Evaluation Time Bias in Asynchronous Evolutionary Algorithms: A Replication Study and a Novel Mitigation Strategy

Joshua Karns
josh@karns.dev
Rochester Institute of Technology
Rochester, New York, USA

Travis Desell
tjdvse@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Abstract

Evolutionary Algorithms (EAs) are a flexible and powerful search technique that are frequently applied to a wide variety of problems. Much of their power comes from their ease of parallelization, lending themselves well to a master-worker parallelization scheme. When a synchronous (μ, λ)-style EA is parallelized and genome evaluation time is not constant, worker processors may spend a significant amount of time idle waiting for other genome evaluations to complete. (μ, λ)-style EAs with a steady-state population are frequently employed to avoid this idle time. There is an existing body of work that suggests that, while this does reduce idle processor time, it may not lead to better solutions because of evaluation time bias. In this work, results from a paper that demonstrate this experimentally are fully reproduced from scratch. During this replication, the roles crossover and population initialization play in this bias were uncovered. This is evaluated experimentally and motivates a mitigation strategy, which is compared to other mitigation strategies and is found to perform about as well as other strategies, without modifying anything other than the way the population is initialized. Moreover, a new open source library was developed in order to facilitate these experiments and further investigations.

CCS Concepts

• **Theory of computation → Parallel algorithms**; **Evolutionary algorithms**.

Keywords

Evolutionary Algorithms, Parallel Algorithms, Asynchronous Algorithms

ACM Reference Format:
Joshua Karns and Travis Desell. 2025. Evaluation Time Bias in Asynchronous Evolutionary Algorithms: A Replication Study and a Novel Mitigation Strategy. In *Genetic and Evolutionary Computation Conference (GECCO '25), July 14–18, 2025, Málaga, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3712256.3726458

GECCO '25, Málaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726458

1 Introduction

Evolutionary Algorithms (EA) are a flexible class of powerful search techniques that have been successfully applied to practical problems from a wide variety of domains. This flexibility also extends to their ability to take advantage of available hardware: EAs that are dominated by genome evaluation times can be parallelized trivially [2]. Parallelization can be done synchronously or asynchronously. In a Synchronous Parallel EA (SPEA), a new generation can only be generated after every member of the current generation has been evaluated. The natural consequence of this is that a SPEA must wait until the genome with the longest evaluation time finishes before the next generation can be created, and in the case of heterogeneous evaluation times this can lead to significant idle time and therefore a drop in efficiency. For this reason the orthodox wisdom is that an Asynchronous Parallel EA (APEA) should instead be employed when faced with variable genome evaluation times as such algorithms can effectively utilize a massive number of processors with minimal idle time.

APEAs higher efficiency for this class of problem compared to SPEAs make them an appealing choice however there can be unintended side effects – APEAs have been shown to prematurely converge to local minima that are relatively fast to evaluate [7, 10, 12, 14, 15]. It is not difficult to intuit this phenomena in mechanistic terms: a fast-to-evaluate genome and its offspring will have many more opportunities to proliferate than a comparatively slow genome. Over time, this may make it difficult for slow-to-evaluate solutions to compete and lead to significant wasted computation on local minima that are fast-to-evaluate.

The problem of evaluation time bias has been observed since at least 2011, when Yagoubi *et al.* presented a work in which the authors compare a SPEA with an APEA on a set of benchmark problems [16]. The authors found that artificially increasing the evaluation time in desirable regions of the search space actually improved the quality of the solutions found. They speculate that this is because convergence is slowed down thus allowing for more exploration of the search space. In a follow up work, Yagoubi *et al.* observe the negative impact of evaluation time bias – experiments with artificially heterogeneous evaluation times demonstrate that the APEAs avoid costly regions of the Pareto Front due to their high computational cost [15].

Rivers *et al.* [10] identify evaluation time bias as a form of parsimony pressure that is intrinsic to asynchronous genetic algorithms. Parsimony pressure is a technique used as a form of regularization in genetic programming, ensuring solutions to not get too large [9]. Normally this is done explicitly, however the observed tendency of some APEAs to prefer individuals with shorter evaluation time can be interpreted as a form of parsimony pressure.

Scott and De Jong [12–14] began a series of investigations into the properties of APEAs by trying to answer several research questions, of particular interest to this work is the following: do APEAs inherently favor faster evaluating individuals? The authors run a straightforward experiment where an APEA was run on a flat fitness landscape, where genomes simply contain a value representing their evaluation time. New genomes are always inserted into the population as their fitness values are all identical, thus any variation in the aggregate genotype must be attributable to an implicit bias innate to the asynchrony of the EA. The authors unexpectedly do not find any bias on a flat fitness landscape, which hints at something they experimentally demonstrate in a subsequent work: evaluation time bias is a systematic process that emerges from the interplay of the fitness landscape and evaluation-time landscape.

This phenomenon has been demonstrated experimentally going at least as far back as 2015 with a work by Scott and De Jong [12]. The authors employ a discrete event simulation to efficiently model the behavior of APEAs with a variety of execution time topologies, running many thousands of experiments to approximate the probability of premature convergence for a given fitness function and corresponding execution time function. They find that the coincidence of local extrema in the fitness landscape and in the evaluation time landscape (*i.e.* how long it takes for a particular genome to be evaluated) produce a bias towards or against the global minimum, depending on whether the surrounding region is relatively slow or fast to evaluate.

APEAs reduce idle time and can reduce the wall clock time required for convergence, and this is the motivation for a few works that aim to address this bias. The method proposed by Harada [7] in 2022 only requires adding a single additional parameter to genomes to serve as a counter. The purpose of the counter is to ensure the fitness landscape is searched in a more uniform manner, free of evaluation time bias. Selected parent genomes are always those who have the smallest counters, and during reproduction their counters are incremented, and the child will have a counter equal to the mean of its parent's counters. This method was used with an asynchronous parallel version of NSGA-III [3] on a variety of benchmark functions and reduced the effect of the evaluation time bias, ultimately leading to an overall faster runtime. Scott *et al.* present an alternative method meant to avoid evaluation time bias with the goal of avoiding excess computation [11]. The method is quite simple: when selecting genomes for reproduction, genomes that are still evaluating are eligible thus allowing slow-to-evaluate genomes to propagate their genetic information earlier.

In this work, the methodology and experiments from Scott and De Jong's [12] 2015 paper demonstrating the existence of evaluation time bias are replicated based only on the materials presented in the original work. Their work did not include any source code, therefore a novel discrete event simulation framework was created specifically for this work. This new open source[1] framework is written in modern C++ and was built to be extensible and fast, allowing millions of APEAs runs to be simulated in mere seconds when multithreaded. This new framework has been designed to be easily extensible, allowing for different fitness functions, evaluation time landscapes, genetic operators and even different EAs to be plugged in with minimal effort.

Scott and De Jong's original findings are largely validated, though one of their figures could not be replicated. In the process of replicating their work, these results are found to be sensitive to two details that are not given much weight in the original work: (1) the inclusion of crossover and (2) genome initialization. We find that two-point crossover as described in the original work exacerbates the bias and hinders exploration into slower to evaluate regions of the search space. Genome initialization is also found to have a controlling effect on convergence behavior, which is particularly strong when crossover is disabled. This discovery is leveraged in a second, novel set of experiments where the population initialization process is modified in order to begin asynchronous evolution with a more diverse population. This method is experimentally demonstrated to reduce evaluation time bias, and is compared to the existing methods proposed by Harada [7] and Scott *et al.* [11]. Unlike those methods however, this approach does not modify the manner in which genomes are selected for reproduction and only modifies the initialization procedure.

The contributions of this work are re-enumerated as follows:

(1) A fast, extensible, and open-source discrete event simulation framework purpose built for this work with modern C++ is presented.
(2) The experimental methodology from Scott and De Jong's paper demonstrating the existence of evaluation time bias is fully replicated and partially validates their original findings in Section 3. One of their figures with confounding implications could not successfully be replicated, suggesting a methodological issue in the original paper.
(3) In the process of replicating the original study, it was discovered that crossover and genome initialization play significant role in this bias. This is experimentally demonstrated in Section 4.
(4) The findings from Section 4 motivate a novel evaluation time bias mitigation technique, only requiring that population initialization be modified slightly. Other methods to reduce evaluation time bias are tested and compared to the one proposed in this work in Section 5.

2 Asynchronous Parallel Evolutionary Algorithms

Asynchronous Parallel Evolutionary Algorithms (APEAs) are usually contrasted with Synchronous Parallel Evolutionary Algorithms (SPEAs) as they address one of their primary drawbacks. (μ, λ)-style EAs can be parallelized using a master-worker model – each generation of λ child genomes are evaluated in parallel on the worker processors [1]. The master will collect evaluated children and wait for all of them to complete during the synchronization step, which is where potential idle time is introduced.

Work by Zăvoianu *et al.* [17] provides supporting evidence for this concern and conclude that an APEA is a better choice than an SPEA in the case of high evaluation time variability. Their methods however use randomly generated evaluation times rather than relying on a consistent topology, meaning any evaluation time biases may be masked.

[1]The source code is available on GitHub: https://github.com/jkarns275/evo-sim

Algorithm 1 The Simple Asynchronous Evolutionary Algorithm Modeled in this Work

```
function INITIALIZE_POPULATION(pop_size)
    population ← ∅
    while |population| < pop_size do
        if ¬ node_available() then
            wait()
        end if
        while node_available() do
            send(random_genome())
        end while
        population ← population∪ next_evaluated_genome()
    end while
    send(reproduce(population))
    return population
end function
function SIMPLE_APEA(pop_size, steps)
    initialize_population(pop_size)
    for i ← 0 to steps do
        evaluated_genome ← next_evaluated_genome()
        opponent ← select_one(population)
        if better_than(evaluated_genome, opponent) then
            population ← (population − opponent)
            population ← population ∪ evaluated_genome
        end if
        send(reproduce(population))
    end for
end function
```

ASPEAs avoid this potentially costly synchronization step by combining a $(\mu, 1)$-evolution strategy with a steady state population, thus worker processors can immediately begin evaluating a new child genome (ignoring EA-related overhead), while genomes are continuously integrated into the population.

The "Simple Asynchronous EA" as described by Scott and De Jong [12–14] in several of their works will be the model of an APEA used in this work. Algorithm 1 provides pseudocode for this model, and is largely copied verbatim.

The algorithm begins by asynchronously initializing the population, where the master sends random genomes to workers for evaluation and places evaluated individuals into the population. In Section 4, the effect of this initialization strategy is explored – asynchronously initializing the population may start the search off with an already biased population, exacerbating the problem.

Next, the algorithm evaluates *steps* genomes asynchronously. Worker processors will evaluate genomes and send the results to the master. When an evaluated genome is received by the master, it will be inserted into the population, and a new individual will be generated via reproduction and sent to the idle worker processor. Assuming the overhead of selection, reproduction, and communication are proportionally low to genome evaluation, this algorithm will minimize idle time of its processors.

2.1 Discrete Event Simulation Framework

Running actual evolutionary algorithms in order to understand their behaviors can be quite costly, particularly when the experiments involve a fitness function that has high evaluation time variability. For this reason, Scott and De Jong [12–14] implement Algorithm 1 in the form of a discrete event simulation in their series of articles related to evaluation time bias.

For this work, a novel discrete event simulation framework was implemented – a link to the source can be found in the footnotes. The framework is written in C++ and is written to execute experiments in parallel – each experiment requires that many thousands of independent simulations are run, which is trivial to parallelize using multithreading. Implementing the simple APEA outlined in Algorithm 1 hinges primarily on a min-priority queue to allow fast transitioning between each simulated genome evaluation, and requires that an evaluation time function $t(x)$ is specified. This evaluation time function will share the same domain as the fitness function and will be used to compute the simulated finish time of each genome (this will be the simulation's current time added to $t(x)$). This finish time is what the priority queue uses to sort its values, making it fast to instantly jump to the next completed genome evaluation.

3 Replication Study

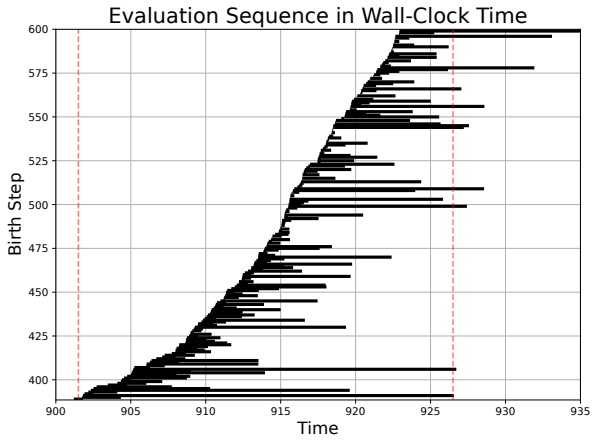

Figure 1: Genomes evaluation time spans plotted by their birth step. The horizontal bars plat the start and end time of genome evaluation.

In this Section, Scott and De Jong's [12] work is replicated and the two hypotheses they present in their 2015 work are affirmed, though one of their figures with confounding implications could not be replicated successfully. The four figures included in their paper are replicated, although Figure 4 shows different results than were found in the original work.

They begin their work with a figure visualizing the manifestation of evaluation time bias in the form of a sequence of fitness evaluation durations, replicated here in Figure 1. The key is that in the time it takes to evaluate one slow genome (shown with the vertical red lines) many generations of advancement for the fast to evaluate genomes take place.

3.1 Methodology

The authors [12] propose four hypotheses pertaining to evaluation time bias, and they are made with respect to a simple function which they then test with their discrete event simulation framework. The function represents a simple topology with two basins of attraction each of which has a corresponding parameter $A, B \in \mathbb{R}$ which roughly equate to depths. The function is as follows for an N-dimensional vector x [12]:

$$\begin{aligned} f_{A,B}(x) = \max(|A|, |B|) \\ - A \exp(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} (x_i - 2\sigma)^2) \\ - B \exp(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} (x_i + 2\sigma)^2) \end{aligned} \tag{1}$$

This function will be referred to as the $\mathrm{f}_{A,B}$ in this work. The two extrema of the function will be located at $(2\sigma, \ldots, 2\sigma)$ and $(-2\sigma, \ldots, -2\sigma)$ for some value of σ (not specified in the original work), where their depths are given by A and B respectively. These extrema will be referred to as A_0 and B_0. The function $\mathrm{f}_{A,B}$ will always have a constant evaluation time, therefore to modify the behavior of the discrete event simulation a time function $t(x)$ must be specified. With this in mind, they make the following hypotheses regarding the behavior of APEAs [12]:

Hypothesis 1a (H1a). *On the fitness function $f_{A,B}$ with $A = B$, an APEA will converge to either optimum with equal probability if individual evaluation times are constant ($t(x) = 1$) or directly proportional to fitness ($t(x) \propto f_{A,B}(x)$).*

Hypothesis 1b (H1b). *If solutions in one basin have shorter evaluation times than in the other basin (e.g. $t(x) \propto f_{A,-B}(x)$), the APEA will be biased toward the fast-evaluating basin.*

Convergence to A_0 can be thought of as a random variable R which follows a binomial distribution for some proportion parameter r, *i.e.* $P(R = 1) = r$. Each simulation on $\mathrm{f}_{A,B}$ can be thought of as drawing a sample from this distribution, thus the true value of r can be directly estimated by running numerous experiments and computing the proportion of them which converge to A_0. The authors provide the following convergence criteria for each run given the best found genome x, yielding random variable R_i [12]:

$$R_i = \begin{cases} 1 & \text{if} \quad ||x - A_0|| < \tau \\ 0 & \text{if} \quad ||x - B_0|| < \tau \end{cases}$$

For some small value of τ, although an exact τ value was not provided. This function is undefined for all x that are not within the n-sphere with radius τ centered around either A_0 or B_0. For this reason, an alternative convergence criteria is used:

$$R_i = \begin{cases} 1 & \text{if} \quad ||x - A_0|| < ||x - B_0|| \\ 0 & \text{else} \end{cases}$$

A sample of $P(R = 1)$ is simply a run of the APEA simulation, and using the definition of R_i (the ith sample from $P(R = 1)$) above the proportion parameter r can be estimated with $\frac{1}{K} \sum_{i=1}^{K} R_i$ for some large number of experiments K. Experiments done in this manner also have a 95% confidence interval computed using the Wilson method in the original work and the experiments done in this work.

In order to test Hypotheses 1a and 1b, the authors approximate $P(R = 1; t(x))$ (probability that $R = 1$ with time function $t(x)$) with 5,000 simulations with fitness function $\mathrm{f}_{A,B}$ with $A = B$ for three different time functions $t(x)$:

(1) $P(R = 1; 1)$, constant evaluation time.
(2) $P(R = 1; \mathrm{f}_{A,B})$, evaluation time that is directly correlated with fitness.
(3) $P(R = 1; \mathrm{f}_{A,-B})$, evaluation time that is slower for one of the basins – B_0 in this case.

In the instance of the "Simple Asynchronous EA" [12] used in their study, child genomes are produced using two-point crossover, and Gaussian mutation is applied to each gene with probability $\frac{1}{N}$, modified genes are offset by a value sampled from $\mathcal{N}(0.0, 0.25)$. Genomes are inserted into the population using random survival selection, and parent genomes are selected using stochastic tournament selection of size 2. In each experiment, the population size is 10 and the number of simulated worker processors is 10 (*i.e.* the size of the priority queue). The dimensionality of the genomes is not specified, although it can be inferred it must be greater than 2 to allow for two-point crossover and less than 30 at which point the algorithm fails to converge after the specified 1,000 iterations. Genomes parameters are all bound within the range of [-10, 10].

The replication experiments copy all algorithms, methods, and experimental parameters exactly as described above based on the original work, unless otherwise specified. An additional experiment is run in this work to compute $P(R = 1; \mathrm{f}_{-A,B})$. As no dimensionality was specified, 10 is selected is it fits the criteria specified in the paper of being small enough so that "all [K] runs converged" in 1,000 steps. In this work the number of simulations run, K, is raised to 1,000,000 rather than using 5,000 – this makes for a reduced margin of error and can be done in a timely fashion with our newly introduced discrete event simulation framework. The number of simulations is sufficient such that the 95% confidence interval computed for each experiment is quite small in most figures, consistently under 0.1%. No values for A and B were specified in the original work, just their ratios, so 10 is selected for both A and B in this first set of experiments. The manner in which genomes are randomly generated during the initialization phase of the simulation is not specified by Scott and De Jong, however experimentation pointed towards genomes being initialized with Gaussian noise from $\mathcal{N}(0, 1)$ so it is used here.

Regardless of whether experiments confirm or deny Hypotheses 1a and 1b, by themselves the implications are not particularly serious as they deal with a fitness landscape that contains two equal minima. Scott and De Jong [12] propose the following hypotheses regarding a fitness landscape with two local minima and a unique global minimum:

Hypothesis 2a (H2a). *Let $A, B \in \mathbb{R}$ with $A < B$. An asynchronous EA using the fitness function $f_{A,B}$ will prematurely converge to local minimum A_0 using a constant time function with probability $P(R = 1; 1)$ (read as "the probability that R = 1 given $t(x) = 1$") equal to $P(R = 1; f_{A,B})$.*

Hypothesis 2b (H2b). *If A_0 has faster evaluation than basin B (e.g. $t(x) = f_{A,-B}(x)$), an asynchronous EA will converge prematurely*

to A_0 with probability $P(R = 1; f_{A,-B})$ such that $P(R = 1; f_{A,-B}) > P(R = 1; 1)$.

Hypothesis 2c (H2c). *If basin A has slower evaluation than basin B ($t(x) = f_{-A,B}(x)$), an asynchronous EA will converge prematurely to A with probability $P(R = 1; f_{-A,B})$ such that $P(R = 1; f_{-A,B}) < P(R = 1; 1)$.*

The same set of fitness functions are again tested using the previously described methodology with A = 10 and B = 1.5A = 15. Again, only the ratio for A and B was supplied in the original work rather than specific values.

The last set of experiments Scott and De Jong describe is one where simulations are run with a flat fitness landscape (*i.e.* $f(x) = 1$) and a heterogeneous time function – specifically $t(x) = \mathrm{f}_{A,-B}(x)$. They run 5,000 simulations for 100 steps using a 2-dimensional genome (two-point crossover must be disabled in this case). At the end of each simulation, a random member of the population is selected and logged. These genomes are placed onto a scatter plot, and their results essentially resemble a bimodal distribution, with both A_0 and B_0 exhibiting attractive forces. 5,000 simulations are run for 100 steps to generate Figure 4.

A logical deduction can be made that this experiment should yield random noise equal to the distribution used to initialize the population, as no child genome will ever be better than its parent, and therefore should never be inserted according to our survival selection (illustrated in Algorithm 1).

All experiments in this work were run on a MacBook Pro with an Apple M3 Max CPU (10 performance cores and 4 efficiency cores) and 36 GB of RAM, running MacOS Sequoia 15.1.1. The 1,000,000 simulations run per experiment were run in parallel on 10 separate threads. Note that no hardware specifications were given in the original work.

3.2 Results

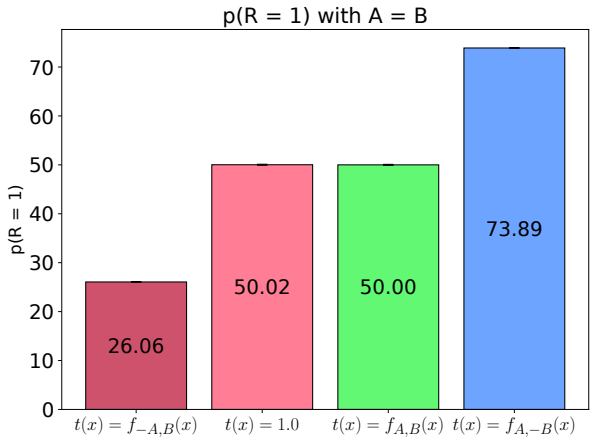

Figure 2: The probability of converging to A_0 (*i.e.* $P(R = 1)$) when $A = B$ with different time functions.

The experiments testing Hypotheses 1a and 1b are shown in Figure 2. The figure shows $P(R = 1; 1)$ and $P(R = 1; \mathrm{f}_{A,B})$ to both be equal to 50%, validating Hypothesis 1a. Hypothesis 1b is validated as well as both $P(R = 1; 1)$ and $P(R = 1; \mathrm{f}_{A,B})$ are less than $P(R = 1; \mathrm{f}_{A,-B})$. These results correspond perfectly with those presented by Scott and De Jong – both in their implications and exact values (though the confidence intervals in the original work were larger, about ±2.5%) [12]. An additional time function was tested for this work as well, providing additional evidence for Hypothesis 1b as $P(R = 1; \mathrm{f}_{-A,B})$ is less than both $P(R = 1; 1)$ and $P(R = 1; \mathrm{f}_{A,B})$.

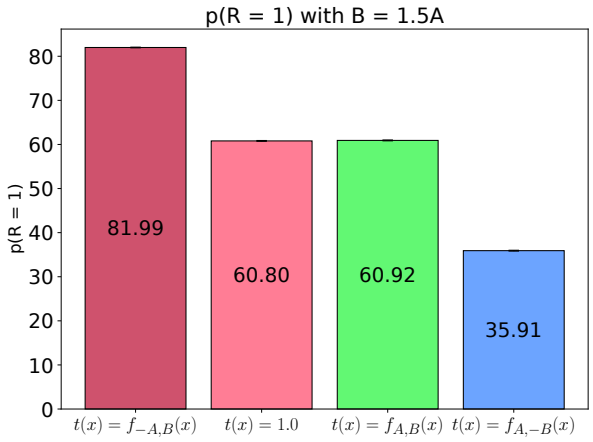

Figure 3: The probability of converging to A_0 (*i.e.* $P(R = 1)$) when $1.5A = B$ with different time functions.

Figure 3 shows experiments run to test Hypotheses 2a, 2b, and 2c. The value $P(R = 1; 1)$ hypothesized in 2a is shown to be about 38%. One might also deduce from Hypothesis 1a that $P(R = 1; \mathrm{f}_{A,B})$ is also around 38%, which is roughly true, although it is slightly smaller.

The experimentally derived value for $P(R = 1; \mathrm{f}_{A,-B})$ is shown to be approximately 60% and greater than $P(R = 1; 1)$, validating Hypothesis 2b. Similarly, the computed value for $P(R = 1; \mathrm{f}_{-A,B})$ is shown to be approximately 17% and less than $P(R = 1; 1)$, validating Hypothesis 2c.

While the conclusions made regarding Hypotheses 2a, 2b, and 2c match those made by Scott and De Jong, their experimental results using a flat fitness landscape could not be replicated. In the methodology (Section 3.1) it is logically deduced that no genomes should ever be inserted according to the survival selection criteria in Algorithm 1, thus the scatter plot should simply be spherical Gaussian noise. The attempt at replication is shown in Figure 4, which appears to show Gaussian noise.

All care to accurately copy the methodology from the original work was given, however there may be an important detail about the experiment that was missing from the original work thus preventing accurate replication. Still, the original findings were quite confounding: the points were shown to be drawn to both the fast and slow basin of attraction [12]. The results obtained in this work however are still confounding and suggest the mechanistic intuition behind evaluation time bias does not tell the whole story, and that selection plays a key role in the manifestation of the bias.

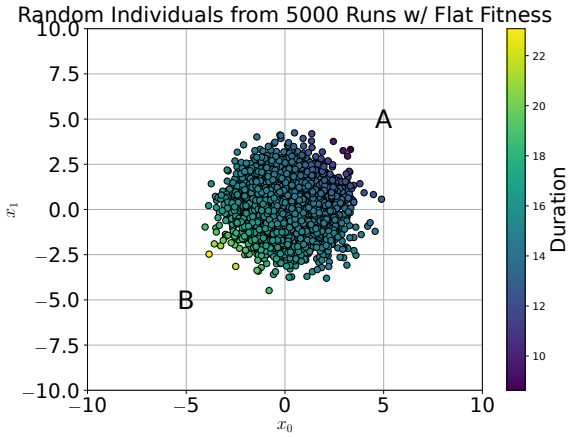

Figure 4: A scatter plot of random genomes at the end of 5,000 runs of the simulated APEA.

4 Role of Crossover and Population Initialization

In the previous Section, replicated results were presented that provide some basic insight into how APEAs with heterogeneous evaluation times behave with a simple example function. Scott and De Jong's original work did not specify the exact method by which genomes created during the initialization phase of their Algorithm (the "random_genome" function in Algorithm 1)[12]. While replicating their work, the finding that genome initialization plays a significant role in the magnitude of the measured bias was discovered. Crossover also appears to play a significant role in exacerbating evaluation time bias, contrary to the assumption made by the original authors in another related work [14].

In this section, a set of experiments are run to demonstrate the role of crossover and population initialization in evaluation time bias. These findings motivate a mitigation technique laid out in Section 5, which is compared to various mitigation techniques that exist in literature.

4.1 Methodology

In order to demonstrate the relationship between initialization, distribution and evaluation time bias, the distribution used to initially generate genomes is made increasingly diffuse by increasing the variance of the 0-centered Gaussian distribution used to generate the genomes. In other words, new genomes are generated by taking N-samples from $\mathcal{N}(0, \sigma^2)$ for an N dimensional genome:

$$\mathbf{G} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix}, \quad g_i \sim \mathcal{N}(0, \sigma^2) \quad \text{for } i = 1, 2, \ldots, N$$

The same algorithm and experimental parameters are used as in Section 3.1, except the initialization method used when generating the initial population. A grid search is done on the variance starting at 0 up through 14 – note that a variance of 0 will simply yield a population of genomes all located at the origin. One experiment is done with crossover enabled, and another done with crossover disabled. The time function for these experiments is $t(x) = \mathrm{f}_{A,-B}(x)$, and is done for both pairs of (A, B) values $(10, 10)$ and $(10, 15)$.

4.2 Results

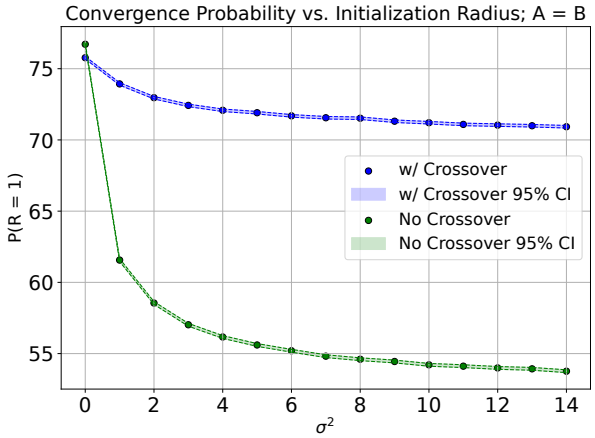

Figure 5: A plot summarizing the relationship between variance (σ^2) and the probability of converging to A_0 $P(R = 1)$ when A = B and $t(x) = \mathbf{f}_{A,-B}$.

Figure 5 and Figure 6 summarize the results. With $A = B$ in Figure 5 the results for $\sigma = 1$ match those in Section 3.2, with a higher convergence rate to A_0 than B_0 stemming from the evaluation time function. It can be observed that this bias dissipates as σ^2 increases, dropping precipitously in the experiments that neglect to use crossover. This suggests two things: (1) the inclusion of crossover can exacerbate evaluation time bias and (2) initial populations that cover more of the search space are less prone to this bias.

The modulating effect of σ^2 shown in Figure 5 is not seen nearly as dramatically in Figure 6. Crossover does again appear to exacerbate the algorithms preferences for the faster to evaluate basin A_0, as the faulty preference for A_0 is notably lower when crossover is not used. With a σ^2 value of 0 the bias is markedly reduced in both cases – a wider initialization radius can both help the EA avoid evaluation time bias as seen in Figure 5 or exacerbate it as seen here.

The effect is quite subtle in the experiments utilizing crossover, suggesting that the exploitative power of crossover can contribute to evaluation time bias arising. The EA used for this study is simple as is the fitness landscape, making its two-point crossover relatively powerful which may be why the effect is dramatic.

5 Addressing Evaluation Time Bias

In the previous section a relationship between premature convergence to A_0 and the population initialization distribution was uncovered. This suggests that, in some cases, a larger initialization distribution can mitigate evaluation time bias. In most practical

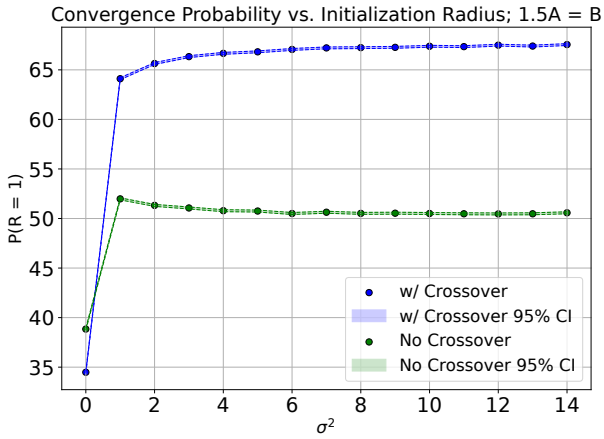

Figure 6: A plot summarizing the relationship between variance (σ^2) and the probability of converging to A_0 $P(R = 1)$ when 1.5A = B and $t(x) = \mathbf{f}_{A,-B}$.

applications, this is not as easy as modifying σ^2 when sampling random noise – APEAs that exhibit this problem often have complex genotypes that may represent an unbounded search space. In such cases, uniform initialization is not possible, however it is possible to initialize the population without inducing evaluation time bias.

Algorithm 1 describes the simple EA being studied in this work, and a key feature of it is that it initializes the population with random generation and asynchronous evaluation – this asynchrony may induce evaluation time bias from the very beginning and hinder a more uniform initialization. For this reason, a new mitigation strategy is proposed: simply evaluating the initial population of genomes synchronously. The proposed changes to the initialization procedure are highlighted in Algorithm 2.

The significance of the proposed method mostly stems from its simplicity – it only requires a small number of genomes to be evaluated synchronously at the beginning of the algorithm whereas other methods to be described later on fundamentally effect the way genomes are selected for reproduction. This method will be referred to as GEN as it modifies the initial generation of the population.

Algorithm 2 Proposed synchronous initialization procedure to APEAs, as opposed to the asynchronous initialization seen in Algorithm 1. The serial genome evaluations could be parallelized.

```
function INITIALIZE_POPULATION(pop_size)
    population ← ∅
        ▷ Synchronously evaluate genomes to fill the population.
    for i ← 0 to pop_size do
        population ← population∪ evaluate(random_genome())
    end for
    np ← get_number_worker_processors()
    for i ← 0 to np do
        send_to_worker(i, reproduce(population))
    end for
     return population
end function
```

Two other mitigation techniques have been proposed in literature, both of which are elegant and require minimal modification to the APEA. First is a method proposed by Harada, in which a counter is added to genomes which counts how many times that genome has passed along its genetic material via crossover [7], referred to as CNT in this work. When a child genome is created, its parents have their counters incremented and then the child's counter is set to the average of the parents counters. These counters are used when selecting genome for reproduction: the genomes with the lowest counters are always selected for reproduction, thus avoiding a situation where only the fast to evaluate regions of the fitness landscape are explored.

The second method is proposed by Scott *et al.* and is called SWEET: "Selection whilE EvaluaTing" [11], and is shortened to SWT in this work. The motivation behind the method is to mitigate excess evaluation of fast to evaluate regions of the fitness landscape and thus avoid excess computation. It is fairly straight forward: the method simply allows genomes to be selected for reproduction before they have finished evaluation. In the case of the simple APEA shown in Algorithm 1, the "reproduce" function must be modified to take an additional parameter representing the set of genomes that are currently being evaluated.

Section 5.1 described the methodology with which these mitigation techniques are tested, and Section 5.2 summarizes the results.

5.1 Methodology

The methodology from Section 3.1 is used but modified to test the three mitigation strategies: (1) SWEET [11] (2) Harada's counting method [7] and (3) the proposed synchronous initial genome generation method described previously.

Each of the of these three strategies as well as a control are tested on the same set of four time functions from Section 3.1, and this is done for 5 pairs of (A, B) value which each increase the disparity between the time function and fitness landscape. The values used for (A, B) are (10, 10), (10, 15), (10, 20), (10, 25), and (10, 30).

5.2 Results

The results from all experiments are summarized in Figure 7. In each subplot, the middle two bars effectively act as a control – demonstrating the behavior of the algorithm with either a constant time function or a time function that corresponds one-to-one with fitness, revealing convergence behavior with no evaluation time bias as per Hypothesis 1a and 2a.

Each mitigation technique in has a clear effect compared to the control experiments, although SWT and GEN appear to increase $P(R = 1; 1)$ and $P(R = 1; \mathrm{f}_{A,B})$ indicating an increase in convergence to local minima A_0, however this may also indicate wasted computation around the fast to evaluate local minima. This holds for every pair of (A, B) except for the first column $(10, 10)$. For CNT, these values are consistently identical to the control row indicating the strategy is effectively neutral in the case where there is no evaluation time bias.

6 Discussion

In this work the work of Scott and De Jong demonstrating the existence of evaluation time bias is fully replicated from scratch and

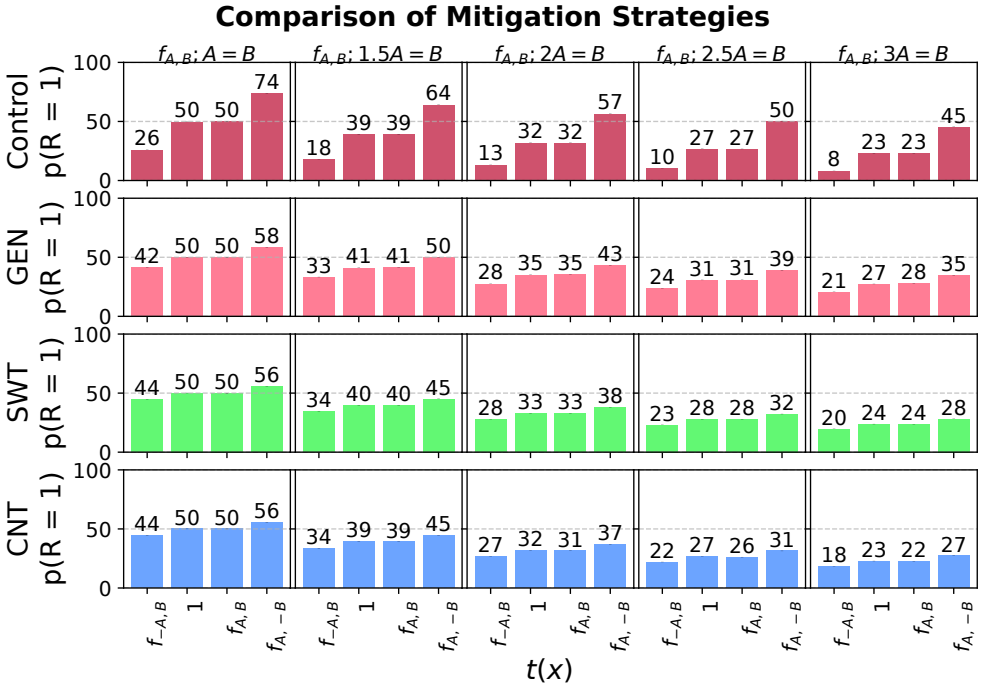

Figure 7: Matrix of bar-plots showing the probability of converging to A_0 under varying conditions. Each row corresponds to a different permutation of mitigation techniques, and each column contains a different pair of A and B values.

experiments demonstrating the key Hypotheses made in the paper are confirmed, though one of the figures could not be replicated [12]. This figure was somewhat confounding however, suggesting there may have been a problem in (1) the experimental setup as described in the paper or (2) a misconfiguration in the experiments run to obtain the data. This work is seminal in the field as it demonstrates in a direct manner the existence of evaluation time bias using a generic and extensible experimental framework.

A novel and open source discrete event simulation framework was purpose built with C++ for this replication experiment, and was further leveraged in two additional sets of experiments. First is a set of experiments which sought to highlight the role crossover and population initialization play in evaluation time bias, and they show that crossover plays an exacerbating role in the bias and population initialization can contextually push the bias in either direction. Second, the findings from the previous section were roughly translated into a mitigation technique: simply evaluating the initial generation synchronously should at the very least mitigate evaluation time bias induced by the asynchronous population initialization used in Algorithm 1. The direct comparison of existing mitigation techniques is, to the best of the authors' knowledge, a novel contribution. Moreover, these experiments serve a further purpose as a replication study for the methods of Harada [7] and Scott *et al.* [11].

The common thread among all of these techniques is they ensure the fitness landscape is searched in a more uniform manner, although the exact means by which they do so are unique. They all achieve somewhat similar performance on the objective function $f_{A,B}(x)$, creating a strong motivation for a future work in which these mitigation techniques are tested directly against one another on a variety of more complex benchmarking functions and potentially several types of APEA (*e.g.* differential evolution, CMA-ES). Our newly provided simulation framework will easily enable this future work, as well as facilitate future research into this phenomenon.

6.1 Limitations

While the head-to-head comparison of these methods is novel and can inform the design of APEAs, this study leaves room for future work in comparing these search strategies on a more complex set of fitness and time landscapes that model real-world problems or be directly applied to such problems. In general, the fitness landscape used in most of the experiments in this work was originally selected by Scott and De Jong to highlight the problem of evaluation time bias [12]. The problem therefore may arise in more complex and non-linear ways than this work may represent. Additionally, the EA used in this work is fairly simple, and findings may be different if more advanced EAs are used, *e.g.*, CMA-ES [5], differential evolution [4], or particle swarm optimization [8], among others. Some groundwork in understanding the evolutionary dynamics surrounding evaluation time bias has been done by Harada in other works, as well as Scott *et al.* [6, 7, 11]. The analysis tools presented in these works can be leveraged to robustly analyze the head-to-head performance of the various evaluation time bias mitigation strategies that exist, and hopefully provide practitioners with an intuition as to which technique is best suited to their problem.

References

[1] Erick Cantu-Paz. 2000. *Efficient and accurate parallel genetic algorithms*. Vol. 1. Springer Science & Business Media.

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

[3] Kalyanmoy Deb and Himanshu Jain. 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation* 18, 4 (2013), 577–601.

[4] Vitaliy Feoktistov. 2006. *Differential evolution*. Springer.

[5] Nikolaus Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[6] Tomohiro Harada. 2019. Mathematical model of asynchronous parallel evolutionary algorithm to analyze influence of evaluation time bias. In *2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. IEEE, 1–8.

[7] Tomohiro Harada. 2022. A Frequency-based Parent Selection for Reducing the Effect of Evaluation Time Bias in Asynchronous Parallel Multi-objective Evolutionary Algorithms. *Natural Computing* (Dec. 2022). doi:10.1007/s11047-022-09940-z arXiv:2107.12053 [cs]

[8] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, Vol. 4. ieee, 1942–1948.

[9] Riccardo Poli and Nicholas Freitag McPhee. 2008. Parsimony pressure made easy. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (Atlanta, GA, USA) *(GECCO '08)*. Association for Computing Machinery, New York, NY, USA, 1267–1274. doi:10.1145/1389095.1389340

[10] Rebecca Rivers, Alex R. Bertels, and Daniel R. Tauritz. 2015. Asynchronous Parallel Evolutionary Algorithms: Leveraging Heterogeneous Fitness Evaluation Times for Scalability and Elitist Parsimony Pressure. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, Madrid Spain, 1429–1430. doi:10.1145/2739482.2764718

[11] Eric O Scott, Mark Coletti, Catherine D Schuman, Bill Kay, Shruti R Kulkarni, Maryam Parsa, Chathika Gunaratne, and Kenneth A De Jong. 2023. Avoiding excess computation in asynchronous evolutionary algorithms. *Expert Systems* 40, 5 (2023), e13100.

[12] Eric O. Scott and Kenneth A. De Jong. 2015. Evaluation-Time Bias in Asynchronous Evolutionary Algorithms. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, Madrid Spain, 1209–1212. doi:10.1145/2739482.2768482

[13] Eric O. Scott and Kenneth A. De Jong. 2015. Understanding Simple Asynchronous Evolutionary Algorithms. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*. ACM, Aberystwyth United Kingdom, 85–98. doi:10.1145/2725494.2725509

[14] Eric O. Scott and Kenneth A. De Jong. 2016. Evaluation-Time Bias in Quasi-Generational and Steady-State Asynchronous Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 845–852. doi:10.1145/2908812.2908934

[15] Mouadh Yagoubi and Marc Schoenauer. 2012. Asynchronous Master/Slave Moeas and Heterogeneous Evaluation Costs. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. ACM, Philadelphia Pennsylvania USA, 1007–1014. doi:10.1145/2330163.2330303

[16] Mouadh Yagoubi, Ludovic Thobois, and Marc Schoenauer. 2011. Asynchronous Evolutionary Multi-Objective Algorithms with Heterogeneous Evaluation Costs. In *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, New Orleans, LA, 21–28. doi:10.1109/CEC.2011.5949593

[17] Alexandru-Ciprian Zăvoianu, Edwin Lughofer, Werner Koppelstätter, Günther Weidenholzer, Wolfgang Amrhein, and Erich Peter Klement. 2015. Performance Comparison of Generational and Steady-State Asynchronous Multi-Objective Evolutionary Algorithms for Computationally-Intensive Problems. *Knowledge-Based Systems* 87 (Oct. 2015), 47–60. doi:10.1016/j.knosys.2015.05.029