In E. Deaton et al., eds., Applied Computing 1994: Proceedings of the 1994 Symposium on Applied Computing, New York : The Association for Computing Machinery, 1994.

AN INITIAL ANALYSIS OF DATA PARALLELISM IN THE FAST MESSY GENETIC ALGORITHM

Laurence D. Merkle and Gary B. Lamont
Air Force Institute of Technology

Keywords

Data Parallelism, Optimization, Genetic Algorithms

Abstract

Genetic algorithms (GAs) are highly parallelizable algorithms, inspired by Darwinian theories of evolution and survival of the fittest, which are used most frequently as function optimizers. The messy GA [4, 5, 6] makes use of partially enumerative initialization (PEI) and tournament selection, with competitions restricted to similar individuals. The fast messy GA replaces PEI with Probabilistically Complete Initialization and building block filtering, and increases the threshold for similarity determinations [7]. Two algorithmic design approaches to parallelization of the fast messy GA are presented. One uses independent subpopulations throughout the execution, while the other uses recombines subpopulations following the primordial phase. Solution quality and execution time are examined theoretically and experimentally against a fully deceptive function. Experiments are performed on an Intel Paragon parallel supercomputer.

Introduction

The genetic algorithm (GA) has its inspiration in processes of evolution and natural selection [9]. Substantial empirical data exists which suggests that the GA is well suited for optimization of some classes of functions (e.g. see DeJong [2]). One class for which the GA is known to be less well suited is that of *deceptive* functions [14]. The *messy GA* is designed specifically to address the limitations of the simple GA associated with functions of bounded deception [4, 5, 6].

The practical application of the messy GA is limited by its very large initial population size and corresponding long execution times. In order to overcome these limitations, Goldberg, et. al. [7] propose several modifications to the original messy GA. The modified algorithm is referred to as the fast messy GA.

This paper examines two parallel fast messy GA designs, both of which exploit the data parallelism present in the fast messy GA. The designs are presented and examined theoretically, and their performance against a difficult test problem is compared experimentally.

*Sponsored by the Air Force Office of Scientific Research and the Air Force Materiel Command's Electronic Systems Center

Messy Genetic Algorithms

Motivation for the messy GA arises from viewing the objective (or *fitness*) function $f(x)$ as the sum of m independent subfunctions $f_i(x_i)$, each defined on the same number of loci k, where k is the estimated level of deception present in the most deceptive subfunction. A *building block* is a set of genes which fully specify the independent variable of some subfunction. We define a *highly fit building block* (HFBB) to be the building block (assumed unique) which optimizes the corresponding subfunction. Thus the (unique) string containing only HFBB's optimizes the objective function.

The messy GA consists of an *initialization*, a *primordial*, and a *juxtapositional* phase. Its improved ability to solve deceptive problems stems from the focus on increasing the proportion of HFBB's in the population before applying recombinative operators. As originally proposed, this is accomplished throught the use of *partially enumerative initialization* (PEI) and tournament selection.

Original Messy GA

PEI results in a population consisting of all possible partial solutions defined over k loci. Thus, each building block is

represented exactly once, although not every string contains a building block, since the loci over which a string is defined may correspond to different subfunctions. For an application in which each string contains ℓ genes, and each gene has C possible alleles, the initial population contains

$$N = k^C \left(\begin{array}{c} \ell \\ k \end{array} \right) \tag{1}$$

solutions. For even modest values of k this is significantly larger than a typical simple GA population size. For example, for a problem using a binary representation with $\ell = 50$ and $k = 5$, the initial population contains 6.78×10^7 individuals. Typical simple GA population sizes are in the tens to thousands.

Tournament selection is then used in the primordial phase to reduce the population size by eliminating less fit partial solutions. Competition is limited to those partial solutions for which the number of common defining loci is greater than the expected value. The *shuffle size* parameter specifies the maximum number of individuals examined in searching for a compatible mate. A locally optimal solution, called the *competitive template*, is used to "fill in the gaps" in partially specified solutions to allow their evaluation and subsequent selection.

The juxtapositional phase is similar to the simple GA in its use of recombinative operators. Standard crossover is replaced by *cut-and-splice*, which is a one-point crossover operating on variable length strings. Splice and bitwise cut probabilities are specified, and are normally chosen to promote rapid string growth from k to ℓ [4].

Fast Messy GA

In order to reduce the size of the messy GA initial population and the execution time of the initialization and primordial phases, Goldberg, et. al. propose three modifications to the original algorithm: use of Probabilistically Complete Initialization (PCI) in place of PEI, use of building block filtering, and more conservative thresholding in tournament selection [7].

The objective of PCI is to ensure that each HFBB has an expected number of copies in the initial population sufficient to overcome sampling noise. Each individual in the PCI population is defined at $\ell' = \ell - k$ loci, which are selected randomly without replacement (it is assumed that $k \ll \ell$). After accounting for noise, the required population size is

$$N = \frac{\left(\begin{array}{c} \ell \\ \ell' \end{array} \right)}{\left(\begin{array}{c} \ell - k \\ \ell' - k \end{array} \right)} 2 z_\alpha^2 \beta^2 (m-1) 2^k \tag{2}$$

where ℓ, ℓ', k, and m have been defined previously, α is a parameter specifying the probability of selection error between two competing building blocks, $P[Z \geq z_\alpha] = 1 - \alpha$ where Z is a standard normal random variable, and β^2 is a parameter specifying the maximum inverse signal-to-noise ratio per subfunction to be detected.

The fast messy GA uses tournament selection and building block filtering (BBF) to enrich the population in the primordial phase. The effect of several iterations of tournament selection is to eliminate nearly all of the individuals in the population which contain fewer HFBB's, while increasing the number of copies of individuals which contain more. BBF then reduces the number of defining loci for each individual by randomly deleting some number of genes. In the process it disrupts many but not all of the HFBB's. Some individuals containing HFBB's remain to receive additional copies in subsequent iterations of tournament selection.

For any particular problem, some HFBB's may contribute more to the total fitness of the optimal solution than others. Also, a small number of individuals may contain multiple HFBB's. Since all of the HFBB's must be represented in the population if there is to be any chance of combining them, competition is restricted to those individuals which contain building blocks corresponding to the same subfunction. Theoretically, this may be achieved by allowing competition only between those individuals which share at least

$$\theta = \lceil \frac{\lambda^2}{\ell} + z_{\alpha'}^2 \sigma \rceil \tag{3}$$

common defining loci, where λ is the number of defining loci for the individuals, ℓ and z are as defined above, and

$$\sigma^2 = \frac{\lambda^2 (\ell - \lambda)^2}{\ell^2 (\ell - 1)} \tag{4}$$

is the variance of the distribution of a random variable L corresponding to the number of defining loci shared by two randomly selected individuals. Thus, the θ is calculated using a normal distribution approximating the actual distribution of L in the initial population. The approximation ignores the dependence of L's distribution on the dynamics of the genetic population in subsequent generations.

By carefully choosing a primordial schedule of tournament selection and BBF, a population can be generated which consists of strings of length k and which is dominated by HFBB's. In current practice, it is common to use a full shuffle and an empirically determined threshold schedule [10].

Exploitation of Data Parallelism

A common approach to implementation of GAs on coarse grained parallel architectures is the *island model* [8]. In this approach each processor executes a separate GA on a subpopulation. Numerous variations exist in which either the selection operation executing on a particular processor is affected by other processors' subpopulations, or processors communicate some portion of their subpopulations to other processors. These approaches extend with some modifications to the messy GA [12]. This paper examines the solution quality and execution time obtained using two designs of the fast

messy GA based on the island model. The designs are chosen for their simplicity, and do not consider alternatives such as non-homogenous subpopulations [12] and migration [11].

The first design, called the *fully parallel* design, uses independent subpopulations throughout the execution. One processor (the *controller*), in addition to executing the fast messy GA, reads the input parameters and broadcasts them to the remaining processors. From Equation 2, each subpopulation contains

$$N = \left\lfloor \frac{1}{P} \frac{\binom{\ell}{\ell'}}{\binom{\ell-k}{\ell'-k}} 2 z_\alpha^2 \beta^2 (m-1) 2^k \right\rfloor \qquad (5)$$

individuals, where P is the number of processors. Following completion of the juxtapositional phase, a brief serial phase occurs in which each processor communicates its best individual to the controller, which then reports the overall best individual and its fitness.

The second design, called the *partially parallel* design, also uses independent subpopulations in the primordial phase. Prior to the juxtapositional phase, the controller processor receives each of the other processors' subpopulations and combines them in a single population. The controller then carries out the juxtapositional phase serially. After the juxtapositional phase, the controller reports its best individual and its fitness.

Solution Quality

We are interested in the distribution of the random variable B_g, the number of HFBB's in an arbitrary individual in generation g. We begin by deriving the distribution of B_0, the number of HFBB's in an arbitrary initial string. The probability that an initial string contains i HFBB's is given by

$$P[B_0 = i] = \sum_{j=i}^{m-1} P[F = j] P[B_0 = i | F = j] \qquad (6)$$

where F is the number of fully specified building blocks in the string. The distribution of F involves an occupancy problem with k indistinguishable objects and m indistinguishable cells, the solution of which involves considering all distinct partitions of k into exactly m parts [13]. It also involves the number of ways to choose the unspecified bits within a chosen building block. For example, for $\ell = 50$ and $k = 5$, the number of loci combinations giving 7 fully specified blocks is

$$\begin{aligned} &\binom{5}{1}\binom{5}{1}\binom{5}{3} \times \binom{10}{2}\binom{8}{1} \quad + \\ &\binom{5}{1}\binom{5}{2}\binom{5}{2} \times \binom{10}{1}\binom{9}{2} \quad = 270,000. \end{aligned}$$

Since there are 2,118,760 total loci combinations, $P[F = 7] = 0.1274$. Now the probability that i of the j fully specified building blocks in a string are HFBB's is

$$P[B_g = i | F = j] = \sum_{j=m-k}^{m-1} \binom{j}{i} (2^k - 1)^{j-i} (2^k)^{m-j-1}. \qquad (7)$$

We note that this probability is indepedent of the number of processors and is the same for both designs. The distribution of B_0 for $\ell = 50$ and $k = 5$ is shown at Table 1.

Table 1: Distributions of B_g

i	B_0	Threshold
0	0.8329	39
1	0.1545	35
2	0.0120	28
3	0.0005	23
4	¡.0001	18
5	¡.0001	15
6	¡.0001	13
7	¡.0001	10
8	¡.0001	9
9	¡.0001	7

Next we consider the effects of tournament selection on B_g. There are three mutally exclusive events E_1, E_2, and E_3 in which a string containing exactly i HFBB's is included in generation g, so that

$$P[B_g = i] = P[E_1] + P[E_2] + P[E_3]. \qquad (8)$$

E_1 is the event that such a string is selected as the first mate for a tournament, and no compatible second mate is found, for which

$$P[E_1] = P[B_{g-1} = i] P[I] \qquad (9)$$

where I is the event that a randomly chosen string is in incompatible with every other string in the population. Since the number of common defining loci of randomly selected strings has a hypergeometric distribution [1], we have

$$P[I] = \left(\sum_{x=\theta}^{\lambda} \frac{\binom{\lambda}{x}\binom{l}{\lambda - x}}{\binom{l}{\lambda}} \right)^{N-1}. \qquad (10)$$

E_2 is the event that such a string is selected as the first mate, and that the second mate has no more than i HBFF's (we assume that the mate with more HBFF's always wins, and that the first mate wins in case of a tie). We write

$$P[E_2] = P[B_{g-1} = i](1 - P[I]) \sum_{j=0}^{i} P[B_{g-1} = j]. \qquad (11)$$

Finally, E_3 is the event that such a string is chosen as the second mate, when the first mate contains fewer than i HFBB's, so

$$P[E_3] = \sum_{j=0}^{i-1} P[B_{g-1} = i](1 - P[I]) P[B_{g-1} = j]. \qquad (12)$$

The probability of the event that BBF results in a string with i HFBB's is

$$P[B_g = i] = \sum_{j=0}^{g-i} P[B_{g-1} = i + j]P[D_{i+j}^{j}(\lambda_0, \lambda_1)] \qquad (13)$$

where $D_i^j(\lambda_0, \lambda_1)$ is the event that BBF disrupts j HFBB's in reducing the string length from λ_0 to λ_1. Accounting for the event that fewer than j HFBB's are disrupted, we have the recursion relation

$$P[D_i^j(\lambda_0, \lambda_1)] = \frac{\begin{pmatrix} \lambda_0 - (i-j)k \\ \lambda_0 - \lambda_1 \end{pmatrix}}{\begin{pmatrix} \lambda_0 \\ \lambda_0 - \lambda_1 \end{pmatrix}} - \sum_{k=0}^{j-1} P[D_i^k(\lambda_0, \lambda_1)] \qquad (14)$$

We note that the distribution of B_g at the end of the primordial phase is identical for both designs. This does not imply that overall solution quality must be identical. In order for the cut-and-splice operator to result in the formation of an individual containing two HFBB's, both HFBB's must be present in the population. Since each subpopulation of the fully parallel design initially contains fewer HFBB's, it is expected that the same is true of the juxtapositional subpopulations for this design. Thus, the solution quality of the fully parallel design is expected to decrease with an increasing number of processors. In contrast, since the number of HFBB's in the combined juxtapositional populations is expected to achieve a maximum for some P, the solution quality of the partially parallel design is expected to achieve a maximum also.

Execution Time

The execution time of both designs of the fast messy GA depends on the execution time of each of the three phases, and the time required for communication. The initialization time is determined by the subpopulation size, and is $O(P^{-1})$.

The primordial phase execution time is a function of the subpopulation size, the shuffle size, and the probability of compatibility for individuals randomly selected from the same subpopulation [11]. Since the shuffle size is chosen to be equal to the subpopulation size, both are inversely proportional to P. The probability of compatiblity of randomly selected solutions does not depend on P in any obvious way. Thus, primordial phase execution time is expected to be $O(P^{-2})$.

Similarly, the juxtapositional phase execution time is a function of subpopulation size, shuffle size, and probability of compatibility. For the fully parallel design, execution time is again expected to be $O(P^{-1})$. For the partially parallel design, juxtapositional phase execution time is expected to be independent of P.

Communication time for the fully parallel design is expected to be $O(P)$ but negligible for $P \ll n$, since only the GA parameters and a single individual from each population is communicated. For the partially parallel design, all individuals except those in controller processor's subpopulation are communicated. Communication time is expected to increase asymtotically with increasing P.

Thus, for reasonable problem sizes, total execution time for the both designs is expected to decrease as $O(P^{-1})$ for $P \ll n$ and increase as $O(P)$ otherwise. For $P > 1$, execution time of the fully parallel design is expected to be less than that of the partially parallel design.

Experimental Results

In order to experimentally determine the effects of each of the designs on solution quality and execution time, experiments are performed using a substantially updated version of the parallel messy GA (PMGA)[3]. The PMGA is part of AFIT's GA Toolkit, which includes several sequential and parallel GAs [3, 11]. Both designs of the parallel fast messy GA are implemented on an Intel Paragon parallel supercomputer in C under the Paragon OSF/1 Operating System Release 1.1 operating system.

For each design, the problem solved is the 50 bit deceptive problem addressed by Goldberg, et. al. [7]. The problem consists of ten 5-bit subproblems, each of which is an order-5 fully deceptive trap function. The total fitness of a solution to the full problem is the sum of the fitnesses of the solutions to the subproblems. The encoding scheme for the function is based on a string of fifty genes and a binary genic alphabet. The bits corresponding to any given subproblem occur in consecutive genes.

The Paragon allocates processors in partitions of any size from 1 to a configuration dependent maximum. For these experiments, partition sizes of 1, 2, 5, 10, 20, 40, 60, and 80 are used. Each implementation is executed 8 times for each of the eight partition sizes, using 8 randomly generated seeds. The same seeds are used for both designs and all partition sizes. The GA parameters are chosen to match those used by Goldberg, et. al. [7, 10]. A total of fourteen applications of BBF are performed, as shown in Table 2. The shuffle number is equal to the subpopulation size, the cut probability is 0.02, and the

Table 2: BBF and Threshold Schedule

Generation	String length	Threshold
0	45	39
7	39	35
11	34	28
15	29	23
19	25	18
23	22	15
29	19	13
35	16	10
41	14	9
47	12	7
53	10	6
59	8	5
65	7	4
71	6	3
77	5	4

splice probability is 1.0. The primordial phase has 81 generations and the juxtapositional has 8. The overflow factor is 1.6, and the total population size is 1786 (Kargupta actually reports a population size of 1785). No outer loop is performed, and the competitive template is forced to consist of all 0's.

The average solution quality obtained using each of the designs for each partition size is shown in Figure 1. Likewise, the mean

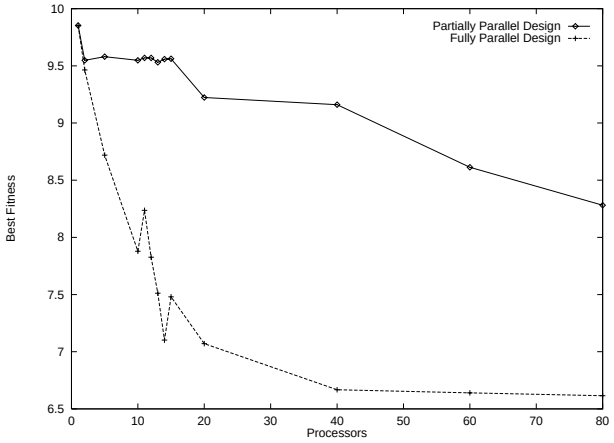

Figure 1: Mean Fitness

execution time for each of the desings for each partition size is shown in Figure 2.

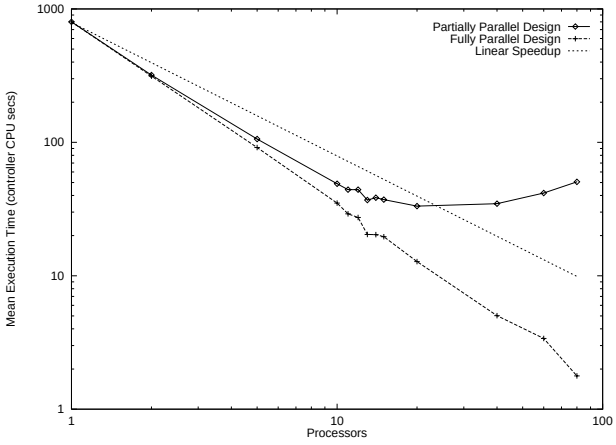

Figure 2: Overall Speedup

Conclusions

The fully parallel design of the fast messy genetic algorithm exhibits apparently super-linear speedup up to at least 80 processors for the test problem used in this paper. It does so at the cost of an asymtotic decrease in solution quality. The partially parallel design also exhibits apparently super-linear speedup, but does so while achieving a maximum in solution quality when the number of processors is near the number of HFBB's necessary to construct a complete solution.

Recommendations

The designs examined in this paper do not consider options such as non-homogenous initial subpopulations. It has been shown elsewhere [12] that the execution time of the parallel messy GA is strongly affected by the distribution of individuals in the initial subpopulations, and in particular by the probability of compatibility between solutions in those populations. Thus, a logical next step is to incorporate non-homogenous initialization strategies in the parallel fast messy GA.

Also, migration has been shown to have significant effects on both solution quality and execution time in the parallel simple GA. It is a natural enhancement of the parallel fast messy GA designs discussed in this paper to incorporate such migration methods in both the primordial and juxtapositional phases.

Finally, current understanding of the fast messy GA is insufficient to provide a methodology for selecting a primordial schedule of building block filtering and appropriate tournament selection thresholds and shuffle sizes. The fast messy GA will be substantially more applicable to real problems when a complete theory is available. Such a theory must provide a means for dynamically determining thresholds and shuffle sizes based upon the current distribution of the GA population.

Author Contact Information

Graduate School of Engineering, Department of Electrical and Computer Engineering, Wright-Patterson AFB OH 45433-7765 {lmerkle, lamont}@afit.af.mil

References

[1] Deb, Kalyonmoy. *Binary and Floating Point Optimization Using Messy Genetic Algorithms.* PhD dissertation, Department of Engineering Mechanics, The University of Alabama, Tuscaloosa, AL 35487-2908, April 1991.

[2] DeJong, Kenneth A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems..* PhD dissertation, The University of Michigan, Ann Arbor MI, 1975.

[3] Dymek, Capt Andrew. *An Examination of Hypercube Implementations of Genetic Algorithms.* MS thesis, AFIT/GCE/ENG/92-M, Air Force Institute of Technology School of Engineering, Wright-Patterson AFB OH, March 1992 (AD-A248092).

[4] Goldberg, David E. and others. "Messy Genetic Algorithms: Motivation, Analysis, and First Results," *Complex Systems*, *3*:493–530 (1989).

[5] Goldberg, David E. and others. "Messy Genetic Algorithms Revisited," *Complex Systems*, *4*:415–444 (1990).

[6] Goldberg, David E. and others. "Don't Worry, Be Messy." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 24–30. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1991.

[7] Goldberg, David E. and others. "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by Stephanie Forrest. 56–64. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1993.

[8] Gordon, V. Scott and Darell Whitley. "Serial and Parallel Genetic Algorithms as Function Optimizers." *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by Stephanie Forrest. 177–183. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1993.

[9] Holland, John H. *Adaptation in Natural and Artificial Systems* (First mit press Edition). Cambridge, MA: MIT Press, 1992.

[10] Kargupta, Hillol. "Personal communication." Tournament selection parameters in the fast messy GA, December 1993.

[11] Merkle, Laurence D. *Generalization and Parallelization of Messy Genetic Algorithms and Communication in Parallel Genetic Algorithms*. MS thesis, Air Force Institute of Technology, WPAFB OH 45433, December 1992.

[12] Merkle, Laurence D. and Gary B. Lamont. "Comparison of Parallel Messy Genetic Algorithm Data Distribution Strategies." *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by Stephanie Forrest. 191–198. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1993.

[13] Roberts, Fred S. *Applied Combinatorics*. Englewood Cliffs, NJ: Prentice Hall, 1984.

[14] Whitley, Darrell. "Fundamental Principles of Deception in Genetic Search." *Foundations of Genetic Algorithms*, edited by G. Rawlins. San Mateo, California: Morgan Kaufmann, 1991.