Evaluation-Time Bias in Asynchronous Evolutionary Algorithms

Eric O. Scott
Computer Science Department
George Mason University
Fairfax, Virginia USA
escott8@gmu.edu

Kenneth A. De Jong
Computer Science Department
George Mason University
Fairfax, Virginia USA
kdejong@gmu.edu

ABSTRACT

Parallelization of fitness evaluation is an established practice in evolutionary computation, and is a necessity in applications where fitness functions are computationally expensive. Traditional master-slave EAs based on a synchronous, generational model incur idle time when there is variance in the time it takes for individuals to have their fitness evaluated. Asynchronous evolutionary algorithms based on a steady-state model can make more efficient use of parallelization by eliminating idle time and reclaiming CPU resources. It is believed, however, that asynchronous EAs are biased toward regions of the search space where solutions take less time to evaluate, and away from regions where fitnesses evaluation is expensive. We show experimentally that asynchronous EAs do indeed exhibit an evaluation-time bias. This bias can either cause or prevent premature convergence. We also show, however, that on a flat fitness landscape, the asynchronous EA is attracted to both fast and slow regions of the search space, and away from medium-speed solutions. This indicates that further work is needed to understand the implications that asynchrony has for EA applications.

Categories and Subject Descriptors

I.2 [**Computing Methdologies**]: ARTIFICIAL INTELLIGENCE—*Problem Solving, Control Methods, and Search*

Keywords

Evolutionary Algorithms; Parallel Algorithms; Asynchronous Algorithms

1. INTRODUCTION

Asynchronous master-slave EAs are a class of parallel evolutionary algorithms that are based on a $(\mu+1)$-style steady-state model of evolution. The chief practical advantage of asynchronous EAs is that in applications where the computational cost of running an EA is dominated by fitness evaluation, asynchronous EAs can leverage an unbounded number of processors for fitness evaluation with a nearly 100% utilization rate. By contrast, traditional master-slave EAs, which are based on a generational, (μ, λ)-style model, may leave a substantial fraction of CPU resources idle when there is variance in the time it takes to evaluate individuals [4, 7].

Asynchronous EAs are thus an attractive tool when fitness evaluation times are long and variable. These conditions are especially likely to occur when solving design or optimization problems that involve executing an expensive stochastic simulation – a class of applications that has been growing in popularity in recent years. Examples include tuning the parameters of a complex scientific model to minimize the error between its predictions and real-world data, and evolving the shape of a mechanical part to maximize the performance of a simulated engine (ex. [5]). Efficient parallelization is necessary in such applications, as a single fitness evaluation may take on the order of minutes or even hours to complete. Under the right conditions, asynchronous EAs can facilitate a significant reduction in the amount of time it takes to solve a problem. Churchhill et al., for example, apply an asynchronous EA to a multi-objective tool-sequence optimization problem for automatic milling machine simulations [3]. The asynchronous method solved the problem just as well as a competing generational EA, but required 30-50% less wall-clock time.

By eliminating idle CPU resources, using an asynchronous EA on these tasks ensures that we have maximized our parallelization efficiency in terms of the number of individuals we can evaluate per unit time. The algorithm's search behavior, however, differs from the generational EA in a poorly understood and problem-specific way. The distribution of individual evaluation times in the population, the relationship (if any) between evaluation time and fitness, and the number of slave processors available all affect the search trajectory of an asynchronous EA. It is not clear when this behavior may help or hinder the algorithms' ability to find good solutions quickly, and the complex dynamics of the asynchronous model make it difficult to answer such questions analytically.

In particular, practitioners often fear that the search trajectory of an asynchronous EA may be biased toward fast-evaluating regions of the search space and away from solutions that take a longer time to evaluate. Yagoubi et al. mention that an asynchronous EA was indeed slower in discovering solutions to a multi-objective problem that were located in a region of the search space that was artificially configured to have longer evaluation times than the rest of

GECCO '15, July 11 - 15, 2015, Madrid, Spain

 ISBN 978-1-4503-3488-4/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768482

the landscape [6]. In another experiment, they found that penalizing high-quality solutions by giving them long evaluation times actually improved performance by helping to prevent premature convergence on a test problem. However, we previously found no statistically significant evidence of a bias toward fast or slow regions in an experiment on a flat fitness landscape [4], indicating that under some conditions evaluation-time bias may be small or negligible.

This paper helps make sense of these results by investigating the question of evaluation-time bias more closely on artificial problems. We use a discrete-event simulation of an asynchronous evolutionary algorithm to show that asynchronous EAs do indeed have a systematic bias toward fast solutions and away from slow ones while solving an illustrative optimization problem. Additionally, however, we show that in the absence of a fitness gradient, the asynchronous EA is attracted to both fast-evaluating regions of the search space and slow regions of the search space, and moves away from medium-speed regions.

2. THE SIMPLE ASYNCHRONOUS EA

Master-slave parallelization is most commonly achieved by using a (μ, λ)-style EA, where the entire population of λ children have their fitnesses evaluated in parallel by a number of slave processors [2]. These algorithms then *synchronize* by waiting for all the fitness evaluations to complete before proceeding to the next generation.

When the time it takes to evaluate each individual is constant and much greater than any other EA overhead (such as reproductive operators or network communication), the synchronous master-slave model keeps all the available CPU resources completely utilized. Since the algorithm waits for every individual in the population to finish evaluating, however, all it takes is one unusually slow individual to cause a significant fraction of CPU resources to lay idle while waiting for the next generation. The precise fraction of resources that are left idle depends on the distribution of evaluation times in the population, which may change in a complex way as evolution proceeds [4].

Asynchronous master-slave EAs eliminate this source of idle time by using a steady-state evolutionary model. Individuals are integrated into the population one-at-a-time immediately after their fitness has been evaluated, and a new individual is generated to take its place on the freed up CPU resource, without waiting on anything else. When the cost of fitness evaluation dwarfs all other overhead, an asynchronous master-slave EA has a 100% utilization rate.

Algorithm 1 details the asynchronous evolutionary algorithm we study in this paper from the perspective of the master processor. The first loop initializes the population by sending randomly generated individuals to a free node for evaluation (`send`()). We then breed one individual from the population and send it off for evaluation to fill the free node (`breedOne`() represents both parent selection and reproductive operators). Evolution then proceeds in the `for` loop, which waits for an individual to complete evaluating and thus free up a slave processor (`nextEvaluatedIndividual`()). Each newly evaluated individual competes against an individual chosen by `selectOne`() for a place in the population. Finally, a new individual is generated and sent off to fill the free slave node, and the cycle continues. This description of the algorithm leaves a number of design decisions up to the practitioner.

Algorithm 1 The Simple Asynchronous EA

```
1: function AsynchronousEvolution(n, steps)
2:     P ← ∅
3:     while |P| < n do
4:         if ¬nodeAvailable() then
5:             wait()
6:         while nodeAvailable() do
7:             ind ← randomIndividual()
8:             send(ind)
9:         finishedInd ← nextEvaluatedIndividual()
10:        P ← P ∪ {finishedInd}
11:    newInd ← breedOne(P)
12:    send(newInd)
13:    for i ← 0 to steps do
14:        finishedInd ← nextEvaluatedIndividual()
15:        replaceInd ← selectOne(P)
16:        if betterThan(finishedInd, replaceInd) then
17:            P ← (P − replaceInd) ∪ finishedInd
18:        newInd ← breedOne(P)
19:        send(newInd)
```

We use tournament selection of size 2 to select parents in `breedOne`(), and we use random selection for survival selection (`selectOne`()).

Since we are interested in the case where fitness evaluation dwarfs other EA overhead, experiments with asynchronous EAs can be quite time consuming to run if we wish to obtain statistically significant results. To facilitate doing a large number of runs on artificial problems as well as complete control over the experimental conditions, we implemented Algorithm 1 via a discrete-event simulation, in which each individual is assigned an evaluation time from an evaluation-time function that is defined as part of the experiment. A priority queue is used to instantly jump to the next completed evaluation event when `nextEvaluatedIndividual()` is called.

The experiments in this paper are conducted on simple real-valued optimization problems. We represent solutions as vectors in $\mathbb{R}^n$, and breed single children by applying two-point crossover to the parents and discarding one of the results. We also apply Gaussian mutation with hard bounds and a standard deviation of 0.5, mutating each gene individually with probability $1/n$.

The reason that we anticipate an evaluation-time bias in the asynchronous EA is that, while an individual with a long evaluation time is being executed on one processor, the other processors might evaluate a large number of faster individuals. An example of this is shown in Figure 1, which shows the sequence of evaluation times for 100 steps of an asynchronous EA simulation as it solves a paraboloid function. The dashed lines emphasize the times that an individual with a particularly long evaluation begins and ends. Because the algorithm is operating with simulated evaluation times, only the relative differences in evaluation times matter, and we show time in arbitrary units. By the time the single long-evaluating individual completes, more than 50 individuals have completed evaluation and had a chance to compete for a place in the population. This would appear to put long-evaluating individuals at a disadvantage, since in some cases fast individuals have more opportunity to reproduce.

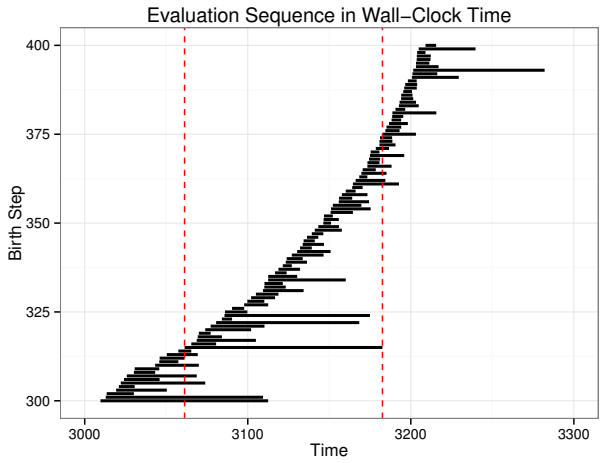

Figure 1: A sequence of fitness evaluation durations in an asynchronous EA. Many fast individuals may complete and enter the population in the time it takes for one slow individual to evaluate.

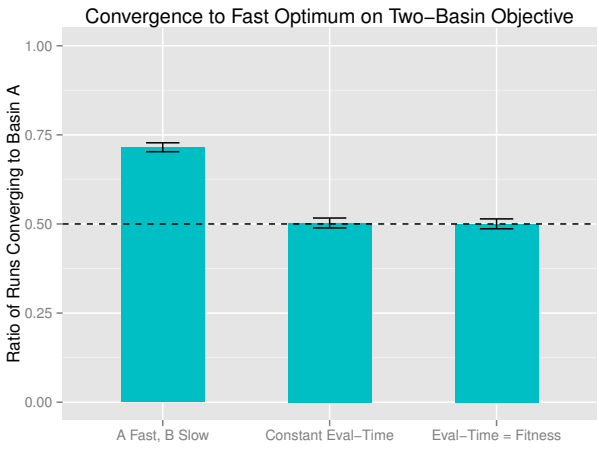

Figure 2: The fraction of runs that converge on optimum A_0 under three different evaluation-time conditions, supporting Hypothesis 1.

3. RESULTS

To demonstrate that asynchronous EAs are biased toward fast-evaluating individuals, we focus on a simple objective function that has two Gaussian basins of attraction centered on local minima A_0 and B_0:

$$\begin{aligned} f_{a,b}(\vec{x}) = &\max(|a|,|b|) \\ &- a\exp\left(-\frac{1}{2\sigma^2}\sum(x_i - 2\sigma)^2\right) \\ &- b\exp\left(-\frac{1}{2\sigma^2}\sum(x_i + 2\sigma)^2\right). \end{aligned} \quad (1)$$

We set $\sigma = 2.5$ and use bounds of $(-10, 10)$ when initializing and mutating each gene. In addition to the fitness function, let $t(\vec{x})$ denote the evaluation time of each individual $\vec{x} \in \mathbb{R}^n$. When the depth parameters a and b are equal, the two basins are identical from a fitness perspective. In this case, we anticipate the following:

Hypothesis 1a: On the fitness function $f_{a,b}$ with $a = b$, an asynchronous EA will converge to either optimum with equal probability if individual evaluation times are constant ($t(\vec{x}) = 1$) or directly proportional to fitness ($t(\vec{x}) = f_{a,b}(\vec{x})$).

Hypothesis 1b: If solutions in one basin have shorter evaluation-times than the other basin, however (namely, $t(\vec{x}) = f_{a,-b}(\vec{x})$), the EA will be biased toward the fast-evaluating basin.

For each of the three choices of $t(\vec{x})$ in Hypothesis 1, we ran $N = 5,000$ independent runs of the asynchronous EA out to 1,000 steps, long enough so that all N runs converged. The population size is 10 and we use 10 simulated slave processors. The result of each run can be represented by the random variable

$$R_i = \begin{cases} 1 & \text{if } \|\vec{x}' - A_0\| \leq \tau \\ 0 & \text{if } \|\vec{x}' - B_0\| \leq \tau \end{cases}, \quad (2)$$

where $\vec{x}'$ is the best individual discovered in the run, A_0 and B_0 are the locations of the two optima, and τ is a small number that serves as a convergence threshold. Now, the number of runs that converge to basin A is $\sum_{i=1}^{N} R_i$, which follows a binomial distribution with proportion parameter p equal to $P(R = 1)$. We use the Wilson method [1] to compute 95% confidence intervals around p .

The results confirm Hypothesis 1, showing that there is a substantial bias toward optimum A_0 when basin A has fast evaluation times and basin B has slow evaluation times (see Figure 2). This result serves as a clear demonstration that evaluation-time bias does indeed occur in asynchronous EAs.

Now, a more realistic concern is how evaluation-time bias affects the chance of premature convergence. Building on the intuition that the algorithm is biased away from slow-evaluating regions of the space, we form the following additional prediction:

Hypothesis 2a: On the fitness function $f_{a,b}$, set $b > a$, so that B_0 is the unique global optimum. When evaluation time is constant ($t(\vec{x}) = 1$), the asynchronous EA will converge prematurely to optimum A_0 with some probability p.

Hypothesis 2b: When basin A has *faster* evaltimes than basin B ($t(\vec{x}) = f_{a,-b}(\vec{x})$), the algorithm will converge prematurely to A_0 with a probability *greater* than p.

Hypothesis 2c: When basin A has *slower* evaltimes than basin B ($t(\vec{x}) = f_{-a,b}(\vec{x})$), the algorithm will converge prematurely to A_0 with a probability *less* than p.

We tested this by setting $b = 1.5a$ and running experiments similar to the above. Figure 3 shows the results. When evaluation-times are constant, about 16% of the runs converge prematurely. When the local optimum is faster, the premature convergence rate rises to 27%, while it drops

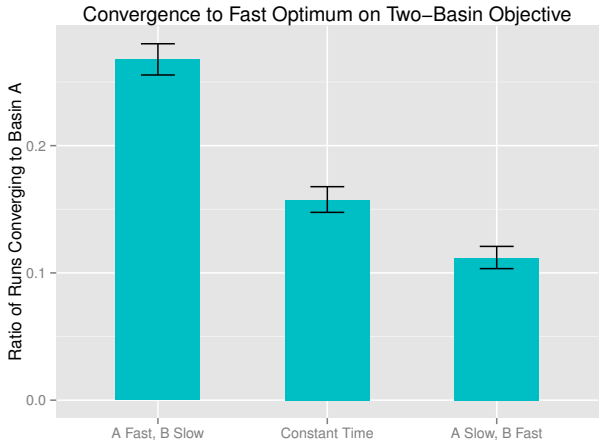

Figure 3: The fraction of runs that converge prematurely to A_0 when B_0 has better fitness.

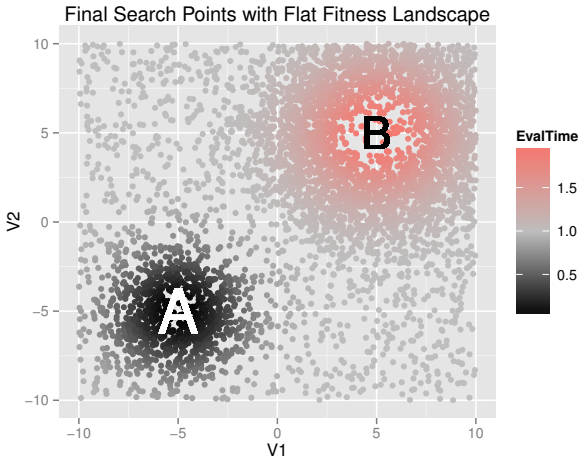

Figure 4: Solutions returned from independent runs on a flat fitness landscape. In the absence of a fitness gradient, the asynchronous EA is attracted to both the fast and slow regions of the space.

to 11% when the global optimum is faster and the local optimum is slow, confirming all three parts of Hypothesis 2. All the differences are statistically significant at $p < 0.05$.

These results are consistent with the simple intuition that asynchronous EAs prefer fast solutions and avoid slow ones. The truth is not quite that simple, however. This can be seen by setting $t(\vec{x}) = f_{a,-b}(\vec{x})$, so that basin A is fast and B is slow, and running the asynchronous EA on a flat fitness landscape, so that the search trajectory is determined solely by evaluation time bias. Figure 4 is a scatterplot of search points that result from 5,000 runs of the asynchronous EA with this configuration. The runs are carried out to 100 steps, and the each point shows a random individual from the final population. In the absence of a fitness gradient, we might expect the algorithm to wander toward the fast optimum. In fact, however, we see that search points cluster heavily around both the fast *and* slow regions, indicating that sometimes the algorithm prefers to move toward slow regions.

4. CONCLUSION

We have confirmed on a simple test problem that there exist cases where asynchronous evolutionary algorithms are biased toward regions of the search space that have faster evaluation times. Such an evaluation-time bias may cause or help prevent premature convergence, depending on the relationship between individual evaluation-times and the fitness landscape.

The dynamics that lead to the evaluation-time bias are complex, however, as we observe evidence that the EA is attracted to both fast and slow regions of the search space on a flat fitness landscape, and away from medium-speed regions. Further study is needed if we wish to understand the cause of this behavior and its practical implications.

In [4], we observed no evaluation time bias on a flat fitness landscape when evaluation time was defined by a binary gene that could take one of two values, `fast` or `slow`. That experiment was simpler than the one we present here and had a smaller number of runs, and may not have been rich enough for an evaluation-time bias to occur or to be detected.

Acknowledgments

This work was funded by U.S. National Science Foundation Award IIS/RI-1302256.

5. REFERENCES

[1] L. D. Brown, T. T. Cai, and A. DasGupta. Interval estimation for a binomial proportion. *Statistical science*, pages 101–117, 2001.

[2] E. Cantu-Paz. *Efficient and accurate parallel genetic algorithms*. Springer, 2000.

[3] A. W. Churchill, P. Husbands, and A. Philippides. Tool sequence optimization using synchronous and asynchronous parallel multi-objective evolutionary algorithms with heterogeneous evaluations. In *IEEE Congress on Evolutionary Computation (CEC) 2013*, pages 2924–2931. IEEE, 2013.

[4] E. O. Scott and K. A. De Jong. Understanding simple asynchronous evolutionary algorithms. In *Proceedings of the Thirteenth Workshop on Foundations of Genetic Algorithms*, FOGA XIII '15, New York, NY, USA, 2015. ACM.

[5] M. Yagoubi, L. Thobois, and M. Schoenauer. An asynchronous steady-state NSGA-II algorithm for multi-objective optimization of diesel combustion. In H. Rodrigues, editor, *Proceedings of the 2nd International Conference on Engineering Optimization*, volume 2010, page 77, 2010.

[6] M. Yagoubi, L. Thobois, and M. Schoenauer. Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs. In *IEEE Congress on Evolutionary Computation (CEC) 2011*, pages 21–28. IEEE, 2011.

[7] A.-C. Zăvoianu, E. Lughofer, W. Koppelstätter, G. Weidenholzer, W. Amrhein, and E. P. Klement. On the performance of master-slave parallelization methods for multi-objective evolutionary algorithms. In *Artificial Intelligence and Soft Computing*, pages 122–134. Springer, 2013.