Evaluation-Time Bias in Quasi-Generational and Steady-State Asynchronous Evolutionary Algorithms

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

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

ABSTRACT

A number of papers have emerged in the last two years that apply and study asynchronous master-slave evolutionary algorithms based on a steady-state model. These efforts are largely motivated by the observation that, unlike traditional (synchronous) EAs, asynchronous EAs are able to make maximal use of many parallel processors, even when some individuals evaluate more slowly than others. Asynchronous EAs do not behave the same as their synchronous counterparts, however, and as of yet there is very little theory that makes it possible to predict how they will perform on new problems. Of some concern is evidence suggesting that the steady-state versions tend to be biased toward regions of the search space where fitness evaluation is cheaper. This has led some authors to suggest a so-called 'quasi-generational' asynchronous EA as an intermediate solution that incurs neither idle time nor significant bias toward fast solutions. We perform experiments with the quasi-generational EA, and show that it does not deliver the promised benefits: it is, in fact, just as biased toward fast solutions as the steady-state approach is, and it tends to converge even more slowly than the traditional, generational EA.

Keywords

Evolutionary Algorithms, Parallel and Distributed Algorithms, Asynchronous Algorithms

1. INTRODUCTION

When the time it takes to evaluate the fitness of candidate solutions to a problem varies, the performance and efficiency of parallel evolutionary algorithms (EAs) can suffer. In particular, classical master-slave EAs that use a generational model of evolution leave some processors idle as they wait for the longest-evaluating individual in each generation to return a fitness value [5]. In extreme cases, as much as 50% of the available computational resources can be left idle as a direct result of variance in evaluation time [7]. The greater the variance in evaluation times, the lower the utilization of the processors.

The problem of idle time has become especially important as applications that involve using search and optimization to tune the parameters of computationally expensive simulations become more popular in science and engineering (ex. [6, 7, 18, 19]). Variance in evaluation times arises for different reasons in a wide class of applications: when tuning the parameters of a computationally intensive simulation, for instance, some configurations may cause the simulator to engage in more expensive and time-consuming operations than others. In genetic programming, a large, bloated program structure will take more time to evaluate than a small, parsimonious one. In a distributed evolutionary algorithm, as a third example, evaluation times may vary if the load or computational capacities of the available compute nodes are heterogeneous. Wherever it arises from, variance in evaluation time can have a pronounced effect on the behavioral characteristics of an EA.

The form this effect takes depends on the algorithm. A number of researchers and practitioners have sought to reclaim idle resources by moving away from a synchronous, generational model and instead adopting an asynchronous, steady-state model (ex. [2, 8, 9]; a more comprehensive overview can be found in [16]). In these steady-state asynchronous EAs (SSAEAs), individual offspring are generated one-at-a-time and sent to a slave processor to have their fitness evaluated. As soon as an individual completes evaluating, it immediately competes for a place in the existing population, without waiting to synchronize with any other individuals. In this way, when fitness evaluation is the most expensive operation in the algorithm, asynchronous parallel EAs have the virtue of being able to keep an unlimited number of processors operating at a nearly 100% utilization rate. The last two years have seen a small surge of new publications that apply or analyze asynchronous EAs of this kind [11, 12, 13, 14, 15, 16, 17, 21].

While SSAEAs can eliminate idle time, they do so at the cost of introducing a dependence between the evolutionary trajectory that the population takes and the evaluation-time characteristics of the problem. Specifically, it seems that these methods exhibit an implicit selection bias that favors individuals that are cheap to evaluate. In some cases, such an evaluation-time bias may be desirable. Martin et al. observe, for instance, that a bias toward fast-evaluating individuals in genetic programming may provide a favorable parsimony pressure [12], and Yagoubi et al. construct an example of a multi-objective optimization problem in

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908934

which penalizing high-quality solutions by giving them long simulated evaluation times improves the performance of an SSAEA by helping to prevent premature convergence [20]. In general, however, practitioners are wary of evaluation-time bias, as it may either hinder the algorithm's ability to find high-quality solutions quickly, or it may otherwise introduce unwanted implications. In scientific applications, especially, the goal is often to find regions of a model's parameter space that maximize goodness of fit to some data set—an additional pressure toward models that are computationally efficient may undermine the validity of such a study's conclusions.

As a consequence of these concerns, some authors have suggested novel algorithms that combine some aspects of existing synchronous and asynchronous EAs, in an attempt to achieve the best of both worlds. In particular, Fonseca and Fleming [10] have proposed what they call a 'quasi-generational' EA (QGEA). The QGEA uses an asynchronous evaluation scheme to minimize idle time, but as new individuals are evaluated, they are are not incorporated directly into the parent population P in steady-state fashion. Instead they are added one-at-a-time to a separate child population P'. Once the child population becomes full, it replaces the parent population, and a new, empty child population is initialized. The key difference between the QGEA and the classical generational scheme is that the QGEA generates more than $|P|$ children from each set of parents, as a means of keeping all of the processors occupied. Fonseca and Fleming argue that this approach retains dynamics that are similar to the generational EA, and that by doing so it will reduce the "bias towards individuals easy to evaluate." They also observe that it is possible to select all of the parents up front, rather than one-at-a-time—and that it is thus possible to use Stochastic Universal Sampling in conjunction with the QGEA [3].

While the quasi-generational EA has been suggested and discussed, to our knowledge Durillo et al. are the only authors who have implemented a QGEA (applying it to a multi-objective optimization problem) [9]. The advantages and disadvantages that the QGEA has in comparison to the generational EA and SSAEA is an entirely open question.

In this paper, we compare the behavior of the quasi-generational and steady-state asynchronous EAs. In particular, we use synthetic problems to examine the extent to which each algorithm shows a bias toward fast-evaluating solutions.

RQ1: What are the pros and cons of the quasi-generational EA (QGEA)? Specifically, does the QGEA exhibit less evaluation-time bias than the steady-state asynchronous EA (SSAEA)?

Evaluation-time bias arises in a complex way from the interaction between variance in evaluation times, the availability of processors, the population size and the fitness landscape. As of yet, there is no theory of evaluation-time bias in EAs that explains these interactions in any significant way, or that begins to make it possible to predict how evaluation-time bias will affect an EA's performance on particular problems. Defining what it means for one algorithm to have "less" evaluation-time bias than another is itself an open research question. Investigating **RQ1**, then, will lead us to ask some basic questions about evaluation-time bias in asynchronous EAs and how it works:

RQ2: How does evaluation-time bias arise in asynchronous EAs? What does it mean for one algorithm to be more biased than another?

2. METHODOLOGY

We begin by presenting the details of the algorithms under study, and summarizing the hypotheses and experimental framework we will use to operationalize our research questions.

2.1 Master-Slave Algorithms

Of the rich menagerie of parallel metaheuristics that have been proposed and studied over the years [1], the master-slave evolutionary algorithms are among the simplest and most frequently used in practice. A master-slave EA maintains a single, global population on the master processor, and dispatches individuals in the population out to slave processors to have their fitness evaluated. Sometimes reproductive operators are also executed on slave nodes. Master-slave EAs are beneficial when the cost of fitness evaluation is substantially greater than the cost of transmitting individuals from the master to the slaves.

2.1.1 The Generational Master-Slave EA

The best known master-slave EA is the parallel generational evolutionary algorithm. Most readers are likely to be quite familiar with this EA, so we only pause to offer a few clarifying remarks. By 'generational,' we mean that this algorithm operates in the style of a (μ, λ)-EA, where μ parents are selected from the population to create λ offspring (typically, $\mu = \lambda$). The parents are then discarded, and the offspring become the new population, with no overlap (generation gap) between the generations. Fitness evaluation is performed in parallel by sending the λ offspring out to a number of slave processors. The algorithm waits for all individuals to have their fitness evaluated before the parents for the next generation are selected.

Any time there is variance in evaluation times, the processors used by the generational EA incur idle time. The amount of time the slaves spend in an idle state is determined by both the number of slaves relative to the population size and the distribution of evaluation times among the offspring.

2.1.2 The Steady-State Asynchronous EA

Asynchronous master-slave EAs eliminate the primary source of idle time by ensuring that, any time a processor has become free, a new offspring individual is immediately produced to take its place. Almost all asynchronous global-population EAs to date have been built on the steady-state model: Individuals are integrated into the population one-at-a-time immediately after their fitness has been evaluated, in the style of a $(\mu + 1)$-EA.

A general pseudocode template for asynchronous evolution is described in Algorithm 1 from the perspective of the master processor. Both the steady-state asynchronous EA (SSAEA) and the quasi-generational EA (QGEA) can be de-

scribed in terms of this asynchronous template—they differ only in the implementation of the INTEGRATE() method.

Algorithm 1 A General Asynchronous EA

```
1: function ASYNCHRONOUSEVOLUTION(μ, T, steps)
2:     P ← ∅
           ▷ Begin parallel initialization of the population
3:     for i ← 1 to T − 1 do
4:         SEND(RANDOMINDIVIDUAL())
5:     while |P| < μ do
6:         SEND(RANDOMINDIVIDUAL())
7:         P ← P ∪ {NEXTEVALUATEDINDIVIDUAL()}
                              ▷ Begin main evolutionary loop
8:     for i ← 0 to steps do
9:         SEND(BREEDONE(P))
10:        INTEGRATE(NEXTEVALUATEDINDIVIDUAL(), P)
```

The first two loops of Algorithm 1 initialize the population by sending randomly-generated individuals to a free slave node for evaluation (SEND()). As individuals complete evaluating, they are retrieved one-by-one via NEXTEVALUATEDINDIVIDUAL() and added to the population. Together, these two loops generate a total of $\mu + T - 1$ random individuals, where T is the number of slave processors—enough to fill the population *and* ensure that all but one processor is fully occupied when evolution begins.

Evolution then proceeds in the **for** loop, which generates an offspring individual and sends it to the free processor—BREEDONE() here indicates both parent selection and reproductive operators—and then integrates the next individual that completes evaluating into the population. In the SSAEA, the INTEGRATE() method takes the form of Algorithm 2. The newly evaluated individual competes against an individual chosen by SELECTONE()—if the offspring individual is better, then it replaces the selected individual in the population.

Algorithm 2 Steady-state insertion into a population

```
1: function INTEGRATESTEADYSTATE(ind, P)
2:     replaceInd ← SELECTONE(P)
3:     if BETTERTHAN(ind, replaceInd) then
4:         P ← (P − replaceInd) ∪ ind
```

This high-level description leaves a number of design decisions up to the practitioner. The choice of selection operators is not integral to our research question in this paper—but in our experiments we use tournament selection of size 2 to select parents in BREEDONE(), and we use random selection for survival selection (SELECTONE()). We consider one individual to be BETTERTHAN() another if its fitness is greater than *or equal to* the other. The details of the representation and reproductive operators we use will be explained below.

2.1.3 The Quasi-Generational Asynchronous EA

The QGEA has been proposed as an asynchronous variant of the generational EA, where idle time is reclaimed by generating extra offspring to fill the vacant CPU resources [9, 10]. As individuals complete evaluating, they are added to an offspring population. When the offspring population reaches a size equal to the parent population, it replaces the parent population, and a new, empty offspring population is created. In this way, observe Durillo et al., "the master does not have to wait until all the individuals of a generation have been evaluated" [9].

For simplicity, we interpret the QGEA as a variant of the general asynchronous EA given in Algorithm 1. To implement generational dynamics, we define an INTEGRATE() procedure that inserts individuals into an offspring population until it is full—see Algorithm 3. Here we treat the offspring population P' as a global variable that is initialized to be the empty multiset.

Algorithm 3 Quasi-generational insertion into a population

```
1: function INTEGRATEGENERATIONAL(ind, P)
2:     P′ ← P′ ∪ {ind}
3:     if |P′| = |P| then
4:         P ← P′
5:         P′ ← ∅
```

This integration method ensures that changes to the population occur in a generational way, in the sense that the population is completely replaced every μ steps.

2.2 Analyzing Evaluation-Time Bias

Since we are interested in the case where fitness evaluation dwarfs other EA overhead, experiments with parallel 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 the above algorithms via a discrete-event simulation, in which each individual $\vec{x}$ is assigned an evaluation time from an artificially-imposed evaluation-time function $t(\vec{x})$ 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.

Intuitively, the reason that we anticipate an evaluation-time bias in the asynchronous EAs 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 minimizes a paraboloid function. Evaluation time is proportional to fitness, and the EA utilizes 10 simulated processors. 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. Once 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.

The implications that this observation has for the dynamics of an EA are presently not well understood. Naïvely, we might expect that the fast-evaluating subset of the population in an asynchronous EA are akin to the famously exponential growth of rabbits in 19th-century Australia: the faster you mature, the more fast-evaluating children you will have, and so forth. However, besides the fact that the total population size is fixed, there are at least two reasons that

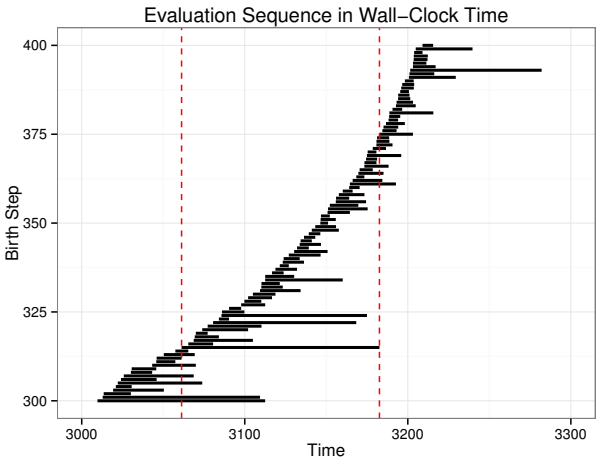

Figure 1: A sequence of fitness evaluation durations in an SSAEA.

the reproductive advantage that fast individuals receive in an asynchronous EA is limited in a way that it was not for the Australian rabbits:

1. While a slow individual is evaluating, the CPU resource it is occupying is made unavailable. So, if half the processors are occupied with slow individuals, only half remain for fast-evaluating individuals to exploit.

2. Individuals do not reproduce as soon as they finish evaluating. Evaluations trigger a breeding event that produces a new child, but the parents are selected from the general population. Individuals only reproduce when they are selected.

These interactions between available processors and individual evaluation times give rise to a variety of open questions. This paper focuses on: 1) Do asynchronous EAs seek out fast-evaluating solutions even on flat fitness landscapes? 2) Can evaluation-time bias lead an algorithm to prefer to move into certain basins of attraction over others? 3) Does evaluation-time bias occur mostly close to initialization, or does it occur throughout the run?

To answer these questions we report here on a number of experiments in which we compare the steady-state and quasi-generational asynchronous EAs by examining their behavior on flat fitness landscapes, by examining which optimum they converge to on a simple multimodal problem, and by isolating bias at initialization from bias that occurs later in the run.

The experiments in this paper are conducted on simple real-valued optimization problems. These problems are not intended to form a representative benchmark with real-world features. The purpose of this kind of study is rather to gain targeted insight into aspects of the algorithms' behavior. In particular, we focus on how the evaluation-time properties of landscape impact the behavior of each algorithm. Each synthetic problem is defined by a tuple of two functions (f, t), where $f(\vec{x})$ represents the fitness of individual $\vec{x}$ and $t(\vec{x})$ defines its evaluation time (in arbitrary units). Because there is always some small amount of variation in the time a function takes to execute on a computer, thanks to process scheduling effects if nothing else, we always add a small amount of Gaussian noise to the simulated evaluation time $t(\vec{x})$.

In all the experiments below, we represent solutions as vectors in $\mathbb{R}^n$, and breed single children by applying Gaussian mutation with hard bounds and a standard deviation of 0.5, mutating each gene individually with probability $1/n$. We do not use crossover, and we consider only the case where the number of slave processors T is equal to the population size. These design decisions clearly limit the generality of our results, but they are sufficient for our purpose, which is to isolate the effect of evaluation-time bias, rather than to study its interactions with particular representations or reproductive operators.

3. RESULTS

3.1 Bias on Flat Fitness Landscapes

Evaluation-time bias can be understood as an implicit selection effect: at some point during a run, fast-evaluating individuals gain a reproductive advantage over slow ones—that is, they are selected as parents more frequently than can be explained by other factors (such as fitness). The simplest and most direct way to study evaluation-time bias, then, is to ask what happens when the fitness of all individuals is equal.

We hypothesize that, on a flat fitness landscape, both asynchronous EAs will display a strong evaluation-time bias at the beginning of each run, but that bias will be negligible beyond that point. Bias at the beginning of the run clearly ought to occur for the following reason. During the initialization phase of Algorithm 1, a total of $\mu + T - 1$ individuals are generated while the population is being filled, and the very first μ individuals that complete evaluating are the ones that form the initial population. The initial population, then, will contain a disproportionately large number of fast-evaluating individuals.

> **Hypothesis 1**: Let the quasi-generational EA and the steady-state asynchronous EA be run on a flat fitness landscape, and let different parts of the landscape take different amounts of time to evaluate. Then, for both algorithms:
>
> a) At the beginning of the run, fast-evaluating individuals will become overrepresented in the population.
>
> b) After initialization, the population will show *no preference* for fast or slow regions of the search space.

We test Hypothesis 1 by visualizing the location of the population in the search space over time. The heat maps in Figure 2 show the result of running the SSAEA with population sizes $\mu = 10$ and $\mu = 100$ on a task where the fitness function is flat ($f(x) = 1$), and evaluation times follow a 1-dimensional parabola centered on the origin ($t(x) = x^2$)—so individuals close to 0 are very fast-evaluating. Initial individuals are generated randomly from a uniform distribution across the genotype space, $\mathcal{U}(-10, 10)$. The heat map shows the distribution of the population as a function of the number of evolutionary steps, averaged over 500 runs. In each

plot, the dashed line represents the end of the initialization phase—i.e. the step at which the population is first filled.

At the beginning of the run, a clear pattern of initialization bias emerges: individuals with very short evaluation times enter the population first, while the slowest individuals do not begin to enter the population until around step 400 (in the $\mu = 100$ case). When the population is large, we see that the initial overrepresentation of fast individuals persists—confirming Hypothesis 1a. Similar results were obtained for the QGEA (not shown).

When the population is small, however ($\mu = 10$), the effect of mutation is strong enough to quickly erase the effects of initial evaluation-time bias. In fact, by step 2,000, the average distribution of the population for both the SSAEA and the QGEA is statistically indistinguishable from a uniform distribution (using a Kolmogorov-Smirnoff test with $p > 0.05$). We obtained similar results when we replaced the evaluation-time function $t(x)$ with a two-Gaussian function that has a fast and a slow region (as shown on the left of Figure 3), a linear function ($t(x) = x - 10$), and with a Rastrigin function (not shown). This indicates that any evaluation time bias that exists on flat fitness landscapes after the initialization period is so weak that it can be entirely overpowered by the disruptive effects of mutation. We take this as confirmation of Hypothesis 1b: no observable evaluation-time bias exists after initialization on flat landscapes.

3.2 Preference for Fast Basins of Attraction

Our second experiment considers non-flat fitness landscapes, and is designed to detect even very weak biases:

> **Hypothesis 2**: Let the quasi-generational EA and the steady-state asynchronous EA be run on a fitness landscape that has two basins of attraction, and let one basin of attraction have fast evaluation times, and the other have long evaluation times. Then:
>
> a) Both algorithms will be more likely to converge to the fast basin than the slow one.
>
> b) The SSAEA will exhibit a stronger preference for the fast basin than the QGEA does.

If true, Hypothesis 1b is evidence in favor of the conjecture of Fonseca and Fleming that the QGEA has a reduced "bias towards individuals easy to evaluate" compared to the SSAEA [10]. We test this on an objective function defined over $\mathbb{R}^n$ 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. When the depth parameters a and b are equal, the two basins are identical from a fitness perspective. Because of the symmetry in the objective function, most evolutionary algorithms will converge to either of the two optima with equal probability.

In addition to the fitness function, we introduce an asymmetry into the problem by assigning each individual $\vec{x}$ a simulated evaluation time $t(\vec{x})$, where the eval-time function $t : \mathbb{R}^n \mapsto \mathbb{R}$ is given by:

$$t(\vec{x}) = f_{a,-b}(\vec{x}). \quad (2)$$

Evaluation-time is thus equal to fitness, except that evaluation-time basin surrounding optimum B_0 is inverted. This ensures that A_0 is surrounded by fast-evaluating solutions, and B_0 is surrounded by slow-evaluating solutions.

We ran $N = 5,000$ independent runs of both asynchronous EAs out to 2,000 steps—long enough so that all N runs converged. The population size is 10 and we use 10 simulated slave processors, and the search space has two dimensions. The result of each run can be represented by the random variable

$$R_i = \begin{cases} 1 & \text{if } \|\vec{x}' - A_0\| \leq \tau \\ 0 & \text{if } \|\vec{x}' - B_0\| \leq \tau \end{cases}, \quad (3)$$

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 r equal to $P(R = 1)$. We use the Wilson method [4] to compute 95% confidence intervals around r. If the asymmetry in evaluation times has no affect on the basin the algorithm coverages to, then r will be equal to 0.5.

The empirical results of the two-basin experiment are shown in Figure 4. Both the QGEA and the SSAEA display a statistically significant preference for the faster basin, confirming Hypothesis 1a. However, we find that the QGEA displays a *stronger* evaluation-time bias than the SSAEA. The difference is small, but statistically significant at $p < 0.05$. On these grounds, **we reject Hypothesis 2b**. Under the conditions of the two-basin experiment, it turns out that the QGEA is in fact quite biased toward fast solutions.

3.3 Initialization Bias in the Two-Basin Function

The two-basin problem gives us limited insight into evaluation-time bias. We know that both asynchronous EAs are biased at initialization, and initialization bias alone may be sufficient to set the population on a trajectory toward the faster optimum. Here we repeat the two-basin experiment from Hypothesis 2, but now we control for evaluation-time bias at initialization.

> **Hypothesis 3**: Most of the evaluation-time bias in the QGEA and SSAEA occurs during the initialization stage of the run.

We test this by configuring both the fitness function $f(\vec{x})$ and the evaluation-time function $t(\vec{x})$ to have a *constant* value for the first 100 steps of each evolutionary run. After 100 steps, the functions revert back to their original values of $f(\vec{x}) = f_{a,b}(\vec{x})$ and $t(\vec{x}) = f_{a,-b}(\vec{x})$ (with $a = b$). This way there is no evaluation-time bias during initialization. Any bias that we can still measure is due to the dynamic interaction between the fitness landscape and evaluation time.

The results—again as estimates of the probability of each algorithm converging to optimum A_0—are shown in Fig-

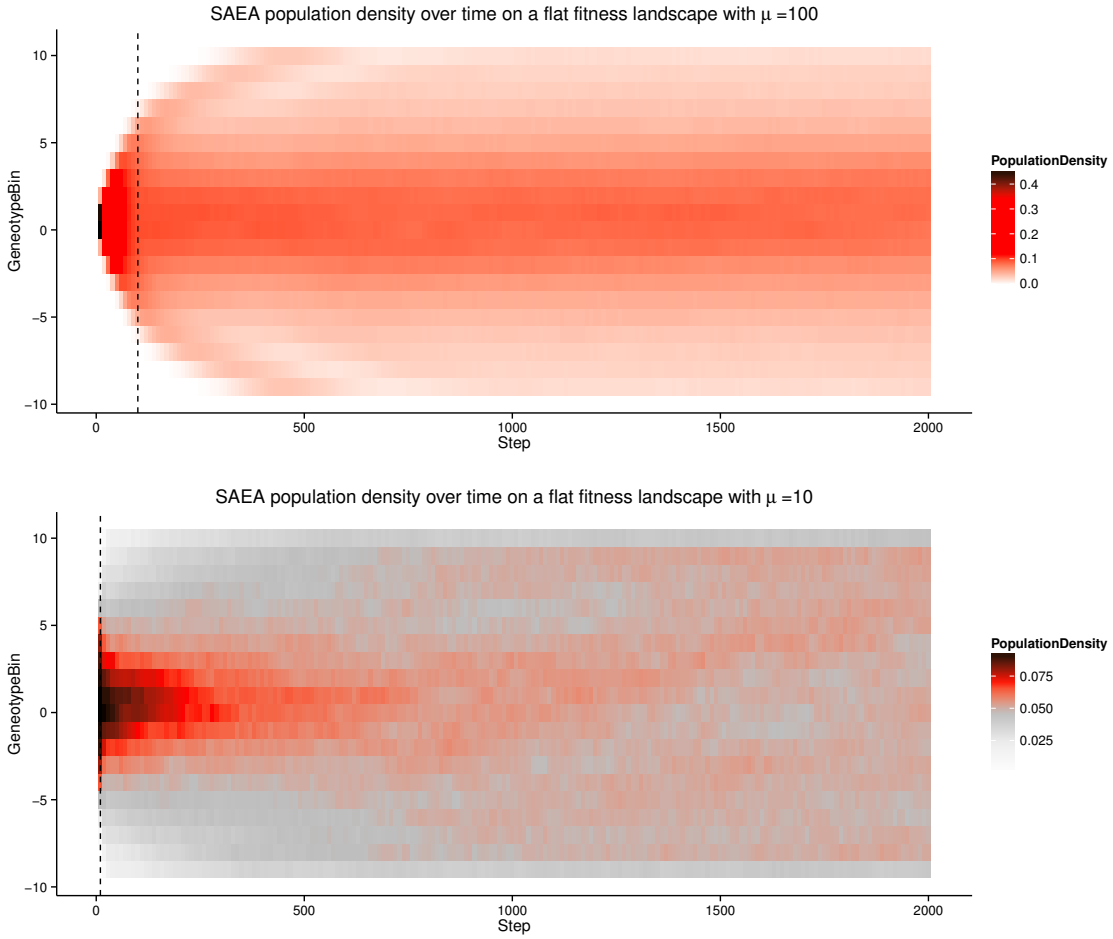

Figure 2: Distribution of genotypes over time in an SSAEA on a one-dimensional flat fitness landscape, where individual evaluation times are equal to the square of the genotype. Top: Population size of $\mu = 100$. Bottom: $\mu = 10$.

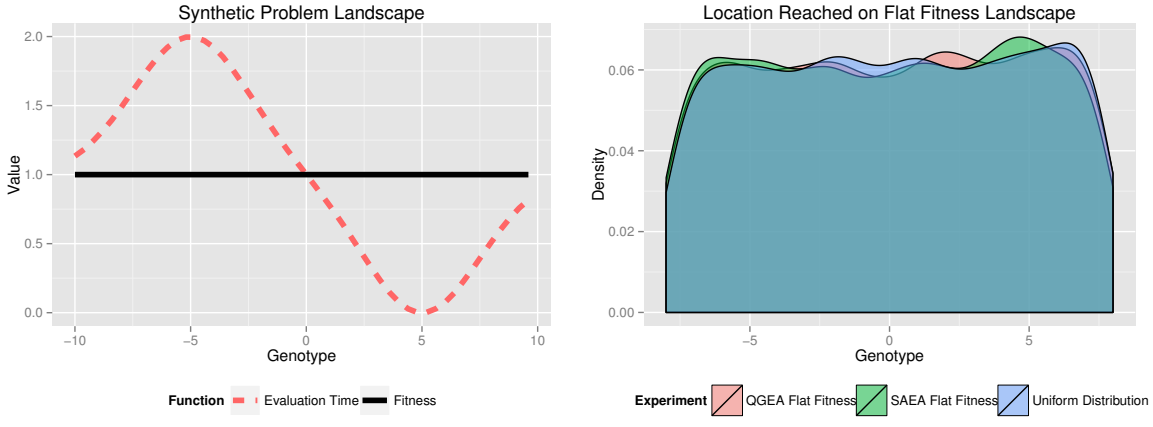

Figure 3: Left: A synthetic evaluation-time function over a 1-dimensional genotype space, and a flat fitness function. Right: Distribution of individuals in the final population after 2,000 evolutionary steps.

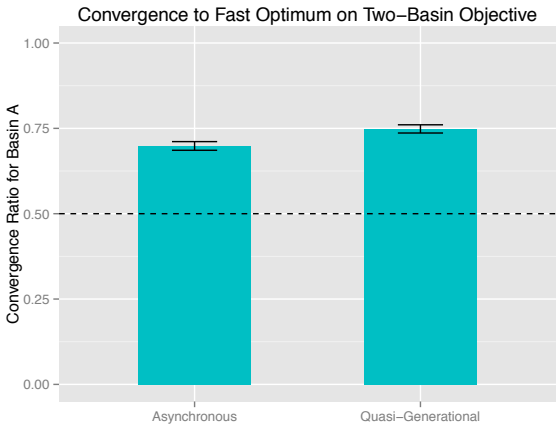

Figure 4: Ratio of runs that converge to the faster of two basins of attraction.

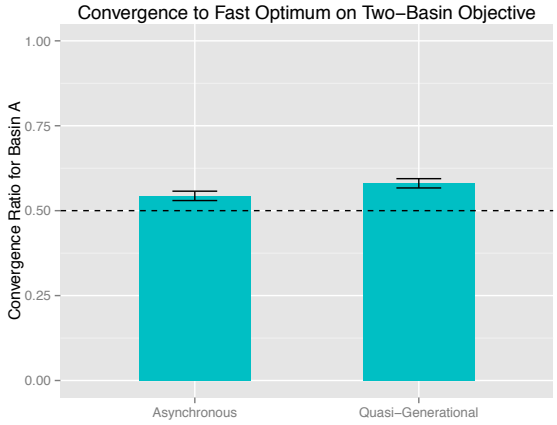

Figure 5: Ratio of runs that converge to the faster of two basins of attraction when initialization bias is controlled for.

ure 5. Removing initialization bias caused a significant reduction in this measure of evaluation-time bias, confirming Hypothesis 3. Both algorithms still display a statistically significant preference for optimum A_0, however. This indicates that, unlike flat fitness landscapes, we can detect an evaluation-time bias on non-flat fitness functions that occurs after initialization.

3.4 Convergence of the QGEA

We conclude with a brief observation of a potential disadvantage of the QGEA apart from evaluation-time bias: the QGEA, it turns out, is an algorithm that converges very slowly on unimodal functions. Figure 6 shows the number of steps that it takes the generational EA, the SSAEA, and the QGEA to reach a threshold fitness value of $\tau = 0.5$ on paraboloid functions of differing dimensionality. In this experiment, evaluation times are held constant. Each bar depicts the convergence time of $N = 500$ independent runs. The quasi-generational EA consistently takes longer to converge than the generational EA. This is a consequence of the asynchronous dynamics of the QGEA, which cause it to generate more than μ children at each generation.

4. DISCUSSION

Asynchronous evolutionary algorithms are able to make efficient use of parallel processing resources when fitness evaluation is expensive enough to make the master-slave model viable—but asynchronous EAs bring evaluation-time bias with them as side effect. Evaluation-time bias may hinder the results of some applications, have no impact on other applications, and in some cases it may even be beneficial. The current state of the literature does not give us the theoretical insight we need to make accurate predictions about how evaluation-time bias will impact EA performance on a particular problem.

We began our investigation of the quasi-generational EA (**RQ1**) on the belief that it can serve as a behavioral intermediate to the classical, generational EA and the steady-state asynchronous EA. In this study, however, we found that this is not the case. In the simplified scenarios we used to measure evaluation-time bias, we find that the QGEA has a slightly *stronger* preference for fast-evaluating regions of the search pace than the SSAEA. Furthermore, the evolutionary trajectory of the QGEA does not mimic the generational EA closely, but instead takes substantially longer to converge on smooth, unimodal landscapes. Our results suggest that the QGEA does not have the particular benefits that it has been conjectured to offer.

Secondly, this paper marks the first attempt that we know of to analyze the nature of evaluation-time bias in asynchronous evolutionary algorithms (**RQ2**), especially by distinguishing between bias that occurs at initialization and bias that occurs later in the run. Despite the strong intuitive case to be made that evaluation-time bias can significantly affect the trajectory of an EA under the right conditions, the empirical evidence seems to suggest that the effect is relatively mild after initialization.

Many open questions still remain about evaluation-time bias in asynchronous EAs that fall outside the scope of this study. Our experiments made a number of assumptions—in particular, we used a fixed mutation width and no crossover, and we assumed that the number of processors was equal to the population size. In our experience, the representation and operators do not have a major effect on the kind of results that we have presented here—we observed some similar results, for instance, with higher-dimensional search spaces, with crossover enabled, and with binary representations on simple pseudo-Boolean functions. These kind of parameters may have a substantial impact on the magnitude of evaluation-time bias in certain circumstances, however. Furthermore, the simple two-basin objective that we used to measure evaluation-time bias, while a useful instrument for our purposes here, may not capture the information that is most useful for predicting the behavior of an EA on new problems. Future empirical studies could move closer to a predictive theory of evaluation-time bias by investigating more complex landscapes, or by considering how a population moves asynchronously along smooth fitness gradients.

5. ACKNOWLEDGMENTS

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

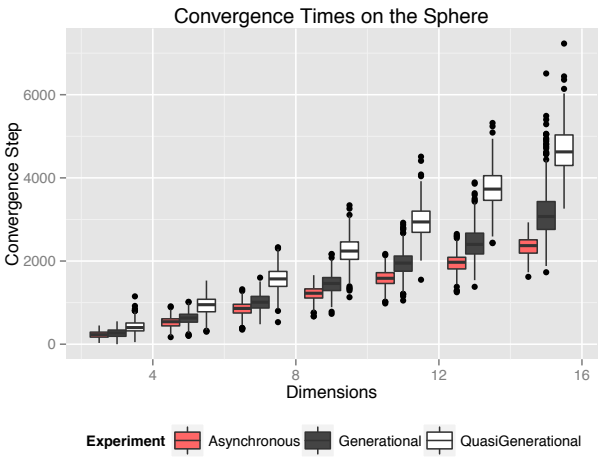

Figure 6: Time-to-convergence for master-slave EAs on the paraboloid function.

6. REFERENCES

[1] E. Alba, editor. *Parallel Metaheuristics: A new Class of Algorithms*. John Wiley & Sons, Hoboken, New Jersey, 2005.

[2] E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17(4):451–465, Jan. 2001.

[3] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms (ICGA'87)*, pages 14–21, 1987.

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

[5] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Springer, 2000.

[6] K. Carlson, J. Nageswaran, N. Dutt, and J. Krichmar. An efficient automated parameter tuning framework for spiking neural networks. *Frontiers in Neuroscience*, 8(10), 2014.

[7] 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'13)*, pages 2924–2931. IEEE, 2013.

[8] M. Depolli, R. Trobec, and B. Filipič. Asynchronous master-slave parallelization of differential evolution for multi-objective optimization. *Evolutionary Computation*, 21(2):261–291, 2013.

[9] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba. A study of master-slave approaches to parallelize NSGA-II. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS) 2008*, pages 1–8. IEEE, 2008.

[10] C. M. Fonseca and P. J. Fleming. Parallel implementation of multiobjective genetic algorithms. In *Symposium on Intelligent Systems in Control and Measurement*, Miskolc, Hungary, 1998.

[11] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, and Q. Zhang. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 2015.

[12] M. A. Martin, A. R. Bertels, and D. R. Tauritz. Asynchronous parallel evolutionary algorithms: Leveraging heterogeneous fitness evaluation times for scalability and elitist parsimony pressure. In *Companion Proceedings of the 2015 Conference on Genetic and Evolutionary Computation (GECCO'15)*, pages 1429–1430, Madrid, Spain, 2015. ACM.

[13] R.-E. Reisch, J. Weber, C. Laroque, and C. Schröder. Asynchronous optimization techniques for distributed computing applications. In *Proceedings of the 48th Annual Simulation Symposium*, pages 49–56. Society for Computer Simulation International, 2015.

[14] S. M. Said and M. Nakamura. Master-slave asynchronous evolutionary hybrid algorithm and its application in vanets routing optimization. In *3rd International Conference on Advanced Applied Informatics (IIAI AAI'14)*, pages 960–965. IEEE, 2014.

[15] C. Salto and E. Alba. Adapting distributed evolutionary algorithms to heterogeneous hardware. In *Transactions on Computational Collective Intelligence XIX*, pages 103–125. Springer, 2015.

[16] E. O. Scott and K. A. De Jong. Understanding simple asynchronous evolutionary algorithms. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms (FOGA'15)*, pages 85–98, New York, NY, 2015. ACM.

[17] K. Tagawa and H. Takeuchi. Dynamic implementation techniques of concurrent differential evolutions for multi-core CPUs. In *14th International Conference on Software Engineering, Parallel and Distributed Systems*, Dubai, United Arab Emirates, 2015.

[18] M.-H. Tayarani, X. Yao, and H. Xu. Meta-heuristic algorithms in car engine design: a literature survey. *IEEE Transaction on Evolutionary Computation*, 19(5):609–629, 2015.

[19] W. Van Geit, E. De Schutter, and P. Achard. Automated neuron model optimization techniques: A review. *Biological Cybernetics*, 99(4-5):241–251, 2008.

[20] M. Yagoubi, L. Thobois, and M. Schoenauer. Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs. In *IEEE Congress on Evolutionary Computation (CEC'11)*, pages 21–28. IEEE, 2011.

[21] A.-C. Zăvoianu, E. Lughofer, W. Koppelstätter, G. Weidenholzer, W. Amrhein, and E. P. Klement. Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems. *Knowledge-Based Systems*, 2015.