# Avoiding Excess Computation in Asynchronous Evolutionary Algorithms

Eric O. Scott[1,2], Mark Coletti[3], Catherine D. Schuman[3], Bill Kay[3], Shruti R. Kulkarni[3], Maryam Parsa[2,3], and Kenneth A. De Jong[2]

[1] MITRE Corporation, McLean VA 22102, USA,
escott@mitre.org,
[2] George Mason University, Fairfax VA 22030, USA
kdejong@gmu.edu
[3] Oak Ridge National Laboratory, Oak Ridge TN 37830, USA
{colettima,schumancd,kaybw,kulkarnisr,parsam}@ornl.gov

**Abstract.** Asynchronous evolutionary algorithms are becoming increasingly popular as a means of making full use of many processors while solving computationally expensive search and optimization problems. These algorithms excel at keeping large clusters fully utilized, but may sometimes inefficiently sample an excess of fast-evaluating solutions at the expense of higher-quality, slow-evaluating ones. We introduce a steady-state parent selection strategy, SWEET ("Selection whilE EvaluaTing"), that sometimes selects individuals that are still being evaluated and allows them to reproduce early. This gives slow-evaluating individuals that have higher fitnesses an increased ability to multiply in the population. We find that SWEET appears effective in simulated take-over time analysis, but that its benefit is confined mostly to early in the run, and our preliminary study on an autonomous vehicle controller problem that involves tuning a spiking neural network proves inconclusive.

**Keywords:** optimization, evolutionary algorithms, parallel computation, asynchronous algorithms

## 1 Introduction

Evolutionary algorithms (EAs) offer a remarkably general approach to search and optimization problems in a diversity of fields and are widely known for the

ease with which they can be parallelized to take advantage of distributed high-performance computing (HPC) resources [8]. Many computationally expensive applications have benefited from the parallelism offered by EAs and related methods like particle-swarm optimization [1].

In recent years, however, concern has grown around a fundamental inefficiency that plagues traditional generational EAs: the most popular distribution strategy for EAs is to follow a *synchronous controller-worker scheme*, where each iteration sends off a population of solutions to be evaluated on worker nodes. In this scheme, the controller waits for the entire population to complete evaluating before evolution continues. When individual evaluation-times vary, however, this means that many processors are left *idle* as the algorithm waits for the longest-evaluating solution to finish processing. The problem of idle time in synchronous EAs is most severe when a large number of processors are used, as just one long-running job can bring a large system to a halt, and network connections, processor heterogeneity, or scheduling concerns introduce additional sources of evalution-time variance [22]. With the advent of widely available super-compute clusters, this situation has become the norm rather than the exception.

Many researchers have turned to *asynchronous fitness evaluation schemes* as an alternative to traditional generational (that is, synchronous) algorithms [14, 22]. In particular, asynchronous steady-state evolutionary algorithms (ASSEAs) can often reach near-perfect resource utilization throughout an experiment (see Figure 1), particularly when the algorithm's computational cost is dominated by fitness evaluation on the workers (as is often the case when, say, tuning the parameters or behaviors of an expensive simulation [7, 11]). Asynchronous EAs are growing in popularity, and have most recently been applied to a variety computationally challenging problems such as deep neural network hyperparameter tuning [13, 5], evolutionary reinforcement learning [15], and simulation problems in air traffic management [19].

While ASSEAs succeed in recovering idle processing resources, their dynamical behavior is impacted by interactions between fitness and evaluation time in a way that more traditional algorithms are not—particularly when evaluation time is a *heritable trait* that is passed from parents to offspring. ASSEAs may sometimes be biased toward fast-evaluating regions of the search space, for example, which may either help or hinder optimization depending on the problem being solved [26, 21].

This paper focuses on a concern that arises when there is a correlation between evaluation time and fitness. Many problems of interest to EAs exhibit just such a correlation: either negatively (as when longer-evaluating solutions tend to have poor fitness values) or positively (i.e., longer-evaluating solutions tend to have good fitness values). Take the case of evolving a controller for a task such as a cart-pole or vehicle control problem, for example. Here, better-performing individuals may "survive" longer (i.e., don't drop the pole or don't crash) and thus take longer to evaluate, resulting in fitness evaluation times being positively correlated with fitness scores. As shown anecdotally in Figure 1, though the asynchronous algorithm reduces the idle time incurred by a cluster

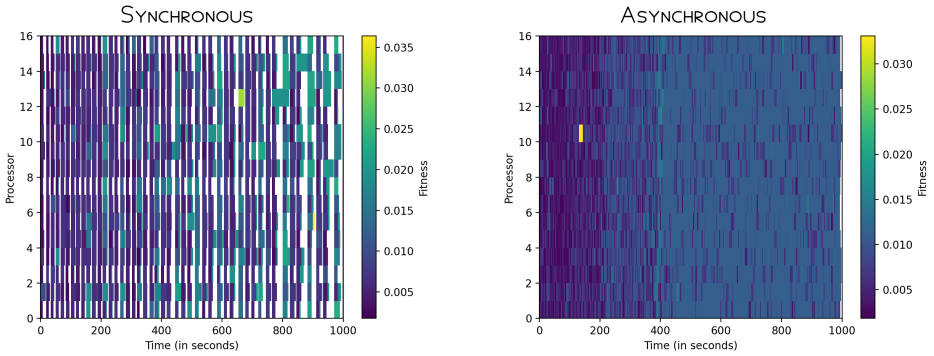


**Fig. 1.** Example runs of a synchronous (left) and asynchronous (right) EA applied to the same vehicle control problem, each visualized as a heatmap of the processor workloads. The synchronous algorithm exhibits a great deal of idle processor time (white gaps). The asynchronous algorithm eliminates idle time, but does not necessarily use the recovered computational resources effectively. In both plots, each block of color indicates a fitness evaluation of a particular duration, and each job is colored according the resulting fitness value.

of processors, it is clear that it is also dominated by faster-evaluating individuals, even though there is a high performing (but slower-evaluating) individual that has been evaluated. Whereas in the synchronous approach, most of the individuals being evaluated trend upward as the evolution goes on, even though there is significant idle time. In this case, the processors are kept busy under the asynchronous scheme, but the work that they are performing in evaluating low-performing individuals is not necessarily "useful" work for this particular application.

We propose to mitigate this problem of *excess work* in asynchronous evolution with a simple mechanism: we allow parents to be selected randomly from not just the population, but also from among individuals that have begun evaluation on a processor but have not yet completed. This gives long-evaluating individuals a change to reproduce before their fitness has been evaluated, potentially accelerating the propagation of the genetic material that they encode.

For convenience, we refer to this method as Selection whileE Evaluating (SWEET). Our core research question asks whether SWEET improves the convergence behavior of ASSEAs in the positive-correlation case:

**Research Question**: Does including processing individuals in selection reduce "excess work" in ASSEAs when evaluation time is positively correlated with fitness (longer individuals are better)? i.e. does it lead to an improvement in performance?

## 2 Background

The problem of idle resources during synchronous data collection affects a wide class of distributed algorithms. In a machine learning context, for instance, mainstream approaches to federated learning across edge devices typically impose a synchronous synthesis step [16]. But this leads to the so-called "straggler problem," in which the system must wait for the slowest learner in its network to complete training [25]. This has recently motivated a family of asynchronous federated learning algorithms that avoid idle time, but which potentially raise other issues—such as biasing the global learner toward information provided by faster-running local learners [3, 2]. Though the details differ, this situation is closely analogous to similar behavior that occurs with asynchronous evolutionary algorithms.

Evolutionary algorithms are particularly well-known for the ease with which they can be parallelized and distributed across large clusters of processors. Parallel implementations of EAs have been deeply studied, and can be broadly divided into 1) *panmictic* models, in which variation and selection operators are applied to a single monolithic population, and 2) *structured population* models (including both island models and spatial EAs), in which multiple sub-populations are evolved separately with some degree of loosely-coupled communication [1]. Panmictic controller-worker methods are among the simplest and most-widely applied of these architectures, and yield significant speedup when applied to problems where the computational cost of fitness evaluation dwarfs other costs.

Asynchronous steady-state evolutionary algorithms first appeared in the 1990's for both single- and multi-objective problems [14, 23], and have steadily gained importance as HPC paradigms have changed across science and engineering [10]. ASSEAs arise naturally as a parallel generalization of (panmictic) steady-state EAs (SSEAs) [24], in that both algorithm families generate offspring solutions on a controller node one-at-a-time which compete for a place in a fixed-size population after their fitness is evaluated. ASSEAs, however, use multiple worker processors to evaluate several solutions simultaneously, integrating them into the population in the order in which they finish. This order-shuffling effect (which Depolli et al. term *selection lag* [9]) introduces considerable complexity into ASSEA behavior whenever some solutions take longer to evaluate than others, which remains an obstacle to fully understanding the strengths and weaknesses of ASSEAs as an optimization strategy [20].

Most theoretical work on ASSEAs, has focused on two main threads of inquiry, rooted in the fact that almost all ASSEA applications are motivated as a means to avoid the idle time incurred by synchronous controller-worker algorithms. First, researchers have sought to quantify the increase in resource efficiency that asynchrony provides—by reporting empirical performance improvements [4], and by analytically deriving lower bounds on expected speedup [27, 22]. Second, a common concern is that the selection lag effect in ASSEAs will cause the algorithm to be biased toward regions of the solution space that are cheap to evaluate—making optimization w.r.t. fitness more difficult on some applications. This *evaluation-time bias* effect was first empirically reported by

Yagoubi et al. [26]. In a more detailed analysis, Scott and De Jong found no evidence of eval-time bias on flat fitness landscapes [22], but demonstrated that it occurs as an interaction between fitness and evaluation-time gradients, and that it sometimes occurs most strongly during the initialization phase of the algorithm [21]. In addition, Harada has recently introduced a new theoretical model for analyzing evaluation-time bias [12].

In this paper, we observe that in the case where better solutions tend to evaluate more slowly, asynchronously filling processors with additional (relatively fast-evaluating) computation may not always yield benefit. In some cases the additional evaluations may assist with beneficial exploration of the space; but in others they may consume resources that would be better used in exploitation. In these scenarios, we hypothesize that it would be beneficial not just to minimize idle time or evaluation-time variance, but to use the available processors to rapidly promote new genetic material. The distinction that steady-state algorithms draw between parent selection and survival selection offers us an opportunity to implement this idea: steady-state algorithms often use *random selection* for one of these selection operators. Random selection does not require fitness information, however—which means that we are able to propose a method that selects parents before they have completed evaluating.

## 3 Methodology

We use a combination of simulation and an autonomous vehicle application to evaluate our research question. In this section we describe the asynchronous algorithm under study, and the specific hypotheses that we will test in terms of takeover times and performance. All of our algorithms and simulations were implemented atop the Library for Evolutionary Algorithms in Python (LEAP), which has built-in support for distributed ASSEAs [6].[5]

### 3.1 The Asynchronous Steady-State EA

Asynchronous steady-state EAs seek to reduce the overall idle time of a distributed synchronous EA by ensuring that when a processor has completed an evaluation, a new individual is immediately produced to take its place. With a steady-state approach, individuals are integrated into the population one-at-a-time. Algorithm 1 gives the pseudo-code for the ASSEA approach that we use in this work. In particular, an initial population is created and immediately inserted into a queue to be evaluated across the workers. As their evaluations are completed, they are automatically inserted into the population until the population is full. Once the population is full, a newly evaluated individual is compared with a randomly selected member of the population; if the newly evaluated individual's fitness is higher, it replaces the randomly selected individual within

[5] The code for most of our methodology is published at https://github.com/SigmaX/Avoiding-Excess-Computation-UKCI2021 under the Academic Free License 3.0.

the population. In all of our experiments, we fix the number of total births to allow as the birth budget. If the total number of births is less than the birth budget, we generate a new individual by selecting a parent from the population and then performing reproduction operations to produce a child to be evaluated (incrementing the number of births).

**Algorithm 1 ASSAE design.** This shows details on how we asynchronously update a population of individuals of posed solutions.

```
P_0 ← init_pop()                                          ▷ Initial population
b ← size(P_o)                                             ▷ Initial number of births
P_p ← ∅                                                   ▷ Initialize an empty pool
async_eval(P_0)                                           ▷ Fan out population to workers
while I_e ← evaluated() do                                ▷ Next evaluated individual
    if is_full(P_p) then  I_c ← select_competitor(P_p)    ▷ Choose a competitor
        if I_e > I_c then                                 ▷ Replace only if better
            remove(P_p, I_c)
            insert(P_p, I_e)
    else                                                  ▷ Pool not full yet, so just insert
        insert(P_p, I_e)
    if b < birth_budget then
        I_p ← select_parent(P_p)                          ▷ Select parent
        I_o ← reproduce(I_p)                              ▷ Create offspring
        async_submit(I_o)                                 ▷ Send to worker for evaluation
        b ← b + 1                                         ▷ Increment births
return P_p                                                ▷ Return best individuals
```

We compare two variations of Algorithm 1: A) the **basic asynchronous** selection strategy implements the `select_parent()` operator by randomly choosing and individual from the pool $P_p$, whereas B) the **SWEET** strategy applies random selection to the union $P_p + E$ of the pool and *currently evaluating individuals*.

### 3.2 Discrete-Event Simulation

We begin our study of asynchronous EAs by imposing artificial evaluation times over fitness landscapes that are otherwise easy to compute. This allows us to perform many experimental runs quickly while we test specific hypotheses in a controlled environment.

For this set of experiments, we follow the example of [21] in using a discrete-event simulation (DES) to simulate the behavior of a cluster of processors under artificial evaluation times while the ASSEA runs. In the simulation, each individual $\mathbf{x}$ is assigned an evaluation time from the function $t(\mathbf{x})$, which is defined as part of the experiment. During evolution, the "cluster" uses a priority queue to instantly step to the next completed evaluation event at each iteration of the algorithm.

The cluster DES assumes that the run time of the algorithm is entirely defined by fitness evaluation costs, and that other aspects of the algorithm—such as the overhead of selection and variation operators, or of scheduling and network communication among a large number of processors—is negligible. This assumption generally holds in a great many applications, but may be challenged in scenarios that involve very high-dimensional genomes (in which case variation operators can be expensive to apply) or evaluation on very large clusters (where scheduling and network effects among thousands of CPUs can pose significant overhead).

*Takeover times* are a traditional methodology for evaluating the degree to which selection operators increase the frequency of high-performing genotypes in an evolutionary population [8]. Here we use takeover times to measure the impact of asynchronous selection strategies on the frequency of a good-fitness, slow-evaluating genotype.

> **Hypothesis 1**: Given an initial population that contains a **single good-fitness, slow-evaluating individual** that evolves under cloning and selection alone, the SWEET strategy will lead to significantly faster *takeover times* than the basic asynchronous selection strategy.

We run an ASSEA using the cluster DES, using an initial population configured to contain exactly one individual of `HIGH` genotype (good-fitness, slow-evaluating). The rest of the population is initialized to have `LOW` genotype (poor-fitness, fast-evaluating). We fix the evaluation-time differential between the two genotypes to a ratio of 100 to 1 (i.e. a fast individual evaluates 100x faster than slow ones).

Because our insertion operator uses tournament competition with zero chance of keeping the poorer individual, the initial high-fitness individual will always successfully complete for a place in the population, and the fraction of `HIGH`-type individuals will grow monotonically under selection and cloning. To formally test **Hypothesis 1**, we report the simulated time required for each of 50 runs to converge to a population of 100% `HIGH` genotype (i.e. the takeover time).

### 3.3 F1TENTH: An Autonomous Vehicle Problem

Our example of a problem with a positive correlation between evaluation time and fitness is a spiking neural network (SNN) learning problem: we aim to learn an SNN controller to drive a simulated vehicle that corresponds to the F1TENTH autonomous racing platform [18].[6] The fitness evaluation for the F1TENTH example computes the mean of how far (in units of distance) the car is able to drive, using the simulated SNN as a controller, before crashing across six training tracks. We use the EA to determine the weights, thresholds, and synaptic delays of an SNN with 20 input neurons (for 10 LIDAR inputs that use a spike-encoding scheme that requires 2 neurons per input), 10 hidden

[6] https://f1tenth.org/

neurons, and 40 output neurons. The 20 input neurons are fully connected to the 10 hidden neurons, and the 10 hidden neurons are fully connected to the 40 output neurons (as in a typical feed-forward neural network). We also include all-to-all connectivity among the hidden neurons, resulting in a recurrent neural network. The 40 output nodes correspond to 11 speed values and 29 possible steering angles.

We train an SNN for neuromorphic hardware deployment to a platform called $\mu$Caspian [17], which requires integer weights, thresholds, and delays. Thus, we use an integer representation in the genome. The genome includes a total of 3,050 genes (i.e., 1,490 for the weights, 1,490 for the synaptic delays, and 70 for the thresholds). The synaptic weights are bounded between -255 and 255 (inclusive), the synaptic delays are bounded between 0 and 15 (inclusive), and the neuron thresholds are bounded between 0 and 255 (inclusive). In all cases, we used random integer mutations with the expected number of mutations set to 1.

**Hypothesis 2**: On the F1TENTH problem, the SWEET strategy will lead to significantly higher-quality solutions and/or require fewer evaluations compared to the basic asynchronous strategy.

## 4 Results

### 4.1 Simulated Takeover Times

The results of our DES experiment with takeover times are shown in Figure 2. The SWEET strategy exhibits a considerably shorter (and statistically significant via Wilcoxon rank-sum at $p \approx 10^{-9}$) median takeover time (505.5 vs. 621.5, where the units of time are arbitrary in the simulation). Plotting the individual takeover curves shows that the behavior of the two algorithms differs qualitatively, with SWEET assuming a sort of pseudo-generational trajectory, as groups of `HIGH`-type individuals repeatedly complete evaluating at nearly the same time. Overall, these results **confirm Hypothesis 1** and suggest that the SWEET strategy allows high-fitness genetic material from long-running individuals to more rapidly increase its frequency within a population—though we observe that the majority of the benefit occurs at the beginning of the run.

### 4.2 F1TENTH

In Figure 3 the left figure depicts the difference between the best fitnesses for each run between two sets of runs. One run was an ASSEA that used uniform random selection from the single population that is asynchronously updated with freshly evaluated individuals. The other run similarly selected parents from the same population, but also selected with equally uniform chance individuals that were still being evaluated. Though the median values and distributions were similar (Wilcoxon rank-sum test $p = 0.8$), the "SWEET" approach was able to find a better solution. The best for the "basic async" out of 10 runs was 0.069066,

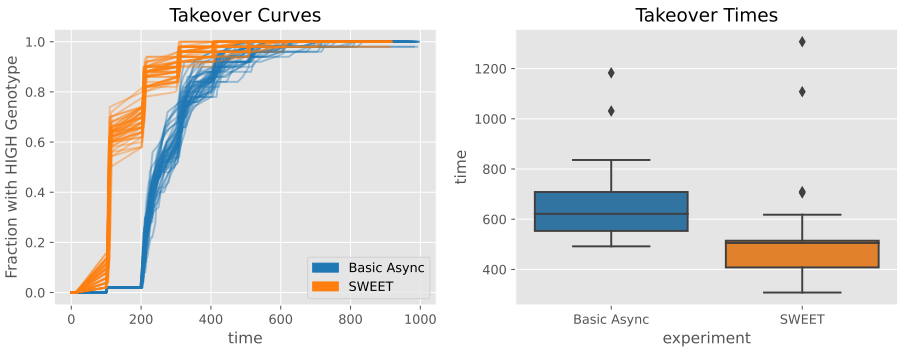


**Fig. 2.** Takeover experiments for 50 runs under selection & cloning, seeded with exactly 1 `HIGH`-type individual and 49 `LOW`-type. Left: individual `HIGH` frequency curves for each run. Right: distribution of the time until convergence to a `HIGH` frequency of 1.0.

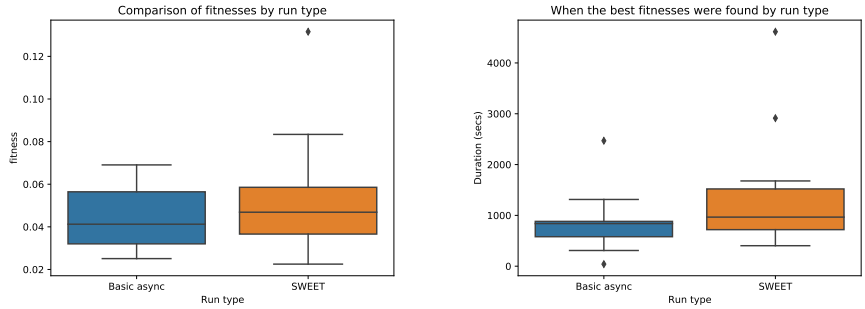


**Fig. 3.** The left figure compares the best fitnesses between the runs where parents were selected just from the population vs. selecting from the population and the set of currently evaluating individuals. These are respectively labeled "basic async" and "SWEET". Higher-fitnesses are better and reflect the quality of the simulated F1TENTH race car on various race tracks. The figure on the right shows the distribution of how long it took to find the best fitness.

and for the "SWEET" 0.131565. The right figure compares the times it took to find the best solution for each run by run type. Though the "basic async" runs do appear to have taken less time, they are not significantly so (Wilcoxon rank-sum $p = 0.38$). However, we would expect that it would take longer to find the best fitnesses since that approach intentionally is also potentially selecting longer-evaluating individuals.

Figure 4 shows the average best-so-far fitnesses between the two sets of runs with the sample points taken every 250 births for simplification and with a 95% confidence interval. Though the confidence intervals show significant overlap between the "basic async" and "SWEET" results, nonetheless it is promising that the "SWEET" results found better models and with fewer births than with the "basic async" approach. Overall, **Hypothesis 2 remains unconfirmed**.

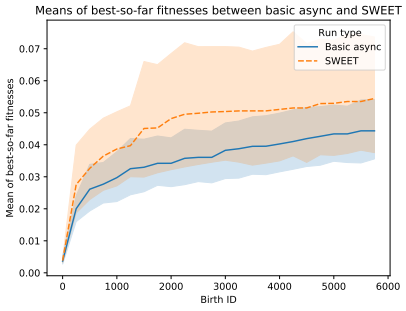


**Fig. 4.** This shows the means of the best-so-far fitnesses aggregated by run and run type with the shaded areas showing the 95% confidence intervals. Though there is significant overlap between the two sets of run, the SWEET runs did find better solutions.

## 5 Conclusions

We have proposed a simple steady-state parent selection strategy, SWEET ("Selection WhilE EvaluaTing"), which is intended to offset the tendency that asynchronous steady-state EAs have to inefficiently sample an excess of fast-evaluating solutions at the expense of higher-quality, slow-evaluating ones. Simulated experiments with takeover times suggest that our method is able to more quickly increase the frequency of high-performing, slow-evaluating alleles. But our results on the F1TENTH autonomous vehicle problem are inconclusive, and more study will be needed to evaluate the potential of the SWEET strategy on real-world problems.

## 6 Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract number DE-AC05-00OR22725.

This material is also based upon work supported by the United States Air Force Award #FA9550-19-1-0306. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

## References


[1] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. Vol. 47. John Wiley & Sons, 2005.

[2] Zheng Chai et al. "Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data". In: *arXiv preprint arXiv:2010.05958* (2020).

[3] Yujing Chen et al. "Asynchronous online federated learning for edge devices with non-iid data". In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE. 2020, pp. 15–24.

[4] Alexander W Churchill, Phil Husbands, and Andrew Philippides. "Tool sequence optimization using synchronous and asynchronous parallel multi-objective evolutionary algorithms with heterogeneous evaluations". In: *2013 IEEE Congress on Evolutionary Computation*. IEEE. 2013, pp. 2924–2931.

[5] Mark Coletti, Alex Fafard, and David Page. "Troubleshooting deep-learner training data problems using an evolutionary algorithm on Summit". In: *IBM Journal of Research and Development* 64.3/4 (2019), pp. 1–12.

[6] Mark A Coletti, Eric O Scott, and Jeffrey K Bassett. "Library for evolutionary algorithms in Python (LEAP)". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 2020, pp. 1571–1579.

[7] Matteo D'Auria et al. "Distributed, Automated Calibration of Agent-based Model Parameters and Agent Behaviors". In: *Autonomous Agents and Multiagent Systems (AAMAS)* (2020).

[8] K. A. De Jong. *Evolutionary computation: A unified approach*. Cambridge, MA: MIT Press, 2006.

[9] Matjaž Depolli, Roman Trobec, and Bogdan Filipič. "Asynchronous master-slave parallelization of differential evolution for multi-objective optimization". In: *Evolutionary computation* 21.2 (2013), pp. 261–291.

[10] Juan J Durillo et al. "A study of master-slave approaches to parallelize NSGA-II". In: *2008 IEEE international symposium on parallel and distributed processing*. IEEE. 2008, pp. 1–8.

[11] Chathika Gunaratne and Ivan Garibay. "Evolutionary model discovery of causal factors behind the socio-agricultural behavior of the Ancestral Pueblo". In: *Plos one* 15.12 (2020), e0239922.

[12] Tomohiro Harada. "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. 2019, pp. 1–8.

[13] Max Jaderberg et al. "Population based training of neural networks". In: *arXiv preprint arXiv:1711.09846* (2017).

[14] Jinwoo Kim. "Hierarchical asynchronous genetic algorithms for parallel/distributed simulation-based optimization." In: (1994).

[15] Kyunghyun Lee et al. "An Efficient Asynchronous Method for Integrating Evolutionary and Gradient-based Policy Search". In: *Advances in Neural Information Processing Systems* 33 (2020).

[16] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[17] J Parker Mitchell, Catherine D Schuman, and Thomas E Potok. “A small, low cost event-driven architecture for spiking neural networks on fpgas”. In: *International Conference on Neuromorphic Systems 2020*. 2020, pp. 1–4.

[18] Matthew O’Kelly et al. “F1tenth: An open-source evaluation environment for continuous control and reinforcement learning”. In: *Proceedings of Machine Learning Research* 123 (2020).

[19] Alessandro Pellegrini et al. “Simulation-Based Evolutionary Optimization of Air Traffic Management”. In: *IEEE Access* 8 (2020), pp. 161551–161570.

[20] Khaled Rasheed and Brian D Davison. “Effect of global parallelism on the behavior of a steady state genetic algorithm for design optimization”. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. Vol. 1. IEEE. 1999, pp. 534–541.

[21] Eric O Scott and Kenneth A De Jong. “Evaluation-Time Bias in Quasi-Generational and Steady-State Asynchronous Evolutionary Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 2016, pp. 845–852.

[22] Eric O Scott and Kenneth A De Jong. “Understanding simple asynchronous evolutionary algorithms”. In: *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*. 2015, pp. 85–98.

[23] Timothy J Stanley and Trevor N Mudge. “A Parallel Genetic Algorithm for Multiobjective Microprocessor Design.” In: *ICGA*. Citeseer. 1995, pp. 597–604.

[24] L Darrell Whitley. “The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best.” In: *International Conference on Genetic Algorithms*. Vol. 89. Fairfax, VA. 1989, pp. 116–123.

[25] Qi Xia et al. “A Survey of Federated Learning for Edge Computing: Research Problems and Solutions”. In: *High-Confidence Computing* (2021), p. 100008.

[26] Mouadh Yagoubi, Ludovic Thobois, and Marc Schoenauer. “Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs”. In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE. 2011, pp. 21–28.

[27] Alexandru-Ciprian Zăvoianu et al. “On the performance of master-slave parallelization methods for multi-objective evolutionary algorithms”. In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2013, pp. 122–134.