs) for the Internet need to be designed to address these differences. This paper presents a framework for Internet island-model DGAs that uses adaptation methods to maintain efficiency and robustness in a volatile and dynamic run-time environment. The applicability of the methods is demonstrated on benchmark tests, and a real-world optimization problem in VLSI design.

Categories and Subject Descriptors: D.2 Software Engineering: Miscellaneous

General Terms: Performance Reliability Algorithms

Keywords: genetic algorithms, migration topology, adaptation, internet computing

1. INTRODUCTION

The Internet is the most powerful parallel and distributed computation environment in the world and the idle cycles and memories of computers on the Internet have been increasingly recognized as a huge untapped source of computation power. As a result, the research and practice on developing Internet-based parallel and distributed GAs have attracted a great deal of attention recently, with most attention focused on client/server architectures (e.g. [4, 12, 3]). Peer-to-peer (P2P) architectures have also been considered, most notably in the DREAM project [11].

Developing parallel computation applications on the Internet is quite different from in traditional parallel computation environments, such as multiprocessor systems, because the Internet differs from those in many respects. Firstly, its communication latency is significantly higher and communication bandwidth is narrower than traditional parallel computation environments. Secondly, it is dynamic and volatile since the number of participating computers and their performance cannot be predicated beforehand and they may withdraw at any time. Thirdly, because of security reasons, participating computers may not be able to communicate with each other directly. Fourthly, participating computers may be heterogeneous.

The results reported in this paper are part of a larger research project, G2DGA, aimed at developing a middle-ware for Internet-based genetic algorithms. The main approach used to address the volatility and dynamic nature of the Internet run-time environment is to use adaptation. Due to limitations of space only a broad overview of the system and the most significant experimental results can be included, but more detailed reports can be found on-line [1]. The framework is described in Section 2. Section 3 describes a method for adapting the migration topology, and Section 4 describes a method for sizing of population size and number of islands. Section 5 concludes the paper with an overview for future research.

2. INTERNET-BASED GA FRAMEWORK

While most Internet computing applications, including GAs, use the client/server model, this research uses a P2P model to make the system more scalable by reducing server bottle-neck and single point of failure problems. In contrast to pure P2P DGAs like DREAM [11], G2DGA is implemented as a hybrid P2P with two types of objects, (i) islands that run a GA process, and (ii) a supervisor that perform monitoring and adaptation. The supervisor creates the island objects and defines a migration policy that is sent to each of them, specifying the migration interval, rate, and a list of neighbouring islands. The islands run the GA and handle migration, which is asynchronous. The migrants are sent directly between the peers (islands), while the supervisor collects feedback data from the islands and is responsible for adaptation. By its hybrid design G2DGA can retain a global overview of the GA which provides opportunities for adaptation that are difficult to achieve in a pure P2P architecture, while avoiding the bottle-neck and single-point-of-failure problems associated with traditional client-server solutions since; (i) the supervisor is implemented as a P2P node. If the computer goes off-line or becomes busy, the P2P load-balancer will transparently move the supervisor object to another computer, and (ii) the supervisor is optional, and only used to improve the performance of the GA. The islands will continue to work without it.

G2DGA is based on G2P2P [9], which is a P2P distributed object framework based on .NET remoting. G2P2P uses a hash-based object addressing scheme to separate the

[1] http://plas.fit.qut.edu.au/Wiki/Users/JohanBerntsson.html

GECCO'05, June 25–29, 2005, Washington, DC, USA.

logic view of P2P nodes from the hardware they run on, making it easy to implement object migration. G2P2P also handles code distribution and fault tolerance transparently. .NET provides a safe execution environment and platform-independent exchange of data by XML.

3. DYNAMIC TOPOLOGY ADAPTATION

For island model GAs, the migration topology has a major impact on DGA performance [2]. The framework uses clustering in the supervisor on the feedback data (elite solutions received from the islands) to find groups of islands that work in similar partitions of the search space, and to optimize the migration topology with the goal of reducing the connectivity while maintaining good performance. The main steps are outlined in the pseudo-code below. Several strategies for *MakeTopology* are described in Section 3.1. If any parameter is to be changed, *UpdateIslandTopology* will send the new migration policy to the islands.

```
procedure Adapt()
  cluster_set=MakeClusters(data_set)
  new_topology=MakeTopology(cluster_set)
  UpdateIslandTopology(new_topology)
```

Clustering is an unsupervised learning method which divides data into natural groups automatically based on similarity. In the current application the problem is simplified by the fact that only a small subset of individuals is evaluated. The computational expense grows with the number of islands, which is much smaller than the total population. Furthermore, it is not necessary to find optimal clusters. A heuristic method that finds useful clusters for efficient adaptation is enough.

The clustering algorithm used in this work is K-medoid [8], which can be applied to all genomes, including those that can only provide nominal data, given a distance function that compares two genomes, e.g. Hamming distance for bit genomes. The aim of K-medoid is to partition the data set of n data points into K groups so as to minimize the total within-group sum of distances about K representative points, or *medoids*, among the data points. The stored number of individuals from each island is limited, and a newly arrived data point replaces the oldest when the buffer is full.

The setting of the number of clusters parameter K is a non-trivial problem. The Minimum Description Length (MDL) criterion from information theory is used to find to find a good K [5]. MDL which is a measure of how efficiently a given cluster model encodes the data set, and the system uses a bootstrap function to try $K \in [1 \dots K_{max}]$ and selecting the k with minimal MDL. The optimal K tends to be small compared to n, and $K_{max} = \sqrt{n}$ has empirically been found to be a reasonable setting.

3.1 Experimental Results

The clusters created by the clustering algorithm are used to create new migration topologies. The topologies tested are:

- Similar: Ring topology between the islands in each cluster, no communication between clusters.
- Diverse: Each island in a cluster communicates with all islands in the other clusters.
- Mixed: Ring topology between islands in each cluster, and one link to each other clusters.
- Fully: Each island communicates with all other islands, regardless of the cluster.

Many experiments have been conducted to evaluate each alternatives over a wide range of test problems. Figure 1 displays the running of F101 from the Whitley test suite [13] with the parameter settings pc=0.7, pm=0.005, 2-tournament, generational GA, 16 islands, 30 individuals/island, elite=1, 1-point crossover, 10 variable, 10 bits per variable, migrant rate=1 individual (best replaces worst), and migration interval=5 generations, averaged over 50 runs. *Connectivity* is also measured, defined as the number of one-way migration paths between the islands. For n islands, this means that a fully connected topology has connectivity $n(n-1)$, and a uni-directional ring topology has connectivity n.

Densely connected topologies have better average fitness and worse optimal performance. The similar topology has 7% of the connectivity of the fully connected, and performs worst in both categories. However, only a 3% increase in connectivity allows the mixed topology to improve the average fitness significantly, and to achieve the best optimal performance. The improved results can be explained thus: the clusters concentrate on exploring promising partitions in search space. The added connectivity in the mixed model is not big enough to force premature convergence, and the mixed topology keeps the number of clusters up compared to other dynamic topologies, dividing the islands into smaller groups and slowing down information exchange between the clusters. This seems to provide good balance between exploration and exploitation, which is key to good GA design [10].

The same set of experiments reported on F101 have also been carried out with the F102 and F8F2 functions with similar results. All experiments used ring topologies within the clusters. Other experiments have been conduced with fully connected topologies, and the general trend is the same, but the total level of connectivity is higher (typically 0.6 instead of 0.1 for mixed), and the optimal performance advantage smaller. More sparse intra-cluster topologies are clearly superior. The results suggest that clustering gives a big boost in performance (especially optimal) for a small increase in connectivity. This is especially evident in the mixed topology. Furthermore the method has been applied to a challenging real-world VLSI floorplanning problem [1]. The adaptive DGA has significantly better optimal performance, and the best found solution is better than the best solution reported in [1], even though the total population is smaller.

4. POPULATION SIZING

Finding the proper population size for a given GA problem is of crucial importance for good performance, and the intention of the proposed method is to find set of values for the number of islands and island sizes that provides a total population size big enough to solve the problem efficiently. Parameters such as crossover, mutation rates, and migration policy are important, but they are also fairly tolerant, while a proper population sizing is crucial and can to some extent overcome suboptimal settings of other parameters. If the population size or number of islands is too small the GA

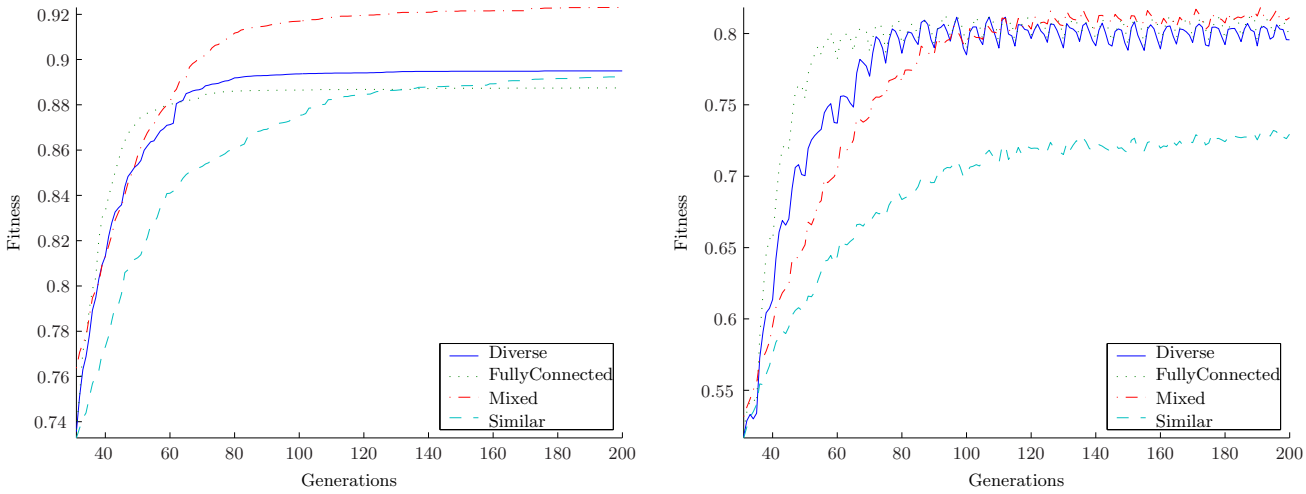

Figure 1: Continuous adaptation; optimal (left) and mean fitness (right)

will convergence prematurely, and too large values are inefficient. A manual approach to population sizing could be to first try with a small population and number of islands, and then increase each parameter until no further gain in performance is detected. The proposed method works in a similar manner, but instead of working in an ad-hoc way, the population adapter automates the process and hides it from the user.

The basic idea of the population adapter is to run several distributed GAs with competing sets of number of islands/island size settings in parallel. Let n denote the number of islands, and d denote the island size, which is the same for all islands. The total population is $n * d$. At any given time three DGAs (denoted DGA_0, DGA_1, and DGA_2) are run in parallel with the following set of parameters:

$$< n, d >, < n, d * 2 >, < n * 2, d >$$

This sets up a competition between the basic DGA_0, DGA_1 with more islands, and DGA_2 with bigger island population sizes. The DGAs are allowed to run until one of the DGAs overtake its competitors, or the termination criterion has been met.

The method has similarities to the parameter-less GA [6], but with significant modifications: (i) both *population size*, and *number of islands* are adjusted in parallel, (ii) the population adapter is both *competitive* and *collaborative*, and (iii) the population adapter terminates automatically when no further improvements are found.

4.1 Competitive Evaluation

Evaluation of DGAs can be cut if they are being overtaken by other DGAs, or converging. Overtaking is detected by comparing average fitness of a DGA with DGAs with bigger total population size, since it is unlikely that the smaller DGA with lower average fitness will succeed in getting better optimal results than the bigger DGA. The convergence criterion is problem dependent, and should be set by the user. Once convergence or overtaking is detected, the population adapter takes the following action:

- DGA_0 converges
  no action.
- DGA_1 and DGA_2 converged
  All DGAs restarted.
- DGA_1 overtakes DGA_0
  $d = d*2$, $DGA_1 \rightarrow DGA_0$, DGA_1 and DGA_2 restarted (using seeding).
- DGA_2 overtakes DGA_0
  $n = n*2$, $DGA_2 \rightarrow DGA_0$, DGA_1 and DGA_2 restarted (using seeding).

The intuition behind this algorithm is that three hypothesis are evaluated in parallel, and when a DGA overtakes its competitors, it is an indication of the need to adjust the parameter set in the direction suggested by the winning DGA. DGA_1 is testing if more islands are beneficial, and when it overtakes the other DGAs, the routine increases the number of islands in the next round of competitions. DGA_2, which is testing the benefit of increasing the population size on each island, works in the same way.

4.2 Collaborative Restart

The basic case of restarting an DGA is to simply reinitiate the population on each island. This may be inefficient, since each newly restarted DGA will need time to catch up with its competitors even if its population sizing parameter set is better. As an alternative the population adapter can use *seeding*. With seeding, each island in the newly restarted DGA reinitiates its population, but also inserts the best individuals from the other DGAs. E.g., if DGA_1 is restarted, each island in DGA_1 is seeded with the best individual from DGA_0 and DGA_2. In this way, the DGAs collaborate to give new DGAs a bias toward promising regions of the search space, which makes the population adapter more efficient.

4.3 Termination

The population adapter terminates when there is a relatively slim chance of finding a better set of sizing parameters than the current. The termination criterion should not reply on problem specific parameters, such as a known global optima, or a maximum number of generations. Rather, the population adapter uses the convergence status of each DGA. Since it is quite common that DGAs converge in early

Seeding	Success rate	Evaluations		Island size, number of islands						
		Mean	Stddev	80,8	80,16	80,32	160,8	160,16	160,32	320,8
Yes	10/10	407555	93437	-	-	1	3	3	2	1
No	10/10	463970	205637	1	2	1	4	2	-	-

Table 1: Population adapter with the Royal Road problem

trials because of the small values of n and d, it is not possible to terminate as soon as a convergence has been detected. The population adapter therefore does not terminate until (i) each DGA has converged at least once, and (ii) no new best solution is found during the detection phase. If a new best solution is detected, then the best solution so far is updated and the termination detection process is reset. In addition to the general termination criterion, problem specific knowledge can be used.

4.4 Experimental Results

Table 1 summarizes the outcome of ten runs of the population adapter, using one variant of the Royal Road problem [7]. To provide a reference for the population adapter experiments a series of experiments with manual settings were conducted, which suggested that a total population size of 2560 is required for good performance. The population adapter successfully finds the optimal value for all runs, but there are clear differences in the number of evaluations and the selected parameters, depending on whether seeding is used or not. Without seeding, most runs lead to comparatively small population sizes. With seeding, bigger population sizes are favoured, which is found in the manual experiments to be advantageous. This is reflected in the number of evaluations, where seeding has a lower average number of generations needed with a significantly lower standard deviation, suggesting that seeding leads to more reliable results. Additional experiments with F101 and the VLSI floorplanning problem described in Section 3.1 show similar results.

Although the population sizing experiments were carried out of standard GAs, the method can be applied to more advanced GAs, such as messy GAs or the Bayesian optimization algorithm, without any significant modifications.

5. DISCUSSION AND CONCLUSIONS

This paper has presented an adaptive framework for Internet-based island model genetic algorithms. Benchmark testing was used to evaluate different design options, and to compare adaptive and static performance. Furthermore, results on a real-world VLSI optimization problem were presented. In future research, we hope to extend on the current system and investigate the applicability of the proposed approach on optimization of other parameters that have an impact on communication overhead, such as migration rate and interval.

6. REFERENCES

[1] J. Berntsson and M. Tang. A slicing structure representation for the multi-layer floorplan layout problem. In *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2004*, volume 3005 of *Lecture Notes in Computer Science*, pages 188–197. Springer-Verlag, 2004.

[2] E. Cantú-Paz and M. Mejia-Olvera. Experimental results in distributed genetic algorithms. In *International Symposium on Applied Corporate Computing*, pages 99–108. Texas A&M University, Monterrey, Mexico, 1997.

[3] F. S. Chong. *A Java based Distributed Approach to Genetic Programming on the Internet*. Master's thesis, Computer Science, University of Birmingham, 1998.

[4] C. Gagnè, M. Parizeau, and M. Dubreuil. Distributed BEAGLE: An environment for parallel and distributed evolutionary computations. In *Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications*. Kluwer Academic Publishers, 2003.

[5] P. Grünwald. A tutorial introduction to the minimum description length principle. In P. Grünwald, I. Myung, and M. Pitt, editors, *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2004.

[6] G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 13–17. Morgan Kaufmann, 1999.

[7] T. Jones. A description of Holland's royal road function. *Evolutionary Computation*, volume 2(4), pages 409–415, 1995.

[8] L. Kaufman and P. J. Rousseeuw. *Finding groups in data : an introduction to cluster analysis*. Wiley, New York, 1990.

[9] R. Mason and W. Kelly. *G2-P2P: A Fully Decentralised Fault-Tolerant Cycle-Stealing Framework*, volume 44 of *ACSW Frontiers 2005*. ACM, 2005.

[10] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

[11] B. Paechter, T. Back, M. Schoenauer, M. Sebag, A. Eiben, J. Merelo, and T. Fogarty. A distributed resource evolutionary algorithm machine (DREAM). In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 951–958. IEEE Press, 2000.

[12] K. C. Tan, W. Peng, T. H. Lee, and J. Cai. Development of a distributed evolutionary computation package. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 1, pages 77–84. IEEE-Press, 2003.

[13] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence*, volume 85(1-2), pages 245–276, 1996.