

Graphed Evolutionary Computing

Rutger ter Borg

Léon Rothkrantz

14th December 2001

Knowledge Based Systems Group
Faculty of Information, Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft
The Netherlands

1 Introduction

Over the past several years, *Evolutionary Computing* (EC) has characterised itself to be an alternate way to search for complex computer structures. Many applications have successfully been taken into use, ranging from data mining tools to automated design of complex industrial plants or military aircraft. Several distinct types of EC have emerged during the last decade. The most prominent fields are Evolutionary Strategies (ES), Genetic Algorithms (GA) [3], Evolutionary Programming (EP) [4] and Genetic Programming (GP) [5].

Many evolutionary algorithms get stuck on a certain point in the search state space before in which it is not able to find improvements. This is called *premature convergence*, which also contributes to poor stability of the answer of the search. A broadly applicable method is proposed here which makes both the quality of the expected answer better, fights premature convergence and is suited for parallel execution. This paper discusses issues found on all types of EC, while focusing on GA for testing purposes.

This paper is organised as follows. After this introduction, the Graphed Evolutionary Computing (GEC) design is presented. Then, a Tree-based Graphed Genetic Algorithm (TGGA) is tested as an example of GEC. After which conclusions finalise this

paper.

2 Graphed Evolutionary Computing

The method proposed here is called Graphed Evolutionary Computing (GEC). It can be typed as coarse-grained parallel evolutionary computing [2, 6], in which exchange of fittest individuals of subpopulations occurs at (random) times, according to a migration model. This type of parallel evolutionary computing is typically only loosely synchronous, and work well even on distributed systems with very limited communications bandwidths.

GEC is represented by a directed graph and is represented as a “normal” graph, with *edges* and *vertexes*. The context of the edges and vertexes in GEC is described next.

- **Edges.** As in a normal graph, edges tie together vertexes to create a connected structure. An edge in GEC represents the path which the output of one evolutionary computing method should follow to the other(s). No restriction is laid upon the amount of incoming and outgoing connections, but the receiving evolutionary computing vertex will start as soon as all inputs are received.
- **Vertexes.** In GEC, a vertex represents any of the EC methods. All vertexes perform an entire evolution and should have access to parameters applicable to the search method and the search itself. As each vertex can have its own properties, it is possible to create a mix of evolutionary

computing methods all with its individual settings in one environment.

Communication only takes place if an entire evolutionary run is complete, which makes the communication bandwidth requirements low. An important aspect of GEC is the amount of solutions sent over the edges. Two important properties play a role here, the *input size* and the *output size*. The input size is the number of solutions which are input to an EA, the output size is the number of solutions which represent the “answer”. Both are discussed below.

- **Input size.** The input size can vary, depending on several parameters as the type and parameters of the EC-vertex. If too many individuals are presented to the EC-vertex, a selection mechanism must be applied in such way that the size complies to the demand.
- **Output size.** If an evolutionary computing process finishes, a set of solutions is selected to migrate. If applicable, a selection mechanism is used to choose that part of the population which is most promising.

The setting of input and output sizes can be used to combine several *local optima* which not only keep the diversity but also the quality of the population as high as possible. Dynamic population sizes is another option. GEC is a very generic way of modelling an EC structure, as all other parallel types of EC can be represented with it. For example, a classical island-type parallel EC can be created by placing EC nodes in a circle, and creating two-way migration paths between neighbouring nodes. Another example is an hypercube like structure, of which a pyramid is the three-dimensional variant. The dependency of completion of evolutions of other EC nodes in the GEC has influence in the scalability of the total structure.

3 Test case: A Tree-based Graphed Genetic Algorithm

A genetic algorithm (GA) is used to test whether GEC is able to compete with sequential EC. A tree

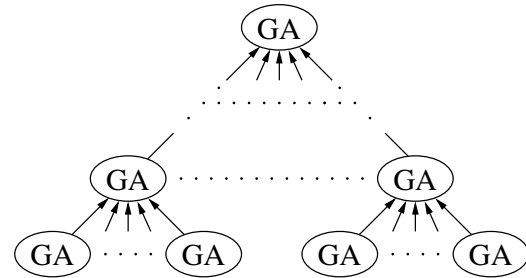


Figure 1: Tree-based Graphed Genetic Algorithm.

Algorithm 1 Elitist selection scheme recombination GA

1. Select parent 1 out of population with tournament selection of size 2.
 2. Select parent 2 randomly, not equal to parent 1.
 3. Create child 1 and child 2 with a 2-point crossover on both parents, no mutation.
 4. Children have to differ with parents.
 5. Out of the 2 children and 2 parents, the best 2 replace the parents.
-

shaped bottom-up directed graph is used as a basis for this test case of graphed evolutionary computing, which can be observed in Figure 1. Each GA-vertex represents an elitist selection scheme GA of which a generic description is made by Thierens [7] and is shown in Algorithm 1.

The order of execution of a Tree-based Graphed Genetic Algorithm (TGGA) behaves like a depth-first search with the exception that “higher” vertexes only can be started if all nodes attached to it are finished. TGGA executes the GAs in an asynchronous way, which allows for a stopping criterion of the searches to be introduced, which can lead to a more effective collective structure. The stopping criterion used in each TGGA can be described as follows: it stops its search if it was not able to find any improvement in as many crossovers as the population size (in one generation).

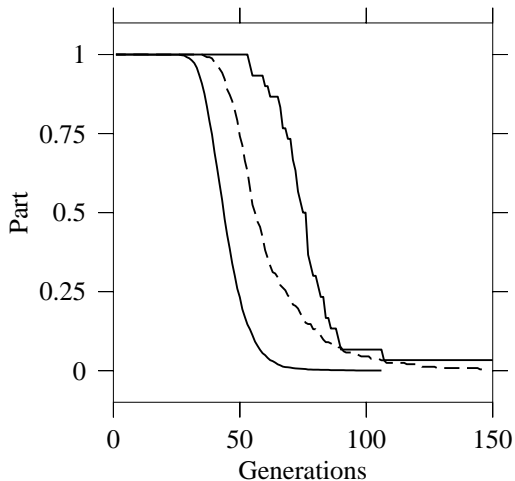


Figure 2: Stopping criterion.

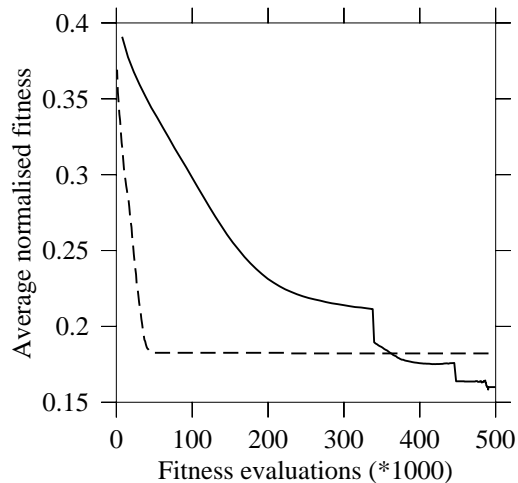


Figure 3: BiUgly 240 results.

3.1 Implementation

The TGGA is implemented to be used in a parallel computing environment, in such a way that each node in a heterogeneous computer environment can participate in the computations. It is based on a manager-worker parallel computing model, in which a manager node delegates all tasks to worker nodes. The manager node collects all outputs of all GA nodes, performs a fitness based ranking on them and sends the result to an idle worker node. Each worker will retrieve its parameters based on the location of the jobs in the tree. Parameters of a place in the tree is based on the depth of the node in the tree, so all nodes on the same “height” will use the same parameters.

All communications between nodes, like sending and receiving a number of individuals is done with LAM/MPI, which stands for a Local Area Micro-computer (LAM) [1] based on the Message Passing Interface (MPI) standard. It allows creating of flexible distributed algorithms on heterogeneous networks of computers. Almost no computing time is used for determining which process gets which task, the manager process keeps track of idle slave processes.

3.2 Testing TGGA

The quality of TGGA is assessed by experiments. The results of a TGGA are compared to the results of a serial GA (SGA), both based on Algorithm 1. Comparison of both methods is based on the normalised fitness value against the number of performed crossovers, which in this case equals the number of fitness evaluations (which is considered to consume the most computing power of a GA cycle). For all cases, averages are taken of 25 separate runs.

The TGGA deployed has depth 3, with 8 branches per node. GA nodes residing “higher” in the TGGA tree have to deal with more difficult problems, as their input consists out of a combined mixture of local optima. Therefore, nodes with smaller depth have a larger population size. Sizes are in order from root to bottom 480, 240, and 120. All nodes receive twice the needed individuals, by fitness ranking selection half of the input is discarded. This is done as some GAs are not able to find any interesting improvement at all. Also, a stopping criterion is used on all nodes of which is graphically depicted in Figure 2. It shows the part of the started nodes which is still running the evolution after a number of generations. The left solid line denotes the evolutions run at depth 2 (the bottom of the tree), the dashed line are

Table 1: Results of several benchmarks

| Problem | X-overs | SGA | D | TGGA |
|---------------|---------|-------|---|--------|
| H-IFF 256 | 413030 | 0.358 | 2 | 0.7033 |
| | 545183 | 0.358 | 1 | 0.391 |
| | 588267 | 0.358 | 0 | 0.025 |
| BiEasy 360 | 254131 | 0.040 | 2 | 0.165 |
| | 325228 | 0.039 | 1 | 0.056 |
| | 338572 | 0.039 | 0 | 0.000 |
| BiUgly 240 | 340224 | 0.177 | 2 | 0.214 |
| | 454656 | 0.177 | 1 | 0.176 |
| | 490848 | 0.177 | 0 | 0.165 |
| ZeroOne 512 | 654566 | 0.012 | 2 | 0.240 |
| | 882816 | 0.012 | 1 | 0.051 |
| | 928339 | 0.012 | 0 | 0.001 |
| RoyalRoad 512 | 10752 | 0.956 | 2 | 0.998 |
| | 18816 | 0.950 | 1 | 0.981 |
| | 25036 | 0.943 | 0 | 0.859 |

nodes at depth 1 and the most right line represents the root node.

As for the SGA, a population size of 2000 is used to keep the quest for diversity higher.

The results of the BiUgly 240 benchmark of TGGA and SGA can be observed in Figure 3. The solid line shows the results of the TGGA, the dashed the results of the SGA. The SGA is able to find improvements until a local optimum is reached near the normalised fitness of 0.18. As no mutation operator is deployed, SGA is hardly able to find any improvement at all after the local optimum is reached. TGGA executes 64 nodes at the bottom level, which translates to a slower convergence process in the beginning, but better results on smaller depths of TGGA nodes.

The results of all benchmarks can be observed in Table 1. The first column shows the name of the problem followed by the length of the GA string used in that benchmark. In next column shows the cumulative number of crossovers which equals the average used crossovers of TGGA at different depths. These depths can be found in column 4. SGA's and TGGA's performance is measured in normalised fitness values which can be found in columns three and five.

4 Conclusion

Graphed Evolutionary Computing is a generic and flexible way to model connected evolutionary algorithms with the use of directed graphs. Selection mechanisms at the input and output of evolutionary processes can effectively discard the non-promising areas of the search state space in several stages of the overall-search. The flexible nature of GEC makes it possible to model different types of parallel evolutionary computing, including mixtures of different evolutionary computing and its settings of parameters.

A tree shaped graphed genetic algorithm (TGGA) is designed as an example of applied GEC. Its bottom-up information flow exists out of a set of solutions (local optima), which TGGA successfully combines and mixes. This makes TGGA a very effective parallel GA. The LAM/MPI implementation of TGGA can be executed on a heterogeneous local area multi-computer [1] in an asynchronous way and has low communication bandwidth requirements.

References

- [1] Burns, G., et al., *LAM: An Open Cluster Environment for MPI*, Supercomputing Symposium '94, Toronto, Canada, 1994.
- [2] Cantu-Paz, E., *A survey of Parallel Genetic Algorithms*, IlliGAL Report 97003, Department of General Engineering, UIUC, 1997.
- [3] Holland, J., *Adaption in natural and artificial systems; an introductory analysis with application to biology, control and artificial intelligence*. Ann Arbor, University of Michigan Press, 1975.
- [4] Fogel, L., *Artificial Intelligence through simulated evolution*. Wiley, New York, 1966.
- [5] Koza, J., *Genetic Programming; on the programming of computers by means of natural selection*, MIT Press, Cambridge, MA, 1992.
- [6] Seredyński, F., *New Trends in Parallel and Distributed Evolutionary Computing*. In *Evolutionary Computation*, pp. 211-230. IOS Press, Amsterdam, 1999.
- [7] Thierens, D., *Selection Schemes, Elitist Recombination and Selection Intensity*, In Th. Back, editor, Proceedings of the 7th International Conference on Genetic Algorithms, pp. 152-159. Morgan Kaufmann, 1997.