Distributed Island-Model Genetic Algorithms Using Heterogeneous Parameter Settings

Yiyuan Gong and Alex Fukunaga
Graduate School of Arts and Sciences
University of Tokyo

***Abstract*—Achieving good performance with a parallel genetic algorithm requires properly configuring control parameters such as mutation rate, crossover rate, and population size. We consider the problem of setting control parameter values in a standard, island-model distributed genetic algorithm. As an alternative to tuning parameters by hand or using a self-adaptive approach, we propose a very simple strategy which statically assigns random control parameter values to each processor. Experiments on benchmark problems show that this simple approach can yield results which are competitive with homogeneous distributed genetic algorithm using parameters tuned specifically for each of the benchmarks.**

I. Introduction

Parallel computing has become pervasive. Modern high-performance computers are highly parallel clusters or grids composed of standard, multicore processors. Cloud computing, grid computing and peer-to-peer (P2P) environments are becoming increasingly mature, enabling many users to access vast computational resources. Genetic algorithms and other evolutionary methods can benefit significantly from this trend, since evolutionary algorithms can be parallelized and distributed straightforwardly.

One standard way to parallelize a GA on a cluster is an *island model* distributed GA, in which each processor executes the GA independently, and there is a periodic migration of individuals among the processors. Assuming that an appropriate problem representation and genetic operators have been designed, this general framework is simple to implement. However, obtaining good performance in practice often requires tuning the GA control parameters such as mutation and crossover rates.

In an academic setting, parallel GA experiments are usually performed on a cluster which is owned/operated by a researcher, and it is possible to allocate the time and resources for parameter tuning experiments. In practice, however, extensive parameter tuning is often not feasible. First, the time available for solving a problem may be quite limited due to urgent deadlines. Second, parallel resources may be very expensive for a GA practitioner. An increasingly common scenario is the use of massive computing resources on demand using cloud/grid computing services, where the user is able to use as many CPUs simultaneously as the budget will allow. In situations like this where one is paying for CPU usage, the cost of control parameter tuning experiments can be prohibitive.

While there has been significant previous work on setting parameters for parallel GAs [1], this remains an active are of research. Although there has been considerable progress in on-line self-adaptation/learning of control parameters (c.f., [2]), the drawback of these methods is that most of them are not completely parameter free – there is a set of meta-level control parameters which control the on-line adaptation. Furthermore, it is not yet possible to predict, in general, how well any of these techniques work in practice for a new, arbitrary problem.

In this paper, we consider a very simple approach to control parameter selection for island-model, distributed GAs. Instead of manually tuning control parameter values or dynamically adjusting parameters during evolution, we simply select a different, randomly selected set of control parameters (including population size, mutation rate, and crossover rate) for each processor. This *randomized, heterogeneous distributed GA* exploits the fact that sufficiently sampling the space of control parameter spaces can result in a near-optimal set of control parameters being discovered (for a single processor). Compared to most other methods, our approach is extremly simple, but surprisingly, the effectiveness of this naive method has not been investigated in the literature. We find that randomized, heterogeneous distributed GAs can perform at least as well as a hand-tuned, standard island-model GA, suggesting that this naive approach might serve as a baseline against which more sophisticated methods should be compared.

The rest of this paper is organized as follows. Section II reviews the standard, island-model, distributed GA which we studied. Section III first investigates the runtime distributions of a standard, sequential GA, which then leads to our randomized, heterogeneous approach to parallelization (Section IV). Section V presents experimental evaluation of this simple randomized approach on several benchmark problems. Section VI compares our approach to related work in the literature, and Section VII discusses the results and directions for future work.

II. An Island-Model Distributed GA

The basic approach to parallelization used in the rest of this paper is a standard, island-model (multiple-deme), distributed GA (DGA) for a computing cluster or cloud/grid environment [3]. In an island-model DGA, each processor contains a local population (*deme*), and each processor executes a GA on its local population. Although each processor runs mostly independently of the other processors, the processors occasionally send individuals to each other using a migration mechanism. In our DGA implementation, the processors are logically

978-1-4244-7835-4/11/$26.00 ©2011 IEEE

```
procedure DGA;
begin
  Initialize and evaluate local population P;
  while(termination condition is false)
    // Create next generation
    for i from 0 to |P|/2
      select 2 parents from P
      mate the parents to generate 2 children;
      evaluate children;
      insert children into P
    end for;
    //process outgoing migrants
    if (a new, local best-so-far individual B
      was found)
      send copy of B to a
        randomly selected neighbour of P;
    endif;
    //process incoming migrants
    while (incoming message buffer B ≠ ∅)
      h = first migrant from (B);
      remove h from B;
      replace worst individual in P with h;
    end while;
  end while;
end
```

Fig. 1. An island-model, distributed genetic algorithm

arranged in a $A \times B$ toroidal grid topology, so each processor has 4 logical neighbors (up, down, left, right), and migration occurs only between these neighbors. Migration occurs when a new, best solution within a population (i.e., a local elite) is found at some island I. This new local elite is sent to a randomly chosen neighbor of I. When an island receives a chromosome sent from its neighbour, the island replaces the worst chromosome in its population with the incoming migrant. Figure 1 shows the DGA pseudo-code which is executed on each processor. The `send` operation for sending outgoing migrants is non-blocking, and incoming migrants are queued, so all communications are asynchronous. Other migration strategies, such as those which explicitly seek to promote diversity [4] can also be implemented asynchronously.

In a standard implementation of the island model DGA, the processors are *homogeneous*. That is, all processors (deme) use the same set of control parameter values (population size, mutation rate, crossover rate, etc).

Island model DGAs are particularly well-suited for cloud computing and grid computing environments due to their low communication requirements. In GA applications where fitness function computations are extremely expensive, this communications bottleneck is not an issue. However, in applications such as numerical optimization where fitness functions are relatively fast, slow interconnects can mean that fine-grained GAs which assign, for example, a single individual per processor, will not scale well. On the other hand, island model DGAs require very little communication, and do not require synchronization, so this model is well-suited for cloud environments where high-speed interconnects between processors are not necessarily available.

III. On the Runtime Distributions of GAs

To motivate our randomized, heterogeneous DGA, we first consider the necessity for parameter tuning. It is well-known that the performance of a GA run on a given problem depends on its control parameters, and that choosing the wrong set of parameters could lead to unacceptable results. Even if we do not insist on the optimality of a set of control parameters, the No-Free Lunch theorems suggest that even if have a set of control parameters which work well on a large set of problems, it is dangerous to depend on such a control parameter to work on any particular problem, because it may perform very poorly on other problems [5].

Thus, if we assume that we run a GA only once in order to try to solve a particular problem, it is *risky* to rely on a single set of control parameters. In particular, suppose we run a GA once with a *randomly* selected set of control parameters. Intuitively, this would most likely result in very poor expected performance, and this would be considered a very poor strategy.

Now, suppose that we run the GA many times, but for each run, we use a different set of control parameters. How does the probability distribution of GA performance look like?

We investigated the variance of performance among runs of a single population, sequential GA (the single-population, single-core version of the code in Figure 1), with randomly selected sets of control parameters (mutation rate, crossover rate, and population size). We used the well-known Schwefel, Rastrigin, and Griewangk functions as benchmarks:

$$Griewangk(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \Big(\cos\big(\frac{x_i}{\sqrt{i}}\big) \Big) \quad (-512 \le x_i < 512)$$

$$Rastrigin(x) = 10n + \sum_{i=1}^{n} \big(x_i^2 - 10\cos(2\pi x_i) \big) \quad (-5.12 \le x_i < 5.12)$$

$$Schwefel(x) = \sum_{i=1}^{n} -x_i \sin\Big(\sqrt{|x_i|}\Big) \quad (-512 \le x_i < 512)$$

These three benchmarks are minimization problems. The Schwefel function is a deceptive function where the global minimum is located far from the next best local minima. The Rastrigin function is highly multimodal, with regularly distributed local minima. The Griewangk function is similar to Rastrigin's function, with many widespread local minima.The dimensionality (n), which controls difficulty, was set to 40 for these three benchmarks.

For each benchmark problem, we ran the single-population, sequential GA 100 times. In each run, the mutation rate and crossover rates were selected uniformly from $[0.0, 1.0]$, and the population size was selected uniformly $[10, 100]$. A

standard binary chromosome representation was used, and roulette selection was used. On each run, we measured the amount of time required to find a solution with a target fitness score. The runs were given a time limit of 300, 360, and 600 seconds for the Griewangk, Rastrigin, and Schwefel functions, respectively. If a solution was not found by the time limit, the GA timed out and terminated.

Figure 2 shows the distribution of runtimes for the Griewangk function. We grouped the runtimes of the 100 runs in bins of 10 second intervals, and plotted the frequency of runtimes. Runs that had not reached the target solution quality by the deadline are counted in the bin corresponding to the time limit. Similarly, Figures 3 and 4 show the runtime distributions for the Rastrigin and Schwefel functions, respectively.

First, note that taking 100 random samples is roughly equivalent to coarse-grained parameter tuning. There are only 3 control parameters (mutation rate, crossover rate, population), so it is likely that the best parameter setting from 100 random samples is a near-optimal control parameter setting.

We observe the following:

- For all three benchmark problems, a large fraction of the random control parameter settings resulted in very poor performance (timeouts).
- For all three benchmarks, a substantial portion of the probability distribution is close to the best (leftmost) bins,
- For all three benchmarks, multiple runs (out of 100) successfully solved the problem in less than 10 seconds.

In other words, for all three problems, random parameters perform very poorly in most cases, which is to be expected. However, there is also a substantial fraction of random parameter settings which actually successfully solve the problem quickly. Our results indicate that in a single-population GA, getting "good enough" performance is not very difficult given enough tries, since 100 random samples is sufficient to generate multiple "hits" (successful runs).

These characteristics of the runtime distributions of random GA configurations is related to the well-known phenomenon of long-tailed distributions for combinatorial search algorithms [6]. In essence, the runtimes of search algorithms tend to *not* be distributed as a normal distribution. Instead, the runtimes have "long tails", which means that it is quite likely that any particular run could perform extremely poorly, but a set of shorter runs can successfully sample the "good" side of the distribution and solve problems quickly. Here, we have shown that a similar situation exists for genetic algorithms on our benchmark problems. While techniques that exploit this phenomenon such as periodic restarts [7] and portfolios [8] have been previously applied to sequential GAs, this earlier work, as with most of the work in the GA literature, focused on measuring fitness scores as a function of the number of individuals evaluated, so the long-tailed nature of the runtime distributions was not made clear. Here, by treating a target fitness value as a "solution", and measuring the time required to reach a solution, we were able to clearly observe the long-tailed distribution phenomenon as applied to random configurations of a sequential GA.

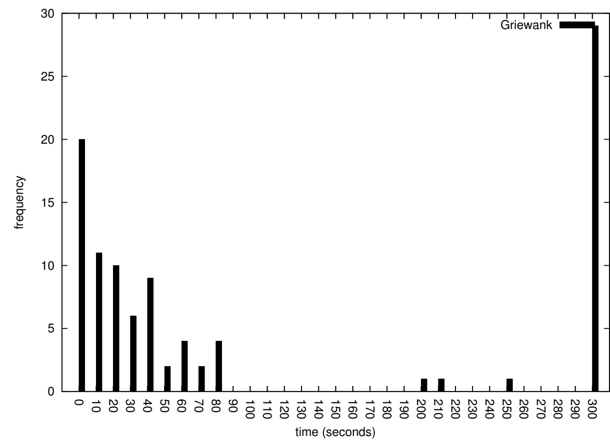

Fig. 2. Runtime distribution for Griewangk function (time to reach target score of 0.5)

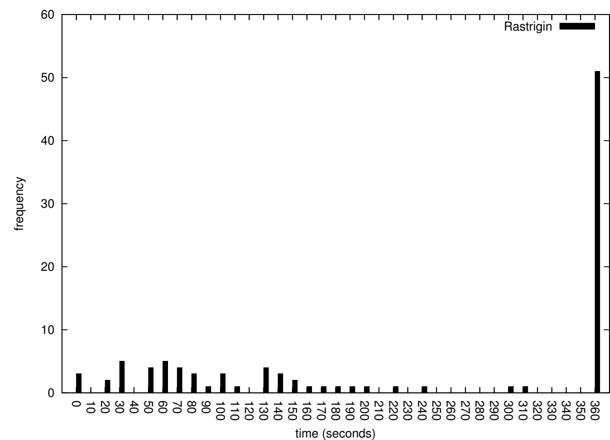

Fig. 3. Runtime distribution for Rastrigin function (time to reach target score of 10)

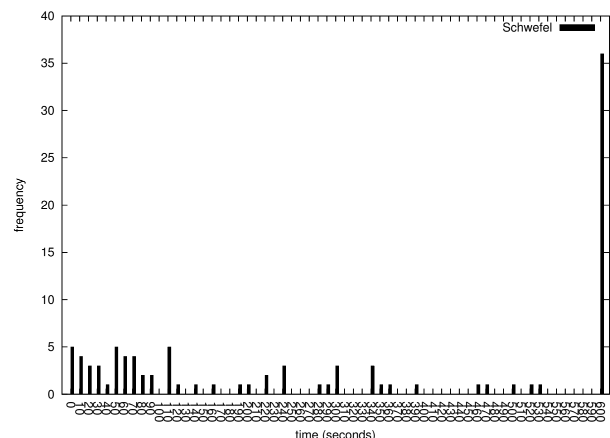

Fig. 4. Runtime distribution for Schwefel function (time to reach target score of 500)

IV. A Randomized, Heterogeneous Distributed GA

We have observed above that in a single-population, sequential GA, good parameter settings are not very difficult to find given enough samples. This suggests the following, simple variation on the island-model distributed GA described in Section II: Given a p-processor parallel system, initialize a population on each processor, where each population uses a set of randomly generated control parameters (as opposed to running p processes with identical parameter settings).

Even if the populations were executed completely independently (without migration), the expected performance of this simple strategy would be no worse than running a single-population, sequential GA p times, each with a different parameter configuration, and taking the best result. The results from Section III suggest that if p is large enough, we would sample a parameter configuration which is well suited for the particular problem and performs well.

Next, we consider the effect of migration in the DGA, which has the effect of propagating newly found, high-quality candidates among processors. Migration is expected to enhance the usefulness of parameter sets, which, on their own, lead to poor results. For example, a parameter set which is poorly suited for global optimization but well suited for local optimization (e.g., small population, low crossover rate, low mutation rate) are now periodically "seeded" with good start points for local optimization from their neighbors. Thus, a mixed ensemble of parameter sets which communicate via migration can perform better than an ensemble of completely independent populations.

In our experiments, the mutation and crossover rates were unconstrained and selected uniformly from $[0, 1.0]$, and the population parameter was constrained to be in $[10, 100]$. Although we generally wanted the control parameters to be as unconstrained as possible, we chose an upper bound for 100 for the population parameter because each processor was running a generational GA, and it seemed that if we allowed the population size to be too large, the frequency of migration would end up being too small. Note that migration rate is not a parameter because, as described in Section II, migration occurs when a new local elite is found at a processor. Large population sizes would not pose an issue regarding migration frequency if we used a steady-state GA instead of the current generational GA; this is an area for future work.

V. Experimental Results

To assess the effectiveness of the randomized, heterogeneous DGA, we compared it to a homogeneous DGA. A *homogeneous DGA* is a standard island-model DGA described in Section II, where all processors use identical GA control parameters.

As benchmark problems, we used the Griewangk, Rastrigin, and Schwefel functions, as well as the 14-input sorting network problem. The n-input sorting network problem is the problem of designing a circuit (network) with the minimal number of comparator elements, such that given any set of n numbers, the circuit sorts the inputs in order. This is a classical problem in theoretical computer science [9] which has been used as a benchmark in evolutionary computation. We use the same genetic representation for sorting networks as Graham, Masum and Oppacher [10]. The 14-input sorting network problem is difficult and time consuming, because evaluating a single candidate individual (sorting network) requires executing the network on 2^{14} test cases[1]. Thus, this is an example of a problem which can benefit significantly from parallel GAs. The fitness function for the sorting network problem counts the number of test cases which are sorted correctly, so this is a maximization problem where *higher* scores are better.

The experiments were conducted on a campus supercomputing cluster consisting of Sun Blade X6250 nodes where each node consists of two quad-core Xeon E5440(2.83GHz) processors (8 cores per node). We implemented the DGAs using MPI, assigning a single island deme to each core. Note that in our experiments, the computational nodes we used are fully allocated to the parallel GAs for the full duration of the GA runs (no other computationally intensive processes are running on the nodes other than the DGAs).

The goals of this study were:

1) Compare the *average* performance of a heterogeneous DGA against the expected performance of the *best* homogeneous DGA for each problem;
2) Compare the heterogeneous DGA against the average performance obtainable by a homogeneous DGA which used randomly selected parameter settings.

Figures 5-12 show the best-so-far solution curves for the heterogeneous DGA and homogeneous DGA on the four benchmark problems. Each graph consists of three curves, which were obtained as follows:

The heterogeneous DGA was executed 20 times on each problem, and are represented by the `heterogeneous` lines in Figures 5-12.

In order to identify (an approximation of) the best homogeneous DGA configuration for each benchmark, we first executed 200 independent runs of the homogeneous DGA, where each run used a different set of randomly generated control parameters (within each run, the homogeneous DGA uses the same set of control parameters on all processors). The parameter values were selected uniformly from the same ranges used by the heterogeneous DGA. The runs were executed with a problem-dependent time limit. From this data, we identified the set of control parameters which yielded the best fitness value in these 200 runs, and called this the *best homogeneous* parameter set.

This procedure was repeated separately for each of the 4 benchmark problems, so for each benchmark, the *best homogeneous* parameters are different, and represent a set of control parameters which are highly tuned for that problem

[1] Although there are A^N sequences of length N using an alphabet of size A, the 0-1 Principle guarantees that correctly sorting all 2^n sequences consisting of 0s and 1s is sufficient to prove that a sorting network will correctly sort any sequence of length N [9].

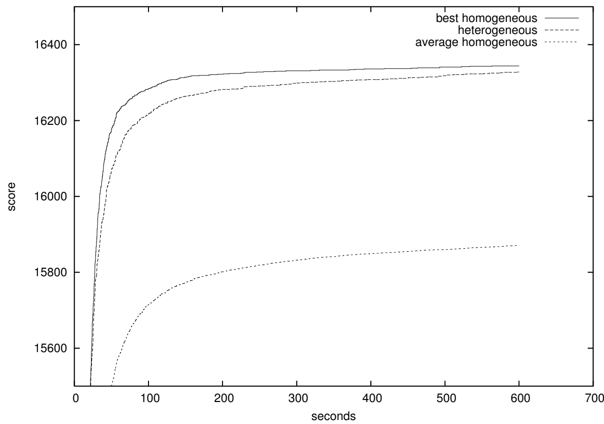

Fig. 5. 14-Input sorting network with 16 processors

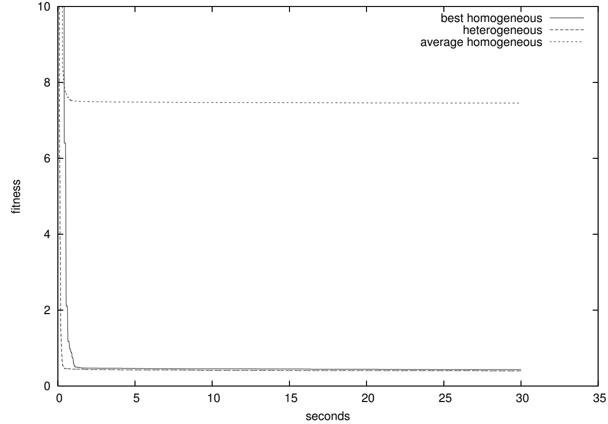

Fig. 6. Griewangk function with 16 processors

(i.e., sampling 200 parameter sets for each problem can be considered a tuning process).

For each of the 4 benchmark problems, we configured the homogeneous DGA to use the best homogeneous parameters on all processors, and executed this configuration 20 times with different random seeds. The average results of these runs is represented by the `best homogeneous` lines in Figures 5-12.

In order to evaluate the average performance obtainable with a homogeneous DGA, we show the average performance of 200 homogeneous DGA runs on each problem, where every run uses a different set of control parameter settings. These are represented by the `average homogeneous` lines in Figures 5-12.

Finally, in order to observe the scaling of performance as the number of processors was varied, so this evaluation of the heterogeneous and homogeneous DGAs was repeated for 16 and 100 processors Figures 5,6,8 and 7 compare the heterogeneous DGA and homogeneous DGA executed on a 4x4 toroidal grid (16 processors). Figures 9,11, 8 and 12 compare the heterogeneous DGA and homogeneous DGA on a 10x10 toroidal grid (100 processors).

It is important to note that since the entire process was repeated for 16 and 100 processors, the `best homogeneous` line in each graph represent the best homogeneous DGA not only for that problem, but for that specific number of processors. Similarly, the randomized heterogeneous configuration is different for each problem and set of processors since different random seeds were used on all runs.

We observed the following from the data:

- Both the heterogeneous and best homogeneous configurations significantly outperform the average homogeneous DGA run.
- With the exception of the Schwefel function on 16 processors, the performance of the heterogeneous DGA is competitive with the homogeneous DGA using the best homogeneous control parameter set.
- As the number of processors is increased from 16 to 100, the relative performance of the heterogeneous DGA

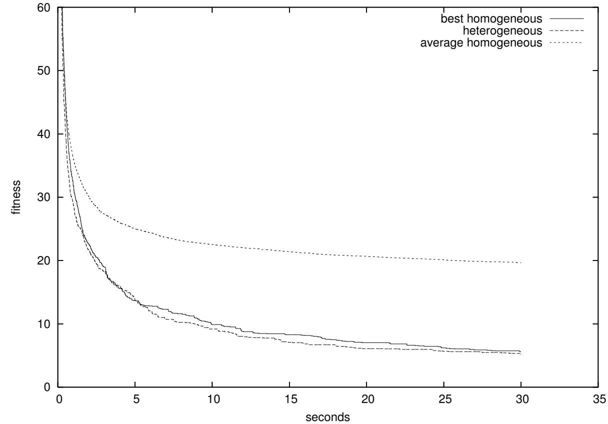

Fig. 7. Rastrigin function with 16 processors

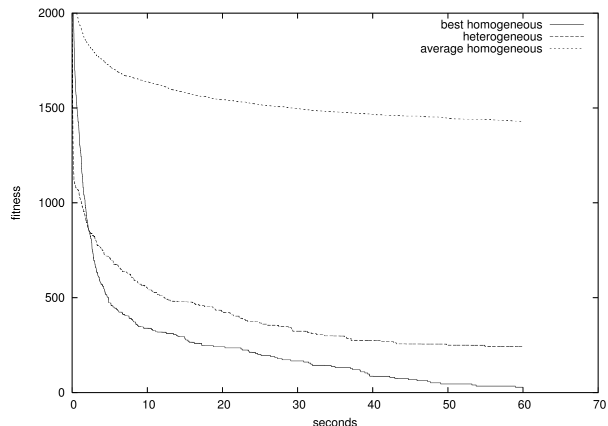

Fig. 8. Schwefel function with 16 processors

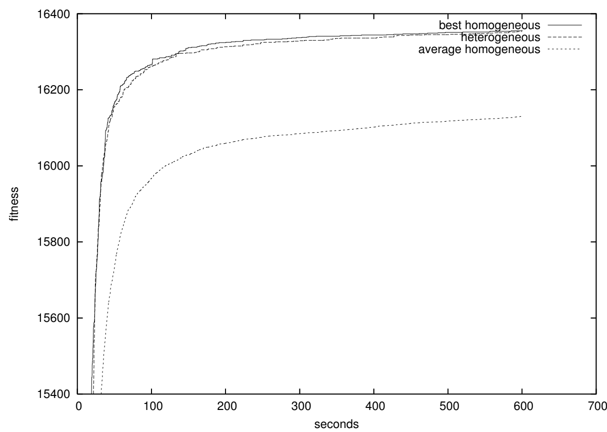

Fig. 9. 14-Input sorting network with 100 processors

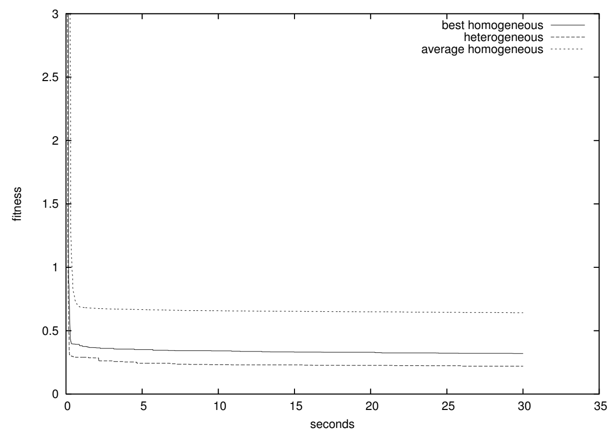

Fig. 11. Griewangk function with 100 processors

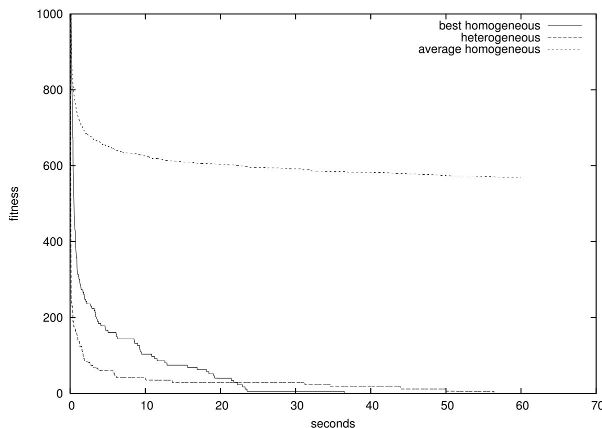

Fig. 10. Schwefel function with 100 processors

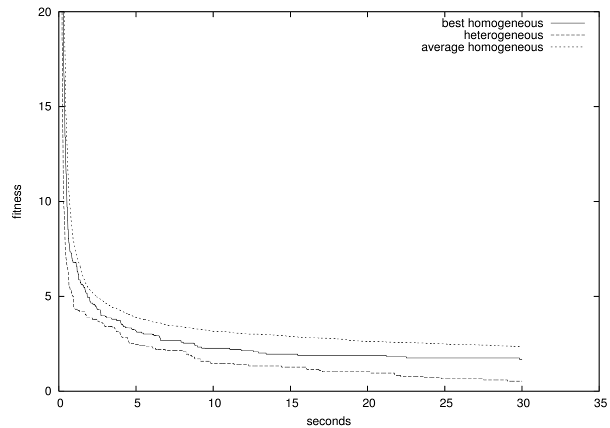

Fig. 12. Rastrigin function with 100 processors

improves.

Thus, the data shows that the expected performance of a heterogeneous DGA is significantly better than that of an average homogeneous DGA run. In other words, using a heterogeneous DGA successfully eliminates the risk of choosing an arbitrary (random) set of control parameters for a given problem.

Furthermore, the performance of the DGA is quite competitive with that of the best (highly tuned) homogeneous DGA setting for each problem. Somewhat surprisingly, the heterogeneous DGA actually outperforms the best homogeneous DGA in several cases (Griewangk and Schwefel functions with 100 processors). This indicates that using a heterogeneous DGA (which did not involve any parameter tuning) is competitive with parameter tuning.

The improvement in results as the number of processors was increased from 16 processors to 100 processors indicates that while 16 random configurations appears to be sufficient for the heterogeneous DGA to perform competitively some problems (Rastrigin and Griewangk), increasing the number of random configurations available does indeed lead to particularly good configurations being discovered, as suggested by the runtime distribution results in Section III.

VI. Related Work

In essence, a heterogeneous DGA is a strategy which seeks to minimize the risk associated with committing to a single, poor set of parameter values for the entire cluster by relying on a diversity of parameter settings. The rational allocation of resources among a set of algorithm is known as an algorithm portfolio [11], [12], which has been previously applied to sequential genetic algorithms [8]. The differences between a heterogeneous DGA and an algorithm portfolio are that (1) in an algorithm portfolio, computational resources are allocated among candidate algorithms based on the components of an algorithm portfolio according to statistical performance profiles of the algorithms on other problem instances, and (2) the components of an algorithm portfolio are executed completely independently and do not communicate. In contrast, the processes (populations) in a heterogeneous DGA, communicate via migration. The practice of running multiple algorithmic configurations in parallel has been shown to be effective for heuristic search algorithms, and has been called dovetailing [13]. In dovetailing, the parallel algorithms run independently and do not cooperate by exchanging candidate solutions or partial solutions.

Our heterogeneous DGA is similar to recent work on parallel hyper-heuristics by Bianzzini et al[14]. In a parallel hyper-heuristic, each island (processor) executes an iteration of some heuristic, and passes the result of executing heuristics among the islands. Our work differs in several respects. Bianzzini et al [14] use a set of 8 heuristics (6 different configurations of differential evolution, a particle swarm optimization, and a random search algorithm). All of the population-based heuristics (DE and PSO) use the same population size If there are more than 8 processors, some processors will be executing duplicate heuristics – this duplication is not a problem because hyper-heuristics operate by rapidly passing the output of each heuristic as the input of another heuristic. They investigate various strategies for dynamically reassigning the 8 candidate heuristics to each processor. Similarly, Leon et al apply a dynamically reconfiguring, parallel hyper-heuristic to 2D packing [15].

In some sense, our heterogeneous DGA is at the opposite end of the spectrum as the work on parallel hyperheuristics [14], [15]. Our focus is on an extremely simple, static, randomized strategy. We statically assign a randomly generated GA configuration to each processor, so there is greater diversity in the algorithmic configurations that are used (compared to [14], [15]), but the configurations are not changed during the run. Communications (migration) in our heterogeneous DGA is infrequent, and occurs only when a local elite is updated. Instead of dynamic reconfiguration, we have focused on how well a simple, *static*, random assignment of configurations to processors can perform compared to a tuned, standard homogeneous DGA. An empirical comparison of these contrasting approaches is an interesting avenue for future research.

The parameter-less GA is an approach to eliminating parameter tuning in GAs [16]. In this approach, Harik and Lobo first "eliminate" some parameters by arguing (based on schema theory) that selection rate and crossover rate should be set to a constant setting for all problems, and turning off mutation completely. The remaining, single parameter is population size. They execute a race among multiple populations of various sizes. Smaller populations are killed and replaced by larger populations when it no longer seems worthwhile to continue running the small population. Although this work was implemented sequentially, the racing populations could be implemented in parallel. The heterogeneous DGA can be seen as a different type of "parameter-less", parallel GA. While Harik and Lobos fixed selection rate and crossover rates at particular values, we simply use a range of reasonable parameters and randomly sample from this range.

Berntsson proposes an adaptive parallel approach where multiple island-model DGAs are run simultaneously (in parallel) [17]. That is, at any time, there are multiple island model DGAs executing in parallel, each of the island DGAs is independent, and each island DGA has a different number of islands, different population size. Unsuccessful DGAs are killed and restarted with different parameters. This approach focuses on the number of islands and island sizes (populations), and uses identical mutation rates and crossover rates across all islands of all of the DGAs.

Numerous approaches to self-adaptive genetic algorithms have been proposed. Many of these approaches, are surveyed in [2]. Although much of the work on self-adaptive strategies has been for sequential, evolutionary algorithms they can be generalized to parallel implementations; some researchers have worked on adaptive methods have been developed island-model evolutionary algorithms (c.f., [18]). However, most of these adaptive approaches involve some meta-level control parameters which control the on-line adaptation, so they are not parameter-free, and it is not yet clear how well any of these techniques will perform for an arbitrary problem. In contrast, our heterogeneous DGA approach is motivated by risk-aversion, and is specifically intended for cases where prior knowledge is unavailable. We do not try to actively automatically tune the system, but take a completely passive approach: Based on our observations of runtime distributions in Section III, we rely on random assignment of parameters to assign good control parameter settings to *some* of the processors.

VII. Discussion and Directions for Future Work

This paper explores the use of an extremely simple, randomized strategy for setting control parameters in a distributed genetic algorithms, where a different, random set of control parameter values (mutation rate, crossover rate, population) are used on every processor in an island-model distributed GA. Our results show that with a sufficient number of processors, this naive, heterogeneous island-model distributed GA strategy performs comparably to the results of a tuned, standard island-model GA (the best homogeneous configuration). This suggests that in applications where time and resources are limited, our simple method can be a viable method for configuring a distributed GA without parameter tuning. This is attractive in situations where there is no time available for tuning/experimentation, or where tuning experiments would incur monetary costs (e.g., cloud computing).

Heterogeneous distributed GAs exploit the fact that with a sufficient number of processors, it is likely that at least some of the processors will end up being assigned a set of random control parameters which performs particularly well on a given problem. Therefore, the approach does not involve any meta-level learning or automated online parameter tuning. While numerous approaches to automated GA parameter configuration have been previously proposed, our naive method is attractive because of its simplicity. Although it is quite possible that a more sophisticated automated configuration methods could be adapted to perform well for a distributed GA, we believe that due to its extreme simplicity, the heterogeneous DGA proposed here can be considered a new *baseline* for evaluating more sophisticated approaches.

While these results appear promising, there are several directions for future work. For example, the impact of communication (migration) policies on the effectiveness of the heterogeneous DGA needs to be clarified. Another direction for future work is to investigate the impact of constraining

and biasing the randomly generated parameter values. In our experiments, there were relatively few parameters (population size, mutation rate, crossover rate), and we tried to allow the heterogeneous DGA to sample from a broad range of values. It is possible that different sets of constraints and biases could significantly affect the performance of a heterogeneous DGA. It is also possible that using a more sophisticated GA at each node which uses a larger number of control parameters could affect how a heterogeneous DGA performs relative to a homogeneous DGA.

ACKNOWLEDGMENTS

This research was supported by the JSPS CompView GCOE, the Okawa Foundation, and a JSPS Grant-in-Aid for Young Scientists.

REFERENCES

[1] E. Cant'u-Paz, "Parameter setting in parallel genetic algorithms," in *Parameter setting in evolutionary algorithms*, ser. Studies in Computational Intelligence, L. F, L. C, and M. Z, Eds. Springer, 2007, pp. 259–276.

[2] S. Meyer-Nieberg and H.-G. Beyer, "Self-adaptation in evolutionary algorithms," in *Parameter setting in evolutionary algorithms*, ser. Studies in Computational Intelligence, L. F, L. C, and M. Z, Eds. Springer, 2007, pp. 121–142.

[3] E. Cant'u-Paz, "Migration policies, selection pressure, and parallel evolutionary algorithms," *Journal of Heuristics*, vol. 7, no. 4, pp. 311–334, 2001.

[4] L. Araujo, J. J. M. Guervós, A. Mora, and C. Cotta, "Genotypic differences and migration policies in an island model," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2009, pp. 1331–1338.

[5] D. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[6] C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz, "Heavy-tailed phenomena in satisfiability and constraint satisfaction problems," *J. Autom. Reasoning*, vol. 24, no. 1/2, pp. 67–100, 2000.

[7] A. S. Fukunaga, "Restart scheduling for genetic algorithms," in *Proc. Parallel Problem Solving from Nature (PPSN V, Lecture Notes in Computer Science 1498)*, 1998, pp. 357–366.

[8] A. Fukunaga, "Genetic algorithm portfolios," in *Proc. IEEE Congress on Evolutionary Computation*, 2000.

[9] D. Knuth, *The Art of Computer Programming*, 2nd ed. Addison Wesley, 1998, vol. 3.

[10] L. Graham, H. Masum, and F. Oppacher, "Statistical analysis of heuristics for evolving sorting networks," in *Prceedings of GECCO*, 2005, pp. 1265–1270.

[11] B. Huberman, R. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 275, no. 5296, pp. 51–54, 1997.

[12] C. P. Gomes and B. Selman, "Algorithm portfolios," *Artifial Intelligence*, vol. 126, no. 1-2, pp. 43–62, 2001.

[13] R. A. Valenzano, N. R. Sturtevant, J. Schaeffer, K. Buro, and A. Kishimoto, "Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms," in *ICAPS*, 2010, pp. 177–184.

[14] M. Biazzini, B. Bánhelyi, A. Montresor, and M. Jelasity, "Distributed hyper-heuristics for real parameter optimization," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2009, pp. 1339–1346.

[15] C. León, G. Miranda, and C. Segura, "A memetic algorithm and a parallel hyperheuristic island-based model for a 2D packing problem," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2009, pp. 1371–1378.

[16] G. R. Harik and F. G. Lobo, "A parameter-less genetic algorithm," in *Genetic and Evolutionary Computation Conference (GECCO)*, 1999, pp. 258–265.

[17] J. Berntsson, "G2DGA: an adaptive framework for internet-based distributed genetic algorithms," in *Proceedings of the GECCO 2005 workshops on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 346–349.

[18] J. Tang, M.-H. Lim, and Y.-S. Ong, "Adaptation for parallel memetic algorithm based on population entropy," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2006, pp. 575–582.