Understanding Simple 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

In many applications of evolutionary algorithms, the time required to evaluate the fitness of individuals is long and variable. When the variance in individual evaluation times is non-negligible, traditional, synchronous master-slave EAs incur idle time in CPU resources. An asynchronous approach to parallelization of EAs promises to eliminate idle time and thereby to reduce the amount of wall-clock time it takes to solve a problem. However, the behavior of asynchronous evolutionary algorithms is not well understood. In particular, it is not clear exactly how much faster the asynchronous algorithm will tend to run, or whether its evolutionary trajectory may follow a sub-optimal search path that cancels out the promised benefits. This paper presents a preliminary analysis of simple asynchronous EA performance in terms of speed and problem-solving ability.

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

Evolutionary algorithms are increasingly being used to tune the parameters of large, complex and stochastic simulation models in various domains of science and engineering. In these applications, evaluating the fitness of an individual may take on the order of minutes or hours of CPU time. Parallel evaluation is thus essential to obtaining EA results from these models in a tolerable amount of "wall-clock" time.

In the neuroscience and agent-based modeling applications we are involved with, we observe a non-negligible amount of parameter-dependent variance in the evaluation times of the simulation models being tuned. At any given generation, the slowest individual in the population may take several times longer to evaluate than the fastest-evaluating individuals.

FOGA '15 January 17 - 20, 2015, Aberystwyth, United Kingdom
ACM Copyright 2015 ACM 978-1-4503-3434-1/15/01 . . . $15.00.
http://dx.doi.org/10.1145/2725494.2725509

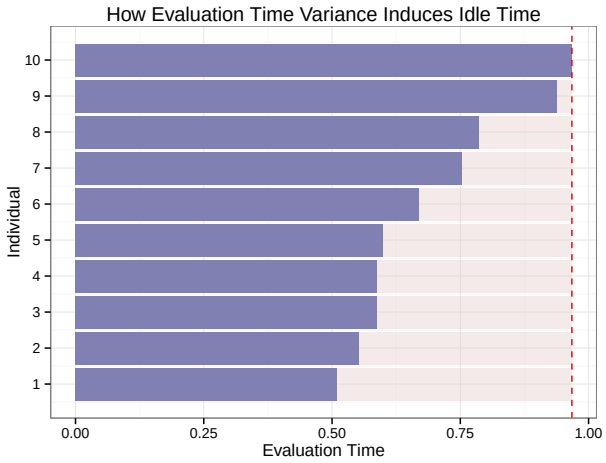

Figure 1: When there is variance in individual evaluation times, a parallelized generational EA suffers idle time.

This poses a problem for the 'generational' master-slave EAs which practitioners often use to parallelize the solution of optimization and design problems. Traditional master-slave EAs synchronize their threads at each generation, in imitation of the sequential EAs they are derived from. As all the CPUs must wait for the individual with the longest evaluation time to complete before moving to the next generation, a significant amount of CPU idle time may result. Figure 1 illustrates this with 10 simulated evaluation times sampled uniformly from [0.5, 1.0]. The lightly shaded region shows the idle CPU time induced as 9 of the 10 nodes wait for the next generation.

A natural alternative, which we analyze in this paper, is to abandon the (μ, λ) generational scheme and turn to a $(\mu + 1)$-style EA with asynchronous evaluation. Here, new individuals are generated one-at-a-time by the selection and reproduction operators as CPUs become available, and are integrated into the population immediately when they finish having their fitness evaluated. As the processing nodes never wait for a generation boundary, idle time is virtually eliminated. We refer to algorithms that follow this approach

as *simple asynchronous evolutionary algorithms* (the details of which will be given in Section 1.2).

Simple asynchronous EAs promise to eliminate the performance impediment that eval-time variance induces in generational master-slave EAs, thereby increasing *throughput* in terms of the number of individuals evaluated per unit time. One contribution we offer in this paper is a model (under simplified assumptions) of just how much extra throughput a simple asynchronous EA gains over a generational EA. This increase in computational efficiency is purchased, however, at the cost of a significant change in EA behavior. It could be that, on some problems, the increased throughput in fitness evaluations is offset by slower evolutionary convergence.

The aim of this paper is to further our understanding of that potential trade-off. The practitioner would like to know whether she can reliably expect beneficial results from a simple asynchronous EA, as compared to a parallel (μ, λ)-style EA. Little to no such guidance currently exists in the literature.

The remainder of this section gives some background on parallel and asynchronous EAs that have been studied, introduces the simple asynchronous evolutionary algorithm up close, and details concerns that may arise about its performance. Sections 2, 3 and 4 proceed with theoretical and empirical analysis of the algorithm's speed and behavior, and we conclude with a summary in Section 5.

1.1 Background

In general, evolutionary algorithms use either a *global* population model (*panmixia*), in which selection and reproduction operators apply to a single monolithic population, or they use a *structured population* model that decomposes the population into components that interact locally. Structured models can be further divided into the 'course-grained' island models and 'fine-grained' cellular EAs, respectively, yielding three broad families of EA (four if you count hybrid approaches separately). Parallelization of all three families has been studied extensively (cf. surveys in [1, 4, 17]).

Asynchronous communication appears only occasionally in the parallel EA literature. Island models have been created that permit asynchronous migrations between subpopulations [2, 13], and asynchronous cell updating has been studied in cellular EAs [10], but in general asynchronous mechanisms have not been advertised as promising major performance advantages for structured population models. Recent work on 'pool-based' EAs allows subsets of a centralized population to be farmed out asynchronously to slave processors for evolution, as a sort of intermediate between panmictic and structured population models (ex. [15]). The purely panmictic (global) population model, however, is by far the easiest kind of EA to parallelize, and is thus the most widely used parallel EA in practice.

Panmictic parallel EAs typically take the form of a (μ, λ)-style generational EA, in which fitness evaluation is performed in parallel, either on a shared-memory machine or over a distributed cluster [5]. Selection and reproduction operators typically operate sequentially on the master processor (though these can sometimes be parallelized as well). These master-slave EAs are especially effective when the execution time of fitness evaluation dwarfs the cost of selection and reproduction. Thanks to its synchronization at each generation (cf. Figure 2), the generational master-slave EA

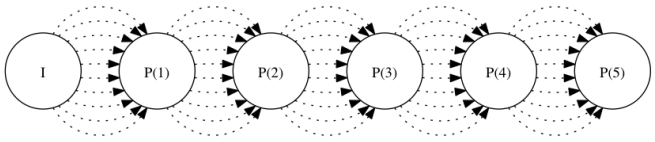

Figure 2: A generational master-slave EA synchronizes after each population has been evaluated in parallel.

is also well-understood: its evolutionary behavior is identical to its slower, sequential counterpart. The potential drawback of synchronization, and the primary motivation for turning to an asynchronous approach, is the idle time already pointed out by Figure 1.

Asynchronous algorithms based on the $(\mu + 1)$-style steady-state EA have been used intermittently by practitioners for decades as an easy-to-implement means of reducing idle CPU resources (e.g. [5, 20]). First used in genetic algorithms [32], asynchronous evaluation was introduced into multi-objective EA (MOEA) applications as early as 1995 (ex. [24, 26]), and similar approaches have been used in master-slave implementations of ant colony optimization [25], differential evolution [18], and particle swarm optimization [12, 16, 27].

Interest in the asynchronous master-slave scheme (which we describe in more detail below) has grown in recent years as EA applications involving computationally expensive simulations become more common. For instance, Churchhill et al. apply an asynchronous MOEA to a tool-sequence optimization problem for automatic milling machine simulations, finding that both the synchronous and asynchronous methods achieve solutions that are comparable in quality after a fixed number of evaluations, but that the asynchronous method completes those evaluations in 30-50% less wall-clock time [6]. Yagoubi et al. similarly apply an asynchronous MOEA to the design of an engine part in a simulation of diesel combustion, with favorable results after a fixed number of evaluations [29].

While several successful applications of asynchronous master-slave EAs are attested in the literature, few studies have attempted to tease out a theoretical or empirical understanding of what kinds of problems they may be poorly- or well-suited for. As we will see below, the performance of the asynchronous EA depends in a readily evident but poorly understood way on the number of slave processors and the distribution of individual evaluation times.

Zeigler and Kim were among the first to suggest the benefits of asynchrony in master-slave EAs and to perform preliminary analysis of their throughput and problem solving capacity [11, 32]. They observed that, with an asynchronous EA, an unlimited number of threads may be used to keep a cluster fully utilized, even when the number of available processors is greater than the population size. Runarsson has shown empirically, however, that as the number of slave processors is increased, more function evaluations are needed for his asynchronous evolution strategy to make progress on unimodal functions [21]. The extra throughput is thus apparently purchased at the cost of making the algorithm less greedy than a non-parallelized steady-state EA.

This is caused by what Depolli et al. call the *selection lag*, defined as "the number of solutions created while an

observed solution is being evaluated" [8]. The selection lag describes the key behavioral difference between the asynchronous EA and the steady-state EA.

While our focus in this paper is on single-objective problems, most recent analysis of the asynchronous master-slave model has taken place in the context of multi-objective optimization, using MOEA approaches that are inspired by the steady-state EA. In empirical studies on small test suites, Durillo et al. and Zăvoianu et al. each find that the asynchronous approach performs well at finding good Pareto fronts in less time than other approaches [9, 31]. Zăvoianu et al. also use an argument based on Amdahl's law to put a lower bound on the speedup in evaluations-per-unit-time that the asynchronous approach provides as compared to a generational approach. They suggest that the asynchronous EA can provide some improvement in computational capacity even when evaluation times are very short and have negligible variance. When evaluation times are much longer than the EA's sequential operations (i.e. reproduction, selection), negligible speedup is predicted unless there is variation in evaluation times. Their analysis applies unmodified to single-objective EAs.

A concern that we give considerable attention to in this paper is how the distribution of evaluation times changes over the course of an evolutionary run, and how this may impact performance. In particular, in some applications, solutions that lie close to the optimum may take much longer (or shorter) to evaluate than poor solutions. It's not clear how the relationship between the fitness landscape and heritable evaluation time traits may affect EA performance. The only prior work we know that has raised this question is Yagoubi et al.'s empirical analysis of a multi-objective test suite in [30], which found that an asynchronous MOEA had a harder time finding good solutions when they were located in a region of the search space that was artificially configured to have slower evaluation times. This raises the concern that asynchronous EAs may in general have a bias toward fast-evaluating individuals.

1.2 The Simple Asynchronous EA

The key notion behind asynchronous master-slave EAs is that we integrate an individual into the population as soon as it finishes evaluating, and immediately generate a new individual to take its place on the open CPU resource. There seems to be no way to accomplish this without abandoning the generational model, which is inherently synchronous, and instead introducing a generation gap (cf. [23]), so as to allow a continuous evolutionary process.[1] The simplest generation gap algorithm is the $(\mu + 1)$-style steady-state EA, and almost all asynchronous master-slave EAs are based on the steady-state EA.

Figure 3 illustrates the difference between traditional steady-state evolution and the simple asynchronous EA. In the steady-state model, a single individual i is generated by selecting parents from the population $P(t)$ at step t. The new individual has its fitness evaluated, and then competes for a place in the population. How that competition takes place is a design decision, but a typical approach (and the one we take in this paper) is to choose a random individual in $P(t)$, and have i replace it in the population *iff* i has a fitness strictly better than the randomly chosen individual. The result of this replacement is the new population $P(t + 1)$. By convention, we say that one 'generation' has passed in a steady-state or simple asynchronous EA when μ individuals have been generated, where μ is the population size.

[1]That said, we note that Durillo et al. have created a hybrid approach in [9] that they call 'asynchronous generational.'

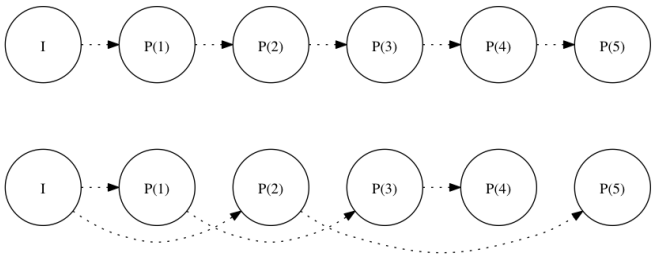

Figure 3: In steady-state evolution, individuals compete for a space in the population at the subsequent step (Top). In the simple asynchronous EA, several evolutionary steps may pass before an individual enters the population (Bottom).

In the asynchronous model, at any given time more than one individual is having its fitness evaluated. In the bottom of Figure 3, $T = 2$ slave processors are in use. As the population is initialized, $T - 1$ extra initial individuals are created to keep the slaves busy while the first evolutionary step executes. From that point on, individuals that are being evaluated on the slaves compete for a place in the population as soon as they finish evaluating. As a result, several evolutionary steps may take place while an individual evaluates.

The simple asynchronous EA we use in this work is detailed from the perspective of the master processor in Algorithm 1. The first loop initializes the population by sending randomly generated individuals to a free node for evaluation (`send`()). When this `while` loop terminates, there are n individuals in the population that have had their fitness evaluated, there are $T - 1$ randomly generated individuals currently being evaluated on the T slaves, and there is one free slave node. 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.

In our experiments below, we use tournament selection of size 2 to select parents in `breedOne`(), and we use random selection for survival selection (`selectOne`()), as parent selection already provides ample selection pressure. Practitioners (for instance, [19]) also sometimes choose to use a FIFO replacement strategy, amongst others, instead of random replacement, since Sarma and De Jong have shown in [22] that using a FIFO strategy in $(\mu + \lambda)$-style EAs leads to search behavior that is more like the (μ, λ)-style EA. See [28] for a study of survival selection strategies in a simple asynchronous EA.

Algorithm 1 The Simple Asynchronous EA

```
1: function AsynchronousEvolution(n, gens)
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 (n · gens) 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)
```

Once the asynchronous dynamics depicted in Figure 3 are understood, several concerns about the algorithm's behavior become readily apparent.

As Rasheed and Davison observe with their application of a similar asynchronous EA, "the creation of a new individual may not be affected by individuals created one or two steps ago because they have not yet been placed into the population" [20]. That is, the parallelism introduces *selection lag*, which (as mentioned above) is the number of evolutionary steps that go by while an individual is being evaluated. Selection lag occurs whether or not there is variance in individual evaluation times. A sequential steady-state EA always has a selection lag of 0, but Depolli et al. prove that, even in the presence of evaluation-time variance, the average selection lag in $(\mu + 1)$-style asynchronous EAs is $T - 1$ steps, where T is the number of slave processors [8].

"Even worse," continue Rasheed and Davison, variance in evaluation times induces a *re-ordering effect*: the algorithm "may get back individuals in a different order than originally created, as some processors/processes may complete their evaluations faster than others (such as a result of heterogeneous processing environments, external loads, etc.)."

Variance in individual evaluation times may originate in a heritable component and/or a non-heritable component.

- **Non-Heritable** eval-time variance may arise from heterogenous CPU resources, load conditions, and process scheduling effects – but it can be especially pronounced in cases where fitness evaluation involves executing a stochastic simulation.
- **Heritable** eval-time variance arises where there is a dependency between the parameters of the simulation and the time it takes to execute. Heritable variance may be independent of fitness, or there may be a relationship between fitness and evaluation time.

We would like to understand how these sources of variance impact both an asynchronous EA's throughput (the number of individuals evaluated per unit time) and its ability to converge to an optimal solution to a problem. In particular, in the remainder of this paper we investigate three research questions, treated now below.

First, we are concerned about the role that the re-ordering effect described above may play in altering the EA's search trajectory. When evaluation time is a heritable trait, many fast-evaluating individuals may be born, evaluated, and compete for a place in the population in the time it takes for a single slower individual to evaluate. This raises the concern that simple asynchronous EAs may be biased away from slow-evaluating regions of the search space. As mentioned in Section 1.1, Yagoubi et al. observe evidence that this does in fact occur in a multi-objective context [30]. They found that on at least one test problem, an evaluation-time bias helped to prevent premature convergence. An evaluation-time bias could just as easily *cause* premature convergence in other problems, however, so if eval-time bias is a significant aspect of asynchronous EA behavior, then it is something practitioners ought to be made aware of.

> **RQ 1**: When individual evaluation times are a heritable trait, does the asynchronous EA give a reproductive advantage to faster-evaluating individuals? That is, is the EA biased toward individuals with lower evaluation times?

After studying evaluation-time bias, we turn to the more basic question of how much idle CPU resources a simple asynchronous EA can reclaim, compared to a generational EA.

> **RQ 2**: How great an increase in throughput does the simple asynchronous EA offer over a parallelized generational EA? How does it depend on the population size, number of processors, and the distribution of evaluation times? How does it depend on how the distribution of evaluation times changes over the course of the evolutionary run?

Doing more fitness evaluations in the same amount of time is only beneficial if those extra evaluations can be put to work to make progress toward the optimum. It's not obvious how the asynchronous and generational EAs differ in balancing exploration and exploitation, so we compare the two algorithms' performance on single- and multi-modal test functions.

> **RQ 3**: How does the convergence time of an asynchronous EA compare to what we would expect from the increase in throughput? Can we be both fast and smart? How does the asynchronous EA's performance depend on the relationship between evaluation time and fitness?

In the next three sections we address **RQ 1**, **RQ 2** and **RQ 3** in turn. The evolutionary behavior of the asynchronous EA is complex and difficult to describe in a purely analytical way. For instance, the replicator dynamics of the system are (perhaps not surprisingly) non-Markovian. As such, while we use some preliminary analytical results where possible, this paper relies heavily on empirical studies to be-

gin improving our understanding of asynchronous EAs. All our experiments use the asynchronous EA implementation provided by the ECJ evolutionary computation toolkit [14], which closely follows Algorithm 1.

2. EVALUATION-TIME BIAS

The intuition depicted in Figure 3 would seem to indicate that when evaluation time is a heritable trait, fast-evaluating individuals may obtain a reproductive advantage in an asynchronous EA. This can be worrying in some applications, such as when scientific simulations are being tuned so that their results match some experimental data. The goal of such parameter tuning is to find the best-fitting model – not a model that runs faster than the alternatives!

The simplest way to study a possible bias toward fast-evaluating genotypes is in isolation from other evolutionary effects. We define a heritable runtime trait by a gene that can take on one of two alleles: `fast` and `slow`, where `slow` takes 10 times longer to evaluate. Further assume that there is no reproductive variation (offspring are generated only by cloning), and that the fitness landscape is flat – i.e. every individual has an equal chance of being selected as a parent. Each cloned offspring replaces a random individual in the population with 100% probability. In this scenario, any systematic change in genotype frequency can only be due to some implicit selection acting on the evaluation-time trait.

Now, given a number of evolutionary steps, will a disproportionate number of `fast`-type individuals be produced on average? We can approach this by letting the random variable X denote the number of `fast`-type individuals that complete evaluation during a fixed interval of m evolutionary steps. Then define X_i as the indicator random variable

$$X_i = I\{\text{The individual evaluated at step } i \text{ is } \texttt{fast}\text{-type}\}, \tag{1}$$

which evaluates to 1 if the proposition is true and 0 otherwise. Now, the linearity of expectation shows that we can answer our question by considering each X_i independently of any other part of the evolutionary trajectory:

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{m} X_i\right] \tag{2}$$

$$= \sum_{i=1}^{m} \mathbb{E}[X_i] \tag{3}$$

$$= \sum_{i=1}^{m} p(X_i = 1) \tag{4}$$

$$= \sum_{i=1}^{m} f_i, \tag{5}$$

where f_i is the probability that the processor that completed evaluation at step i contained a `fast`-type individual. But since individuals are cloned from a randomly selected parent, f_i is simply the expected frequency of `fast`-type alleles that existed when the individual was first cloned and sent off for evaluation.

For large populations we know that f_i will not change very quickly. Empirically, we find that, apart from a very short transient period at initialization, the expected value of the f_i's is roughly constant on a flat landscape with a population size of 100 (Figure 4). We seeded the initial population with random genotypes that were drawn with equal probability

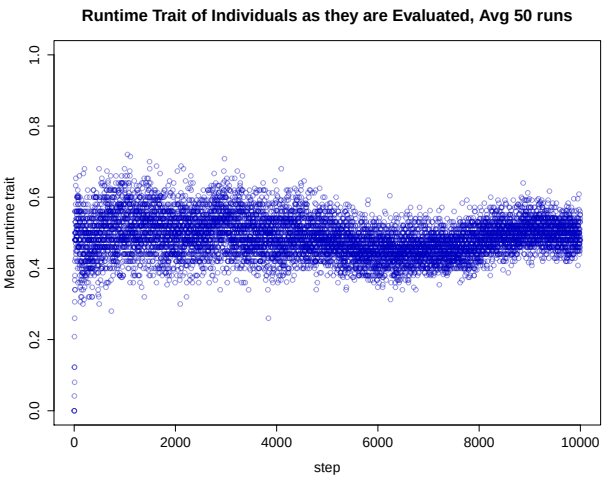

Figure 4: Each point represents the fraction of individuals out of 50 independent runs that finished evaluating with `slow` type at that step. Since the fraction hovers around 0.5, the frequency of each genotype does not change very much.

from both `fast` and `slow` alleles, so that the initial `fast`-type frequency averaged $f = 0.5$. We then ran the asynchronous EA with 10 slave processors for a total of 50 independent runs. This allows us to estimate the expected value of f_i at any step i. We find that in all but just the first few steps, f_i consistently hovers around $f = 0.5$ without any systematic trend up or down.

By Equation 5, then, $\mathbb{E}[X]$ reduces to mf for any value of m. That is, if $f = 0.5$ and $m = 10$ evolutionary steps, we expect to see 5 `fast`-type individuals and 5 `slow`-type individuals produced, for no net change in the population genotype. By this analysis, we do not anticipate any change in the expected genotype frequencies on flat landscapes outside the brief transient phase.

This is confirmed in our experiments (Figure 5). We measured the genotype frequency over time in 50 independent runs and found no evidence of selection pressure favoring either allele. At least on flat landscapes, we have a clear indication that there is no selective advantage for fast fitness evaluations. Similar results were obtained for the case where individual runtimes are allowed to vary continuously along an interval, and for population sizes of 10 and 500 (not shown).

Expected value models like the one we just derived are useful in characterizing the ensemble averages of a sufficiently large number of finite-population models, but not the behavior of individual runs. Notice that in Figure 5 the variance in genotype frequency over the 50 independent runs continues to increase over time. This is an indication that there are in fact significant changes in the `slow`/`fast` ratios on individual runs. This is due to the well-studied problem of genetic drift in finite populations (see, for example, [7]). Alleles are lost simply due to sampling variance occurring in the random parent and survival selection steps. This drift occurs even if the evaluation times of `fast` and `slow`-type individuals are set to be equal.

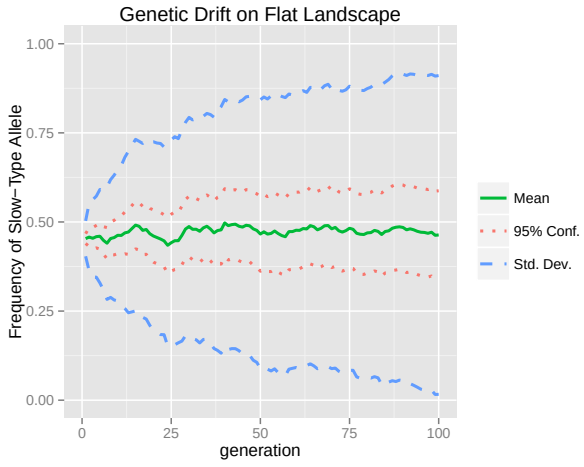

Figure 5: Frequency of `slow`-type individuals on a flat fitness landscape, averaged over 50 runs. No systematic drift toward `fast` alleles is observed.

We have thus answered **RQ 1** in the negative on flat fitness landscapes with cloning: there is no independent selection pressure that favors fast-evaluating individuals. This does not, however, rule out an evaluation-time bias that emerges from a combination of reproduction, variation and selection, so our results do not contradict the bias observed in practice by Yagoubi et al. [30].

3. SPEEDUP IN THROUGHPUT

Asynchronous EAs are interesting primarily because they promises to increase the total number of individuals that complete evaluation per unit of wall-clock time – i.e. *throughput*. We measure throughput by considering the time it takes for an EA to execute a fixed number of evaluations. We call the ratio between two algorithms' throughput the *improvement* in throughput, or the *throughput speedup*. It is important to distinguish throughput improvement from *true speedup*, which would take the quality of the resulting solution into account.[2] In this section we are concerned with the throughput improvement of the simple asynchronous EA over the parallel generational EA. We defer consideration of true speedup – i.e. convergence time – to Section 4.

3.1 A Model of Throughput Improvement

As we stated in the opening to this paper, we have assumed that the evaluation time of individuals dwarfs all other EA overhead, and thus that an asynchronous EA has near-zero idle time. Under this assumption, the speedup in throughput that the asynchronous EA offers is completely described by the amount of idle CPU resources it recovers. Algebraically, the throughput speedup is thus expressed as the ratio

$$S = \frac{1}{1-\hat{I}}, \tag{6}$$

where $\hat{I}$ is the fraction of CPU resources that the generational EA would have left idle.

[2]In [31], Zăvoianu et al. use the term 'structural improvement' for what we call 'throughput speedup.'

To get a quantitative handle on what actual values for the throughput speedup S might look like, consider the case where individual evaluation times are independent and identically distributed according to some distribution. Furthermore, assume for simplicity that the ratio of CPUs available to the population size $T/n = 1$, i.e. there is one CPU for each individual in the population. We will now analyze the expected value of S.

Let $P = \{Y_1, Y_2, \ldots, Y_n\}$ be a set of values drawn i.i.d. from some distribution, where $Y_i \in \mathbb{R}^+$ represents the evaluation time of the ith individual in the population. Then, as illustrated in Figure 1, the absolute idle time suffered by the generational EA is the total number of CPU-seconds processors spend waiting for the longest-evaluating individual to complete:

$$I = \sum_{i=1}^{n} (\max[P] - Y_i). \tag{7}$$

To express idle time as a normalized value between 0 and 1, $\hat{I}$, we divide I by the total number of CPU-seconds available to the algorithm during the generation, which is $n \max[P]$.

$$\hat{I} = \frac{I}{n \max[P]}. \tag{8}$$

Computing the expected value of this ratio distribution is difficult for most distributions, in part because $\max[P]$ and I are not independent. We can derive a lower bound on the expectation, however, if we confine our attention to the case where the Y_i's are sampled from a uniform distribution over the interval $[a, b]$.

$$\mathbb{E}[\hat{I}] = \mathbb{E}\left[\frac{I}{n \max[P]}\right] \tag{9}$$

$$\geq \mathbb{E}\left[\frac{I}{nb}\right] \tag{10}$$

$$= \frac{1}{nb}\mathbb{E}\left[\sum_{i=1}^{n}(\max[P] - Y_i)\right] \tag{11}$$

$$= \frac{1}{b}\left(\mathbb{E}[\max[P]] - \mathbb{E}[Y]\right), \tag{12}$$

where the last step follows by the linearity of expectation. Since $\max[P]$ quickly approaches b as $n \to \infty$, this lower bound on $\mathbb{E}[\hat{I}]$ will be tight for large n.

Finally, in Appendix A we show by some straightforward calculus that for the uniform distribution,

$$\mathbb{E}[\max[P]] = b - \frac{1}{n+1}(b-a). \tag{13}$$

Therefore, in the case where all runtimes are uniformly distributed between a and b, we have the following lower bound on the expected normalized idle time:

$$\mathbb{E}[\hat{I}] \geq \frac{1}{b}\left[b - \frac{1}{n+1}(b-a) - \left(a + \frac{b-a}{2}\right)\right] \tag{14}$$

$$= \left(\frac{b-a}{b}\right)\left(\frac{1}{2} - \frac{1}{n+1}\right), \tag{15}$$

where n is population size, which we have assumed is equal to the number of processors.

We can see from Equation 15 that the expected idle time is determined entirely by the population size n and the ratio of the standard deviation (which is related to $(b-a)$ by a constant factor) to its maximum value b. Moreover, the

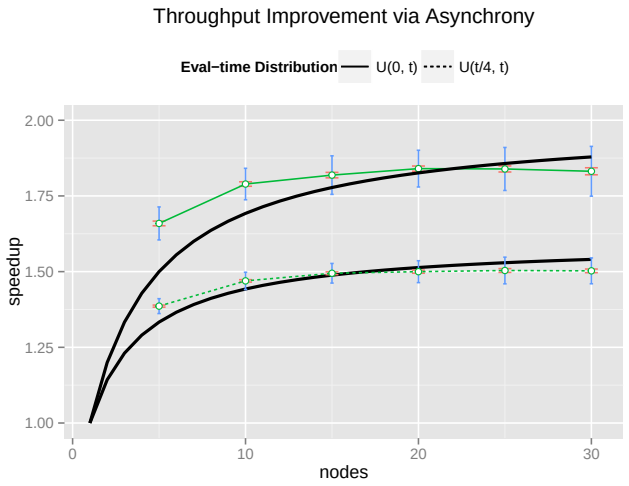

Figure 6: Observed throughput improvement, shown along with theoretical lower bounds predicted by Equation 15 (bold lines).

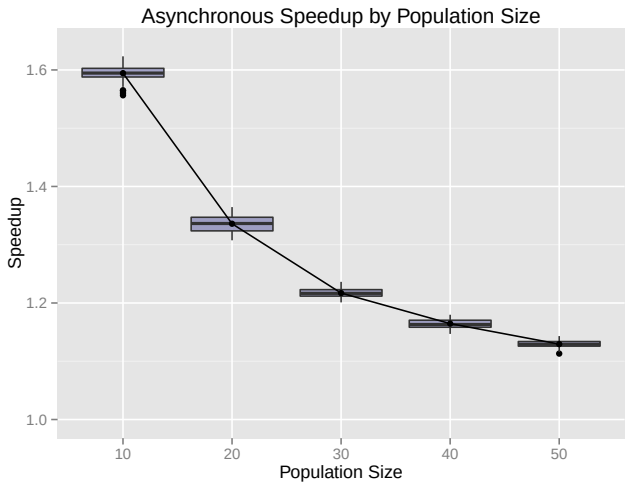

Figure 7: Observed throughput improvement when the number of slave processors is fixed at 10 and the population size varies.

maximum attainable speedup is determined by the limit of the idle time as n grows:

$$\lim_{n\to\infty} \mathbb{E}[\hat{I}] = \frac{1}{2}\left(\frac{b-a}{b}\right). \tag{16}$$

This result indicates that the generational EA will never incur an idleness greater than 50% when evaluation times follow a uniform distribution, and consequently an asynchronous EA can never provide a throughput improvement of greater than 2. We note in passing that for other distributions, such as the Gaussian, a speedup of much greater than 2 is possible.

3.2 Experimental Validation

We used Equation 15 to predict the idle time of the generational EA, and converted this into a prediction of speedup via Equation 6. Figure 6 compares the result against simulation experiments.

Each data point represents the speedup in throughput measured by pairing 50 independent runs of an asynchronous EA against 50 runs of a generational EA. Each algorithm was run for as close as possible to 500 fitness evaluations.[3] For the lower curve, individual evaluation times were non-heritable and sampled uniformly from the interval $[\frac{t}{4}, t]$, where t is a sufficiently long time that EA overhead is negligible. The upper curve is a similar experiment with a larger amount of variance – evaluation times were sampled from $[0, t]$. Since evaluation times were independent of fitness, the objective function is irrelevant to throughput. To simulate specific evaluation times, individuals were configured to simply wait a period of time. Since this was not resource intensive, we were able to simulate large numbers of processors on a shared-memory machine that had just a few cores. The thin error bars indicate the standard deviation in the speedup across the 50 runs. Tighter, wide error bars showing the 95% confident interval on the mean are barely visible in the figure, indicating that we obtained a precise estimate of the expected speedup.

The results confirm that Equation 15 provides a reasonably tight estimate of the throughput improvement. For large numbers of processors ($T = n > 20$), however, the prediction no longer serves as an accurate lower bound. We found that the degree to which the results conform to the prediction vary somewhat depending on the architecture of the computer we run the experiments on. We surmise that the deviation from the prediction at high n is an artifact of our experimental setup, which simulates more processors than we actually have available.

The assumption that $T = n$ simplified our analytical approach to throughput, but in practice our population size will typically be significantly greater than the number of processors. Figure 7 shows the effect of holding the number of processors fixed at $T = 10$, and varying the population size while individual evaluation times are sampled non-heritably from $[0, t]$ (50 independent runs of 250 generations each). As n increases, each processor becomes responsible for evaluating a larger share of the population, and the throughput improvement quickly decreases.

In the experiments shown so far, the variance in throughput improvement from run to run is very small. When evaluation time is a heritable trait, this is no longer the case, as genetic drift and/or selection can significantly alter the distribution of evaluation times as evolution progresses. The amount of throughput improvement we attain over the entire run depends heavily on how the magnitude and variation of evaluation times expands or shrinks over time. How that change occurs depends in turn on how the heritable component of evaluation time is related to an individual's fitness. We return to these nuances (which form the remainder of **RQ 1**) in Section 4, where we consider several scenarios in which the evaluation-time trait is heritable in some way (ex. Figure 9).

The specific results in this subsection are limited to evaluation times that are uniformly distributed and non-heritable. Qualitatively, however, we expect similar results for other

[3]We say "as close as possible" only because 500 does not divide evenly into an integral number of generations for some population sizes.

simple distributions: asynchronous parallelization is especially advantageous over the generational EA when the number of slave processors is large and the ratio T/n of processors to the population size is high. Decreasing returns appear to set in quickly, however, so it is important to have realistic expectations about how much of an advantage asynchrony will offer.

4. PROBLEM SOLVING

The throughput improvement an asynchronous EA offers over a generational alternative is an intuitively appealing metric, because we are inclined to believe that progress toward the solution can be measured by the number of fitness evaluations an algorithm has completed. In his early analysis of asynchronous master-slave EAs, Kim called this the *requisite sample set hypothesis*, and used it to express the importance of throughput [11]:

> "Given an [algorithm] for which the invariance of the requisite sample set size holds, search speed is determined by the throughput of fitness evaluation. The greater the number of processors dedicated to the evaluation of individuals, and the higher the utilization of these processors, the faster the global optimum will be located."

When the number of samples an asynchronous EA needs to find a high quality solution on the given problem is equal to the number of samples a generational EA requires, then throughput improvement is an accurate predictor of true speedup. This assumption is unlikely to hold in practice. In principle, on some problems the asynchronous EA could require so many more samples that it takes longer to converge than the generational EA, despite the increase in throughput.

In this section we continue our approach of simulating various kinds of evaluation-time distributions, but now on non-flat fitness landscapes. The aim is to gain a preliminary understanding of how gains in throughput combine with the evolutionary trajectory of the asynchronous EA to produce true speedup.

4.1 Methods

The performance of an asynchronous EA depends not only on the fitness landscape of the problem at hand, but also on the how an individual's location in the search space is related to its evaluation time. As such, we used four distinct simulated scenarios on each test function to see how the asynchronous EA performs on real-valued minimization problems. In all four scenarios, we represented individual genomes as vectors in $\mathbb{R}^l$, used a Gaussian mutation operator at a per-gene probability of 0.05, and used a 100% rate of two-point crossover. The population size was fixed at $n = 10$ in each case, and the number of slave processors was also $T = 10$.

1. In the **Non-Heritable** scenario, individual evaluation times are uniformly sampled from the interval $[0, t_{\max}]$.

2. In the **Heritable**, fitness-independent scenario, we define a special gene to represent the individual's evaluation-time trait. The trait is randomly initialized on $[0, t_{\max}]$, and undergoes Gaussian mutation within these bounds with a standard deviation of $0.05 \cdot t_{\max}$. This gene is ignored during the calculation of fitness.

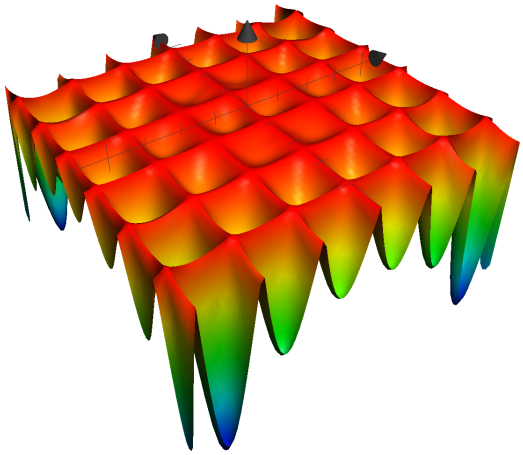

Figure 8: The Hölder table function.

3. In the **Positive** fitness-correlated scenario, the evaluation time $t(\vec{x})$ of an individual $\vec{x}$ is a linear function of fitness with a positive slope m and zero intercept:

$$t(\vec{x}) = mf(\vec{x}) \tag{17}$$

   This simulates the case where evaluation becomes faster as we approach the optimum.

4. In the **Negative** fitness-correlated scenario, evaluation time is a linear function of fitness with a negative slope and a non-zero intercept:

$$t(\vec{x}) = \max\left(0, -mf(\vec{x}) + t_{\max}\right). \tag{18}$$

   In this case, evaluation becomes slower as we approach the optimum (up to a maximum of $t_{\max}$ seconds).

We tested all four scenarios on the 2-dimensional Rastrigin function, the Hölder table function, and the venerable 10-dimensional sphere function. While the Rastrigin function has many local optima, it is linearly separable and has a quadratic macro-structure which makes the global optimum relatively easy to find:

$$f(\vec{x}) = 10l + \sum_{i=1}^{l} [x_i^2 - 10\cos(2\pi x_i)]. \tag{19}$$

The Hölder table function (Figure 8) is also highly multimodal:

$$f(\vec{x}) = -\left|\sin(x_1)\cos(x_2)\exp\left(\left|1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right|\right)\right| + 19.2085, \tag{20}$$

where we have added the non-traditional constant 19.2085 so that the global optima have a fitness of approximately zero. We select the Hölder table because it is moderately difficult, in the sense that the EAs we are studying sometimes converge on a local optimum, and fail to converge on a global optimum after several hundred generations.

During both initialization and the application of reproductive operators, we bound each gene between -10 and 10 on all three objectives, and we set the standard deviation of the Gaussian mutation operator to 0.5. For the fitness-correlated scenarios, we set the parameter m to 1 for the

sphere function and 5 on the Rastrigin and Hölder table. We ran each EA for 250 generations on the sphere function, 500 on the Hölder function, and 1000 on the Rastrigin function.

4.2 Results

4.2.1 Sphere and Rastrigin Functions

Our first observation on non-flat fitness landscapes relates back to Section 2, where we found no evidence of a selective bias toward fast-evaluating individuals on flat landscapes. That result does not preclude the possibility that a special kind of bias could still exist on non-flat landscapes.

In particular, we might anticipate that in the positively correlated scenario, where evaluation time decreases as fitness improves, a preference for fast-evaluating individuals may cause the population to be "accelerated" toward the optimum. Conversely, we might also expect the population to be more reluctant to move into areas of better fitness when evaluation time is negatively correlated with fitness. In fact, however, we observe no statistically significant ($p < 0.05$) difference among the four scenarios in the number of fitness evaluations it takes for the asynchronous EA to converge on the sphere function (not shown). This indicates that as Gaussian mutation leads the population to exploit a unimodal function, the effect of evaluation-time bias is negligible.

Secondly, we would like to know how well the naïve promises of throughput improvement serve as a predictor of true speedup on these simple objectives. Figures 9 and 10 show the true speedup and throughput improvement for each of the four scenarios on the sphere and Rastrigin functions, respectively. The true speedup is measured by considering the amount of wall-clock time it takes each algorithm to reach a fitness value of less than the threshold value $\eta = 2$. Each scenario was tested by pairing 50 independent runs of each algorithm and measuring the resulting speedup distribution.

The asynchronous EA converges in less time than the generational EA in all four scenarios on both functions. On the Rastrigin, but not the sphere, the true speedups are remarkably high. All of them average over 2.0, which is the maximum attainable expected value for throughput improvement as determined by Equation 16, and outliers are visible in Figure 10 with true speedup values in excess of 20 or 30. This is possible because the asynchronous EA requires fewer fitness evaluations to solve the Rastrigin function than the generational EA.

On the sphere function, most of the true speedup is more readily explained by an increase in throughput: We find at $p < 0.05$ with the Bonferroni correction for testing four simultaneous hypotheses that there is no statistically significant difference between the means of the throughput speedup and true speedup in the non-heritable, heritable, and positive scenarios. We reject the hypothesis, however, that the mean true speedup in the negative scenario is equal to the mean of its throughput improvement: the true speedup is higher on average.

When evaluation time is negatively correlated with fitness on a minimization problem, the majority of the processing time is spent in the later stages of the run, where the population consists primarily of high-quality individuals with long evaluation times. The late stages of the run are also where

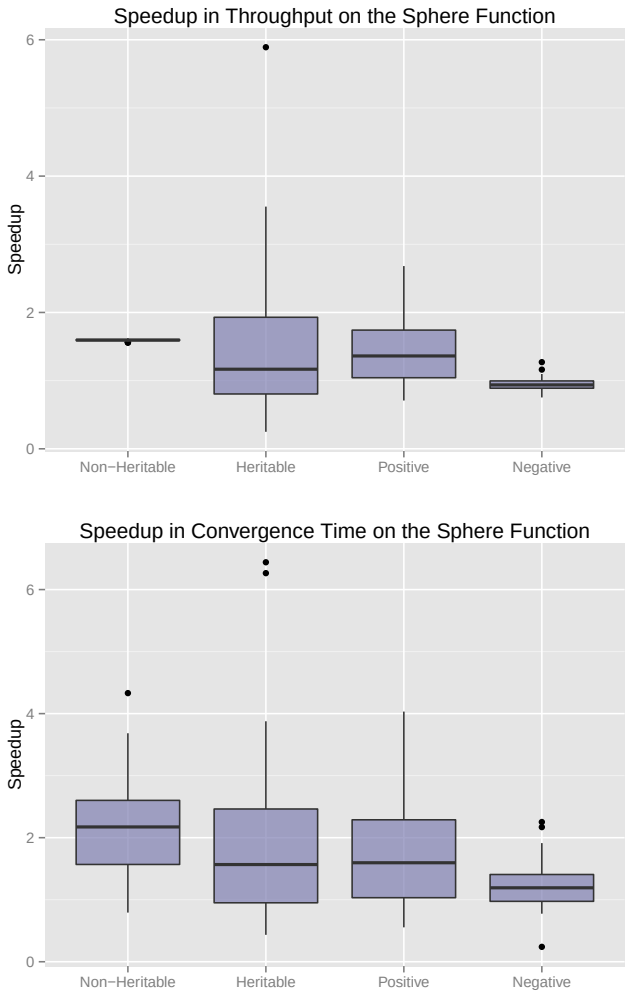

Figure 9: Speedup in throughput (Top) and convergence time (Bottom) of the asynchronous EA on the sphere function.

genetic variation is low in the population, so there is very little evaluation-time variance. As we saw in Section 3, low evaluation-time variance translates to very little difference in throughput between the two algorithms – when there is no eval-time variance, there is no idle time to eliminate. As such, we predict that the throughput improvement in the negative scenario will be very close to 1 on any objective function, provided that the algorithm is run for a sufficiently long period of time.

The true speedup in the negative scenario is observed at an average of 1.23 +/- 0.05 (95% confidence interval) for the sphere function and 3.09 +/- 0.58 for the Rastrigin. The fact that the true speedup is greater than 1 for both functions indicates that it is the evolutionary trajectory of the asynchronous EA, not the elimination of idle time, that is producing the true speedup.

In fact, the asynchronous EA is fundamentally greedier than the generational EA. This can be seen in Figure 11, which depicts average best-so-far trajectories on the sphere function for the non-heritable scenario. $(\mu + \lambda)$-style EAs

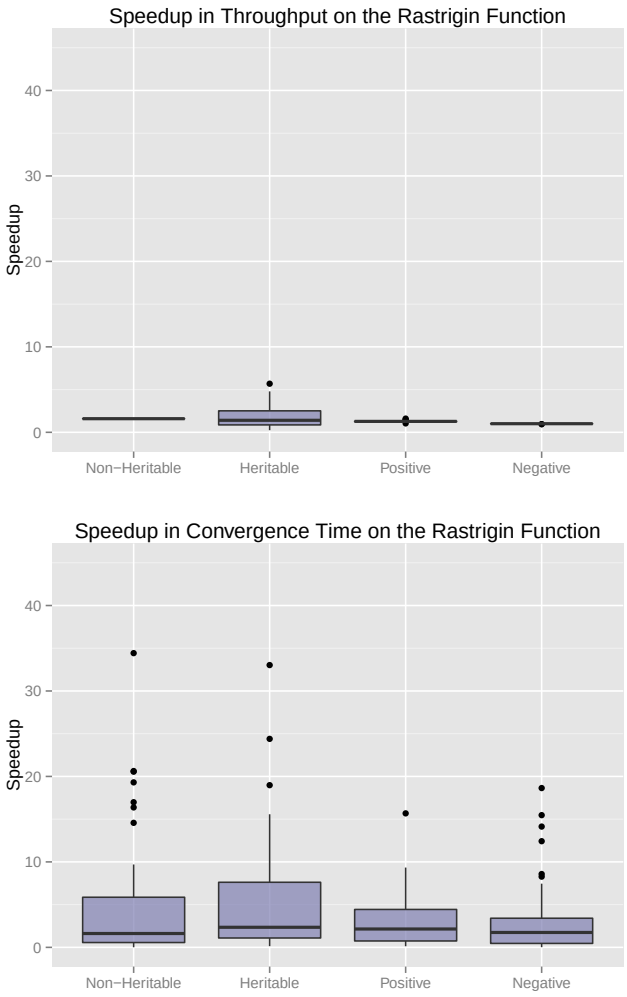

Figure 10: Speedup in throughput (Top) and convergence time (Bottom) of the asynchronous EA on the Rastrigin function.

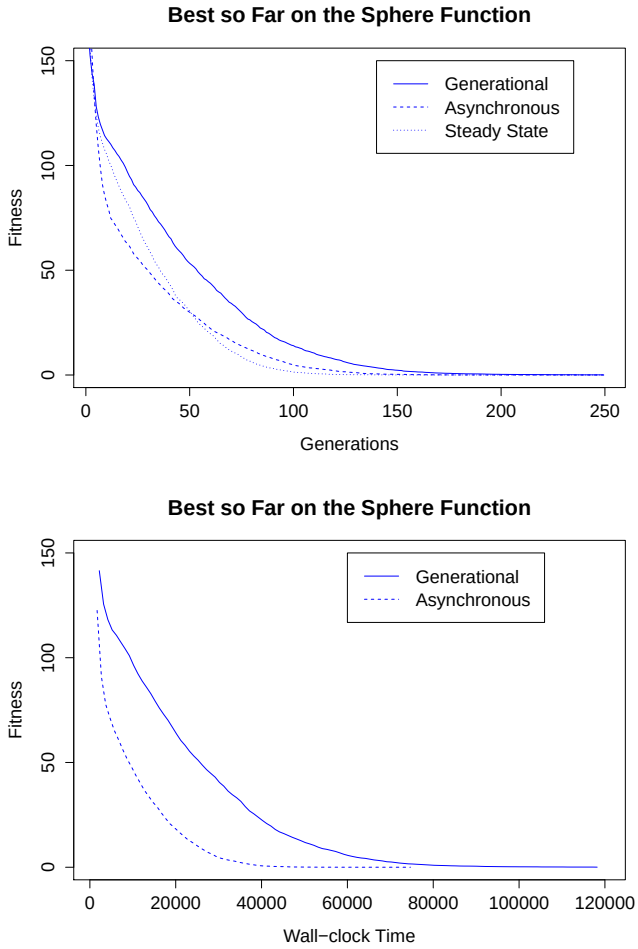

Figure 11: On the sphere function, the simple asynchronous EA consistently takes fewer fitness evaluations on average to reach the optimum (Top). This contributes to its ability to find the solution in less wall-clock time than the generational EA (Bottom).

are known to be greedier than (μ, λ)-style EAs in general, and the simple asynchronous EA is no exception. In all four scenarios on both the sphere and Rastrigin functions, the asynchronous EA converges in fewer fitness evaluations than the generational EA, not just fewer seconds. Part of the true speedup in Figures 9 and 10, then, is attributable to an increase in throughput (working "faster"), but a major component is due to the fact that the asynchronous EA takes a different search trajectory than the generational EA. Because the asynchronous EA requires fewer fitness evaluations to reach the optimum, in these cases we can say it works both faster and "smarter."

4.2.2 *Hölder Table Function*

Both algorithms frequently fail to converge to a global minimum on the Hölder table function. Since many of our runs fail to converge, we cannot fully describe the run-length distribution, which we used to compute the distribution of speedups on the other objectives. Instead, we characterize just part of the run-length distribution by measuring the *success ratio* of each algorithm after a fixed amount of resources have been expended (such as wall-clock time). We define the success ratio as the fraction of 50 independent runs that achieved a fitness of less than $\eta = 2$. Cf. [3] for a discussion of run-length distributions, success ratios and related measures.

In all four scenarios, the asynchronous EA is able to find a global optimum on the Hölder table function much more frequently than the generational EA does. When both algorithms do find a solution, the asynchronous EA finds it at least as quickly as the generational EA does. This can be seen for the heritable and negative scenarios in Figure 12 (the results for the other two scenarios are qualitatively similar).

On all three test functions, then, we have found that the asynchronous EA does not exhibit any particularly adverse performance. To the contrary, it does a good job of delivering the promised speedup in time-to-convergence on the two more tractable problems, and it allows us a much smaller risk of premature convergence on the more difficult Hölder table problem. These results are, of course, problem-dependent, and performance may vary on other objective functions. What this preliminary study has confirmed is that, on a few simple functions, the idiosyncrasies of the asynchronous EA's behavior did not undermine it's ability to improve EA performance. We have learned something about where asynchronous EAs can succeed, even if we do not yet understand how they might fail.

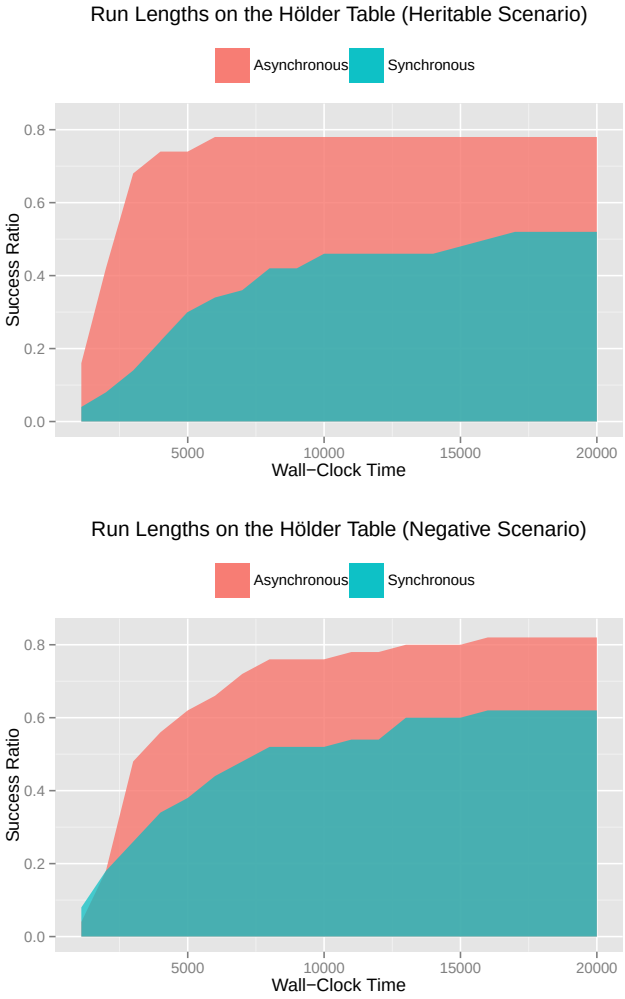

Figure 12: The asynchronous EA is able to find a global optimum on the Hölder table function more often than the generational EA.

5. DISCUSSION

The potential benefits of asynchronous master-slave EAs were first recognized at least twenty years ago [32]. The growing prevalence of evolutionary computation applications that involve resource-intensive simulations, however, pushes evaluation time to a higher position on the list of pertinent concerns in the EA community than it was in the 1990's. This makes the throughput-enhancing advantages of asynchronous master-slave EAs especially relevant.

In this paper, we have made a number of observations that we hope can save practitioners some confusion as they turn to asynchrony to improve their EA performance. Using the assumption of uniformly distributed evaluation-time

variance as an example, we have quantitatively explained how the amount of throughput you gain by choosing a simple asynchronous EA over a generational one depends on the number of slave processors, the size of the population, and the variance of the eval-time distribution.

Throughput improvement merely quantifies how much "faster" the algorithm is working, though, not other implications of its evolutionary behavior, such as how much "smarter" (or not) it is at traversing the search space. We were concerned that asynchronous EAs may have a pernicious bias toward fast-evaluating individuals. The evidence we have presented, however, has ruled out evaluation-time bias that is independent of selection and reproductive variation, and further suggests that evaluation-time bias does not affect the EA's ability to exploit a basin of attraction.

We have clarified the distinction between non-heritable and heritable components of evaluation-time variance, and demonstrated that the latter has different implications on performance depending on whether it is negatively correlated with fitness, positively correlated with fitness, or independent. We also found that when individual evaluation times have a deterministic negative correlation with fitness, the variance in evaluation time quickly drops off, and the asynchronous EA ceases to provide an increase in throughput over the generational competition. Finally, we have emphasized the distinction between throughput improvement and true speedup, and in our experiments we found that the true speedup was often far greater than what the reduction in idle time can account for by itself.

5.1 Threats to Validity

This work isolates several interesting properties of asynchronous EA behavior, but was based entirely on simplifying assumptions about evaluation-time variance, its heritability and its relation to fitness. In our experience with tuning the parameters of spiking neural network simulations, it is not uncommon to observe evaluation times that follow a highly skewed or multi-modal distribution, even in just the non-heritable component that arises from changing the simulator's pseudo-random number seed. The relationship between fitness and heritable evaluation time is also much less straightforward than the linear models in Section 4 imply. Furthermore, most of our experiments in this paper used very small population sizes ($n = 10$), assumed a uniform distribution of evaluation times, and were limited to a real vector genetic representation and Gaussian mutation operator. All these are good reasons to question the extent to which our results will generalize to practical applications.

5.2 Future Work

We have ruled out some kinds of evaluation-time bias, but we know from [30] that evaluation-time bias does exist in some asynchronous EA applications. Further work is needed to understand what kinds of problems may exhibit a bias against slow-evaluating parts of the search space.

The model of throughput improvement we presented for the uniform distribution can be extended to produce similar predictions for evaluation-time distributions that are more likely to be encountered in practice. More challenging, the simple asynchronous EA is quite complex from a dynamical systems perspective. The question of whether and when an asynchronous EA might exhibit a selection effect on the evaluation-time trait could benefit from a better theoretical understanding of how evolution and evaluation times interact. We also may need a better understanding of asynchronous EA dynamics before we can competently describe what kinds of functions they are prone to fail on. What would an 'asynchronous-deceptive' function look like?

Our discrimination between the selection lag and the reordering caused by eval-time variance raises the possibility of constructing an *order-preserving* parallel EA, that allows a selection lag but does not allow re-ordering. Such an algorithm could preserve some of the throughput improvement offered by the asynchronous EA while eliminating its evaluation-time bias.

Acknowledgments

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

6. REFERENCES

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

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

[3] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer, 2006.

[4] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

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

[6] 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.

[7] K. A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, MA, 2001.

[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 2008*, pages 1–8. IEEE, 2008.

[10] M. Giacobini, E. Alba, and M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation — GECCO 2003*, volume 2723 of *Lecture Notes in Computer Science*, pages 955–966. Springer Berlin Heidelberg, 2003.

[11] J. Kim. *Hierarchical asynchronous genetic algorithms for parallel/distributed simulation-based optimization.* PhD thesis, The University of Arizona, 1994.

[12] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly. Parallel asynchronous particle swarm optimization. *International Journal for Numerical Methods in Engineering*, 67(4):578–595, 2006.

[13] P. Liu, F. Lau, M. J. Lewis, and C.-l. Wang. A new asynchronous parallel evolutionary algorithm for function optimization. In *Parallel Problem Solving from Nature—PPSN VII*, pages 401–410. Springer, 2002.

[14] S. Luke. *The ECJ Owner's Manual.* `http://cs.gmu.edu/~eclab/projects/ecj/`, 22nd edition, August 2014.

[15] J. Merelo, A. M. Mora, C. M. Fernandes, and A. I. Esparcia-Alcázar. Designing and testing a pool-based evolutionary algorithm. *Natural Computing*, 12(2):149–162, 2013.

[16] L. Mussi, Y. S. Nashed, and S. Cagnoni. Gpu-based asynchronous particle swarm optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1555–1562. ACM, 2011.

[17] M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In *Third International Conference on Knowledge-Based Intelligent Information Engineering Systems*, pages 88–92. IEEE, 1999.

[18] J. Olenšek, T. Tuma, J. Puhan, and Á. Bűrmen. A new asynchronous parallel global optimization method based on simulated annealing and differential evolution. *Applied Soft Computing*, 11(1):1481–1489, 2011.

[19] M. Parker and G. B. Parker. Using a queue genetic algorithm to evolve xpilot control strategies on a distributed system. In *IEEE Congress on Evolutionary Computation (CEC) 2006*, pages 1202–1207. IEEE, 2006.

[20] K. Rasheed and B. D. Davison. Effect of global parallelism on the behavior of a steady state genetic algorithm for design optimization. In *IEEE Congress on Evolutionary Computation (CEC) 1999*, volume 1. IEEE, 1999.

[21] T. P. Runarsson. An asynchronous parallel evolution strategy. *International Journal of Computational Intelligence and Applications*, 3(04):381–394, 2003.

[22] J. Sarma and K. A. De Jong. Generation gaps revisited. *Foundations of Genetic Algorithms 1993*, 1993.

[23] J. Sarma and K. A. De Jong. Generation gap methods. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Evolutionary Computation I*, pages 205–211. CRC Press, 2000.

[24] T. J. Stanley and T. N. Mudge. A parallel genetic algorithm for multiobjective microprocessor design. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA)*, pages 597–604, San Francisco, CA, 1995. Morgan Kaufmann.

[25] T. Stützle. Parallelization strategies for ant colony optimization. In *Parallel Problem Solving from Nature—PPSN V*, pages 722–731. Springer, 1998.

[26] E.-G. Talbi and H. Meunier. Hierarchical parallel approach for gsm mobile network design. *Journal of Parallel and Distributed Computing*, 66(2):274–290, 2006.

[27] G. Venter and J. Sobieszczanski-Sobieski. Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, 3(3):123–137, 2006.

[28] J. Wakunda and A. Zell. Median-selection for parallel steady-state evolution strategies. In *Parallel Problem Solving from Nature—PPSN VI*, pages 405–414. Springer, 2000.

[29] 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.

[30] 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.

[31] 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.

[32] B. P. Zeigler and J. Kim. Asynchronous genetic algorithms on parallel computers. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 660. Morgan Kaufmann Publishers Inc., 1993.

APPENDIX

A. EXPECTED MAXIMUM OF UNIFORM SAMPLES

Let $P = \{Y_1, Y_2, \ldots, Y_n\}$ be a set of random variables, with the Y_i's i.i.d. from some distribution. Then, from the theory of order statistics, the probability that the maximum value in P obtains a given value is characterized by the cumulative distribution

$$p(\max[P] \leq x) = \prod_{i=1}^{n} p(Y_i \leq x). \tag{21}$$

Differentiating both sides yields the p.d.f.

$$p(\max[P] = x) = \frac{\mathrm{d}}{\mathrm{d}x} \prod_{i=1}^{n} p(Y_i \leq x). \tag{22}$$

We can combine this definition with integration by parts to express the expected value of the maximum value as a

function of the c.d.f. of the Y_i's:

$$\mathbb{E}\left[\max[P]\right] = \int_{-\infty}^{\infty} x \cdot p(\max[P] = x)\mathrm{d}x \tag{23}$$

$$= \int_{-\infty}^{\infty} x \frac{\mathrm{d}}{\mathrm{d}x} \prod_{i=1}^{n} p(Y_i \leq x)\mathrm{d}x \tag{24}$$

$$= \left[x \prod_{i=1}^{n} p(Y_i \leq x) - \int \prod_{i=1}^{n} p(Y_i \leq x)\mathrm{d}x \right|_{-\infty}^{\infty} \tag{25}$$

This expression cannot be solved in closed form for most distributions. The c.d.f. of a uniform distribution over the interval $[a, b]$, however, is given by

$$p(Y_i \leq x) = \int_a^x \frac{1}{b-a}\mathrm{d}y = \frac{x-a}{b-a}. \tag{26}$$

Substituting into Equation 25, we have

$$\mathbb{E}\left[\max[P]\right] = \left[x \left(\frac{x-a}{b-a}\right)^n - \int \left(\frac{x-a}{b-a}\right)^n \mathrm{d}x \right|_a^b \tag{27}$$

$$= b - \frac{1}{n+1}(b-a). \tag{28}$$