Using Cache or Credit for Parallel Ranking and Selection

HARUN AVCI and BARRY L. NELSON, Northwestern University, USA
EUNHYE SONG, Georgia Institute of Technology, USA
ANDREAS WÄCHTER, Northwestern University, USA

12

In this article, we focus on ranking and selection procedures that sequentially allocate replications to systems by applying some acquisition function. We propose an acquisition function, called gCEI, which exploits the gradient of the complete expected improvement with respect to the number of replications. We prove that the gCEI procedure, which adopts gCEI as the acquisition function in a serial computing environment, achieves the asymptotically optimal static replication allocation of Glynn and Juneja in the limit under a normality assumption. We also propose two procedures, called caching and credit, that extend any acquisition-function-based procedure in a serial environment into both synchronous and asynchronous parallel environments. While allocating replications to systems, both procedures use persistence forecasts for the unavailable outputs of the currently running replications, but differ in usage of the available outputs. We prove that, under certain assumptions, the caching procedure achieves the same asymptotic allocation as in the serial environment. A similar result holds for the credit procedure using gCEI as the acquisition function. In terms of efficiency and effectiveness, the credit procedure empirically performs as well as the caching procedure, despite not carefully controlling the output history as the caching procedure does, and is faster than the serial version without any number-of-replications penalty due to using persistence forecasts. Both procedures are designed to solve small-to-medium-sized problems on computers with a modest number of processors, such as laptops and desktops as opposed to high-performance clusters, and are superior to state-of-the-art parallel procedures in this setting.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms**; **Modeling and simulation**; *Simulation evaluation;*

Additional Key Words and Phrases: Simulation optimization, ranking & selection, parallel computing

ACM Reference format:
Harun Avci, Barry L. Nelson, Eunhye Song, and Andreas Wächter. 2023. Using Cache or Credit for Parallel Ranking and Selection. *ACM Trans. Model. Comput. Simul.* 33, 4, Article 12 (October 2023), 28 pages.
https://doi.org/10.1145/3618299

This research was partially supported by National Science Foundation Grant Numbers DMS-1854562 and DMS-1854659. A preliminary version of this article [Avci et al. 2021] was published in the *Proceedings of the 2021 Winter Simulation Conference.*

Authors' addresses: H. Avci and B. L. Nelson, Department of Industrial Engineering & Management Sciences, Northwestern University, Evanston, IL, USA, 60208; e-mails: harun.avci@u.northwestern.edu, nelsonb@northwestern.edu; E. Song, School of Industrial and Systems Engineering, Georgia Institute of Technology, 755 Ferst Drive, Atlanta, GA, USA, 30332; e-mail: eunhye.song@isye.gatech.edu; A. Wächter, Department of Industrial Engineering & Management Sciences, Northwestern University, Evanston, IL, USA, 60208; e-mail: andreas.waechter@northwestern.edu.

1049-3301/2023/10-ART12 $15.00
https://doi.org/10.1145/3618299

1 INTRODUCTION

The **ranking and selection (R&S)** problem is one of the classical stochastic simulation optimization problems. The goal is to identify the system with the largest (or smallest) mean performance among a finite number of systems, where the mean performance of each system can only be estimated using simulation output. In the past several decades, many procedures have been developed to tackle R&S simulation optimization problems. Desirable features for these procedures include delivering statistical guarantees on solution quality, being scalable to any finite number of systems, effectively controlling the number of simulation replications, being adaptive to different computational environments, and, most importantly, doing all of these in a computationally and statistically efficient way. See Hong et al. [2021] for an introduction to, and overview of, R&S procedures.

The aim of many R&S procedures is to return the best system or a good system given the decision-maker's tolerance. The latter is defined as a system whose mean is within the tolerance from the best. Since the performance of each system can only be estimated from simulation outputs, the procedures may fail to achieve this aim. This makes the probability of returning the best system (i.e., **probability of correct selection, PCS**) or a good system (i.e., **probability of good selection, PGS**) less than one in general. Therefore, most procedures are designed to control or minimize this error. In this article, we focus on good selection.

Depending on the statistical guarantee, Hunter and Nelson [2017] classify R&S procedures as fixed precision and fixed budget. Fixed-precision procedures attempt to allocate simulation effort as efficiently as possible until a certain termination criterion is met. These procedures are (ideally) guaranteed to return one of the good systems with a prespecified frequentist PGS. However, fixed-budget procedures allocate a limited simulation budget to maximize PGS. These procedures are typically Bayesian or Bayesian-inspired and include versions of **optimal computing budget allocation (OCBA)** [Chen et al. 2000], **expected improvement (EI)** [Chen and Ryzhov 2019], **knowledge gradient (KG)** [Frazier et al. 2008], and best-arm identification [Jamieson and Nowak 2014]. They tend to use an *acquisition function*, which returns a single system to simulate at each iteration. In this article, we propose a Bayesian fixed-budget R&S procedure, called the gCEI procedure, that can also provide fixed-precision stopping. The procedure adopts the **gradient-based complete expected improvement (gCEI)** acquisition function introduced in Avci et al. [2021], which is closely related to EI.

The EI criterion, initially created by Jones et al. [1998] for Bayesian optimization of a deterministic computer model, does not directly account for the uncertainty in the output from a stochastic simulation. To incorporate this uncertainty, Salemi et al. [2019] propose **complete expected improvement (CEI)** in a Gaussian Markov random field framework for discrete simulation optimization. For R&S problems under a normality assumption, Chen and Ryzhov [2019] propose an adjustment called **modified CEI (mCEI)**. To obtain optimal asymptotic performance (defined below), mCEI enforces the necessary balance between simulating the best system and the rest in the limit. The gCEI acquisition function exploits the gradient of CEI with respect to the number of replications, treating the number of replications as continuous [Avci et al. 2021]. Unlike CEI, gCEI reflects the benefit of additional replications, which plays an important role when the procedure is parallelized.

Typically, fixed-budget R&S procedures choose to simulate systems sequentially and adapt as more output data are obtained. We define an *allocation* as the fraction of a fixed budget of replications that is assigned to each system. Using large-deviation theory, Glynn and Juneja [2004] provide an expression for the asymptotically optimal static allocation under a fairly general distributional assumption. Their allocation is optimal in the sense that PCS converges to one at the fastest possible rates as the simulation budget increases. Unfortunately, this rate-optimal allocation cannot be computed in practice, because it depends on parameters of the underlying output

distributions. Therefore, several procedures have been designed to be adaptive and achieve a rate-optimal allocation asymptotically while aggressively pursuing the best system in small samples. Recently, many such procedures have been shown to converge to various asymptotically optimal allocations; see, for example, the **asymptotically optimal myopic allocation policy (AOMAP)** of Peng and Fu [2017], mCEI of Chen and Ryzhov [2019], and the **top-two Thompson sampling (TTTS)** of Russo [2020]. In this article, we prove that the gCEI procedure achieves the rate-optimal allocation of Glynn and Juneja [2004] in the limit. The procedure closest to gCEI in the literature is mCEI, which attains the same limit, but is empirically outperformed by gCEI in finite samples [Avci et al. 2021].

While the procedures mentioned above are designed for a serial computing environment, R&S benefits from the power of parallel computing, leading to the literature on parallel R&S that now spans two decades. In one of the earliest papers that combines the statistical efficiency of R&S with the effectiveness of parallel algorithms, Yücesan et al. [2001] (and its conference proceedings version: [Luo et al. 2000]) introduce a fixed-budget Bayesian OCBA framework for a web-based parallel environment. Similarly, Yoo et al. [2009] implement a modified OCBA that enhances both search and sampling efficiencies. Kamiński and Szufel [2018] propose asynchronous extensions of OCBA and KG as well as a synchronous extension of KG and compare them for small- and large-scale problems. Each of these papers extends the OCBA or KG acquisition function to the parallel setting. More recently, the EI and CEI acquisition functions have been extended to selecting the best *set* of q solutions to simulate in parallel; see Ginsbourger et al. [2007] and Semelhago et al. [2022], respectively. Unfortunately, the computational overhead of these extensions is prohibitive.

In addition to the above-mentioned parallel R&S procedures that are designed for a fixed-budget setting, there are several others designed for a fixed-precision setting. Using a simple divide-and-conquer approach in an indifference-zone formulation, Chen [2005] applies a multistage procedure on a local network with a small number of processors. Substantially extending the divide-and-conquer approach, the **good selection procedure (GSP)** of Ni et al. [2017] delivers a PGS guarantee under the assumption of normally distributed output. In a fixed-precision, frequentist setting, Luo et al. [2015] extend KN [Kim and Nelson 2001] to two parallel versions: synchronized **vector-filling KN (VKN)** and **asynchronous asymptotic parallel selection (APS)**. VKN exactly imitates KN in terms of conducting comparisons and making elimination decisions, while APS makes elimination decisions only when specific simulations complete and uses all of the available outputs at that time. Pei et al. [2018] propose a frequentist parallel R&S procedure, called **bisection parallel adaptive survivor selection (bi-PASS)**, which has a statistical advantage for a large number of systems. In another study, the same authors evaluate the performance of bi-PASS by comparing it to GSP and a subset selection procedure for large-scale problems [Pei et al. 2020, 2022]. Zhong and Hong [2022] propose the knockout tournament procedure in which the unknown best does not have to be compared to all other systems to be declared as the winner. See Hunter and Nelson [2017] and Hong et al. [2021] for surveys of parallel R&S.

Compared to the serial environment, making decisions in a synchronous or asynchronous parallel environment is more challenging. In the synchronous environment, where systems are simulated in batches, one needs to choose which system(s) to simulate and decide how many replications to allocate to each chosen system in the batch. In the asynchronous environment, where a single system is chosen to simulate whenever a processor becomes idle, one needs to take into account the currently running replications on the busy processors to benefit from additional partial information. *This article contributes to the parallel R&S literature with two different procedures that can address these challenges in both synchronous and asynchronous environments.*

Under both of the parallel procedures, a single system is chosen to simulate for each idle processor by repeatedly calling *any* user-specified sequential acquisition function, including, but not

limited to, gCEI. Accounting for the currently running replications, whose outputs are yet to be available, is important, as more than one replication can be running at the same time in a parallel environment. For the unavailable outputs of these replications, both procedures use the sample means of the corresponding systems as persistence forecasts until the outputs become available. However, the two procedures differ in how they utilize the available outputs. In particular, the first one, called "caching," uses some desired portion of the available outputs and caches the rest until they are needed. This "desired portion" is constructed as if the procedure was implemented in a serial environment without the persistence forecast. However, the second procedure, called "credit," uses all of the available outputs. The caching procedure imitates the corresponding serial procedure, while the credit procedure follows a greedy approach. We prove that, under certain assumptions, the caching procedure using any acquisition function achieves the same allocation limit in the parallel environments as in the serial one. We also prove a similar result for the credit procedure when gCEI is adopted as the acquisition function. Empirically, we show that these parallel versions are not only faster than the serial version in terms of wall-clock time, as one would hope, but also that using persistence forecasts leads to no significant increase in the number of replications consumed relative to the serial version that employs only actual outputs. Stated differently, our parallel procedures using gCEI inherit the replication-efficiency of serial gCEI while executing much faster.

The caching and credit procedures are the most competitive in parallel computing environments with a relatively small number of processors such as one would find in personal computers and for small-to-medium-sized problems. They do not eliminate any system. Elimination offers computational benefits when the number of systems is very large. Also, a large number of processors provides much quicker elimination. Therefore, very large-scale problems, in terms of the numbers of systems and processors, tend to favor elimination-based procedures. In the numerical experiments, we consider problems with up to 100 systems and 64 processors, representing the most common R&S problems in practice. The caching and credit procedures do not require any specialized computing platforms, such as clusters of computers or distributed computing using message passing. They are suitable for the most commonly used multi-processor computing platforms, such as desktops and laptops, and operating systems.

Our preliminary work is published in Avci et al. [2021], where the gCEI acquisition function is introduced and adopted to create the gCEI procedure in a serial environment and demonstrate its finite-sample efficiency; only a sketch of the proof of its convergence to the rate-optimal allocation is provided under the known variance assumption. In this work, we provide the complete proof of the convergence of the gCEI procedure in a more general case with unknown variances. Moreover, the extension to parallel environments is completely new.

The remainder of this article is organized as follows: We formulate the R&S problem along with a stopping condition in Section 2. We state the gCEI acquisition function and the gCEI procedure in a serial environment and show the convergence results in Section 3. We introduce the caching and credit procedures for a synchronous environment in Section 4 and an asynchronous environment in Section 5, respectively. Empirical performance evaluations are in Section 6. Conclusions are provided in Section 7.

2 PRELIMINARIES

Let $\mathcal{S} = \{1, 2, \ldots, k\}$ be the set of systems. Each system $i \in \mathcal{S}$ has an unknown mean μ_i and bigger is better. We assume that $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_{k-1} \leq \mu_k$ unknown to us; however, we prove asymptotic properties of the algorithms for $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_{k-1} < \mu_k$; i.e., no ties for the best system.

From a Bayesian perspective, the unknown mean of each system i, μ_i, has a prior distribution $\mathrm{N}(\bar{\mu}_i(0), 1/\bar{\theta}_i(0))$ where $\bar{\mu}_i(0)$ and $\bar{\theta}_i(0)$ are the prior mean and precision, respectively. The prior

mean represents the initial belief about the true value of μ_i, whereas the prior precision quantifies the confidence in this belief. We employ a non-informative prior (i.e., $\bar{\theta}_i(0) = 0$) and assume that the prior distributions of μ_i's are independent of each other. This assumption is common in the R&S literature and enables us to show the theoretical convergence of the gCEI procedure. Notice that we use μ_i's to denote the true, fixed means of the systems.

At each iteration $t = 0, 1, \ldots,$ a single system $x(t)$ is chosen to simulate. We emphasize that iteration t is defined by the tth simulation decision. Let $Y(t)$ be the simulation output of the corresponding replication of $x(t)$; we consider $(x(t), Y(t))$ as the system-output pair of iteration t. When $x(t)$ starts to be simulated and when $Y(t)$ is obtained depend on the environment. In the serial environment of Section 3, $Y(t)$ is obtained by simulating $x(t)$ at iteration t. In the synchronous parallel environment of Section 4, more than one system is simulated in parallel and the corresponding outputs are obtained in batches periodically. In the asynchronous parallel environment of Section 5, $x(t)$ starts to be simulated at the beginning of iteration t but $Y(t)$ may be obtained at another iteration. We assume that the simulation execution times are finite almost surely, so each output is obtained in finite time with probability one.

Additionally, we assume that the outputs obtained by simulating system i are independent and identically distributed $\mathrm{N}(\mu_i, \sigma_i^2)$ random variables, where $0 < \sigma_i^2 < \infty$ is the variance inherent to the stochastic simulation output of system i. In this article, we initially assume that σ_i^2's are known, but later allow estimates. We also assume that each system is simulated independently of the others (no common random numbers). The use of common random numbers can be counterproductive, since different systems most likely have different numbers of replications under our procedures [Nelson and Staum 2006].

Let $\mathcal{H}^t$ be the "history" consisting of system-output pairs up to, but not including, iteration t. In the remainder of this section, whenever notation is introduced as a function of t, it is conditional on $\mathcal{H}^t$. Also, let $r_i(t)$ denote the total number of replications performed for system i through iteration t. Since we employ the non-informative prior, for each system i the posterior mean is $\bar{\mu}_i(t) = \bar{Y}_i(t)$ for $r_i(t) > 0$, where $\bar{Y}_i(t)$ is the sample mean, and the posterior precision is $\bar{\theta}_i(t) = r_i(t)/\sigma_i^2$. Notice that $\bar{\mu}_i(t)$ is a random variable that represents the updated belief about unknown μ_i.

Let $k(t)$ be the sample-best system at iteration t, i.e.,

$$k(t) = \underset{i \in \mathcal{S}:\, r_i(t) > 0}{\arg\max} \{\bar{\mu}_i(t)\} = \underset{i \in \mathcal{S}:\, r_i(t) > 0}{\arg\max} \{\bar{Y}_i(t)\}.$$

To ease notation, we use $\mathcal{S}_j$ to represent $\mathcal{S}\backslash\{j\}$, e.g., $\mathcal{S}_{k(t)} = \mathcal{S}\backslash\{k(t)\}$. For any $i \in \mathcal{S}_{k(t)}$, the posterior distribution of $\mu_i - \mu_{k(t)}$ given $\mathcal{H}^t$ is $\mathrm{N}(\bar{\mu}_i(t) - \bar{\mu}_{k(t)}(t), \lambda_i(t))$, where $\lambda_i(t) = 1/\bar{\theta}_i(t) + 1/\bar{\theta}_{k(t)}(t)$. Thus, the complete expected improvement (CEI) for system $i \in \mathcal{S}_{k(t)}$ is

$$\mathrm{CEI}_i(t) = \mathbb{E}\left[\max\{\mu_i - \mu_{k(t)}, 0\} \mid \mathcal{H}^t\right] = \sqrt{\lambda_i(t)}\, f\left(\frac{\bar{\mu}_i(t) - \bar{\mu}_{k(t)}(t)}{\sqrt{\lambda_i(t)}}\right),$$

where $f(z) = z\Phi(z) + \phi(z)$ with ϕ and Φ being the standard normal probability density and cumulative distribution functions, respectively [Chen and Ryzhov 2019; Salemi et al. 2019]. The role of CEI is to indicate which system to simulate next to quickly identify the best. To facilitate comparison with fixed-precision procedures in our numerical experiments, we introduce a CEI-based stopping condition, thereby obtaining a fixed-precision Bayesian procedure.

We first define the posterior **probability of good selection (pPGS)** of the sample-best system as

$$\mathrm{pPGS}(t) = \mathbb{P}\left\{\mu_{k(t)} > \mu_i - \delta,\ \forall i \in \mathcal{S}_{k(t)} \mid \mathcal{H}^t\right\},$$

where $\delta > 0$ quantifies the tolerance of the decision-maker for suboptimality. In our problem, $\text{pPGS}(t)$ can be computed as

$$\text{pPGS}(t) = \mathbb{E}\left[\prod_{i\in\mathcal{S}_{k(t)}} \mathbb{P}\left\{\mu_{k(t)} > \mu_i - \delta \mid \mu_{k(t)}, \mathcal{H}^t\right\} \,\middle|\, \mathcal{H}^t\right]. \tag{1}$$

Since evaluating Equation (1) can be computationally expensive for a large number of systems, we use Slepian's inequality [Slepian 1962] to get a cheap lower bound on $\text{pPGS}(t)$:

$$\text{pPGS}(t) \geq \prod_{i\in\mathcal{S}_{k(t)}} \mathbb{P}\left\{\mu_{k(t)} > \mu_i - \delta \mid \mathcal{H}^t\right\}. \tag{2}$$

This lower bound can be computed as

$$\text{pPGS}_{\text{Slep}}(t) = \prod_{i\in\mathcal{S}_{k(t)}} \Phi\left(\frac{\delta - \bar{\mu}_i(t) + \bar{\mu}_{k(t)}(t)}{\sqrt{\lambda_i(t)}}\right).$$

Our procedures stop when $\text{pPGS}_{\text{Slep}}(t) \geq P^*$, where $P^* \in (1/k, 1)$ is the confidence level of the decision-maker. This stopping criterion is also adopted in Branke et al. [2005, 2007]. Let $T = \min\{t : \text{pPGS}_{\text{Slep}}(t) \geq P^*\}$ denote the random stopping time. Notice that if all systems are simulated infinitely often in the limit, then $\text{pPGS}_{\text{Slep}}(t)$ converges to 1 as $t \to \infty$.

We note that $\text{pPGS}_{\text{Slep}}(t)$ can be computed because we initially assume that the variances are known. When the variances are unknown, as in the numerical experiments, we can approximate $\text{pPGS}_{\text{Slep}}(t)$ by applying the Welch approximation to the posterior distribution of $\mu_i - \mu_{k(t)}$ for each $i \in \mathcal{S}_{k(t)}$. Thus, for the unknown-variance case,

$$\text{pPGS}_{\text{Slep}}(t) \approx \prod_{i\in\mathcal{S}_{k(t)}} \Psi_{\nu_i(t)}\left(\frac{\delta - \bar{\mu}_i(t) + \bar{\mu}_{k(t)}(t)}{\sqrt{\hat{\lambda}_i(t)}}\right),$$

where Ψ_ν is the cumulative distribution function of the Student's t-distribution with ν degrees of freedom, $\hat{\lambda}_i(t) = \hat{\sigma}_i^2(t)/r_i(t) + \hat{\sigma}_{k(t)}^2(t)/r_{k(t)}(t)$,

$$\nu_i(t) = \frac{\left[\hat{\sigma}_i^2(t)/r_i(t) + \hat{\sigma}_{k(t)}^2(t)/r_{k(t)}(t)\right]^2}{\left[\hat{\sigma}_i^2(t)/r_i(t)\right]^2/(r_i(t)-1) + \left[\hat{\sigma}_{k(t)}^2(t)/r_{k(t)}(t)\right]^2/(r_{k(t)}(t)-1)},$$

$$\hat{\sigma}_i^2(t) = \frac{1}{r_i(t)-1}\sum_{\tau=0}^{t-1} \mathbb{I}(x(\tau) = i)\left(Y(\tau) - \bar{Y}_i(t)\right)^2$$

and $\mathbb{I}(\cdot)$ is the indicator function [Branke et al. 2007]. Notice that $\hat{\sigma}_i^2(t)$ is the sample variance, an estimator of σ_i^2. Of course, $\text{pPGS}_{\text{Slep}}(t)$ is a posterior statement of belief based on the simulation evidence, not a frequentist PGS characterizing a procedure's performance over repeated trials. However, posterior statements are often used as an approximation of corresponding frequentist statements, especially when employing non-informative priors. For fixed-precision stopping, we will approximate frequentist PGS by $\text{pPGS}_{\text{Slep}}(t)$ in the empirical studies, but one should not expect them to match perfectly.

ALGORITHM 1: Generic Adaptive Procedure in a Serial Environment

1: Let $x(0) \leftarrow i$ for some $i \in \mathcal{S}$. Obtain $Y(0)$. Also, let $\mathcal{H}^1 \leftarrow \{(x(0), Y(0))\}$ and $t \leftarrow 1$.
2: **repeat**
3: $x(t) \leftarrow AF(\mathcal{H}^t)$.
4: Obtain $Y(t)$ by simulating $x(t)$, update $\mathcal{H}^{t+1} \leftarrow \mathcal{H}^t \cup \{(x(t), Y(t))\}$ and $t \leftarrow t+1$.
5: **until** Stopping condition satisfied.
6: Return $k(T) = \arg\max_{i \in \mathcal{S}} \{\bar{\mu}_i(T)\}$ as the selected best system.

3 SERIAL COMPUTING ENVIRONMENT

As a first step toward defining the parallel procedures, we consider the R&S problem in a serial computing environment. At each iteration $t = 0, 1, \ldots$, a single output $Y(t)$ is obtained by simulating $x(t)$. Thus, $\mathcal{H}^t = \{(x(\tau), Y(\tau))\}_{\tau=0}^{t-1}$. Notice that given $\mathcal{H}^t$,

$$r_i(t) = \sum_{\tau=0}^{t-1} \mathbb{I}(x(\tau) = i) \text{ and } \bar{Y}_i(t) = \frac{1}{r_i(t)} \sum_{\tau=0}^{t-1} \mathbb{I}(x(\tau) = i)\, Y(\tau) \text{ when } r_i(t) > 0.$$

We define a generic acquisition function $AF(\cdot)$ that takes the history $\mathcal{H}^t$ as input and returns a single system to simulate as output. Using $AF(\cdot)$, Algorithm 1 presents a generic adaptive procedure. In the following subsections, we describe the gCEI acquisition function [Avci et al. 2021] and the mCEI acquisition function [Chen and Ryzhov 2019] in detail. We refer to a procedure that adopts a specific acquisition function as in Algorithm 1 by the name of its acquisition function, e.g., gCEI procedure.

3.1 gCEI Acquisition Function

Treating $r_i(t)$ for $i \in \mathcal{S}$ as continuous-valued, Avci et al. [2021] show that

$$\frac{\partial \mathrm{CEI}_i(t)}{\partial r_i(t)} = -\frac{\sigma_i^2}{(r_i(t))^2} \frac{1}{2\sqrt{\lambda_i(t)}} \phi\left(\frac{\bar{\mu}_i(t) - \bar{\mu}_{k(t)}(t)}{\sqrt{\lambda_i(t)}}\right) \text{ and } \frac{\partial \mathrm{CEI}_i(t)}{\partial r_{k(t)}(t)} = -\frac{\sigma_{k(t)}^2}{(r_{k(t)}(t))^2} \frac{1}{2\sqrt{\lambda_i(t)}} \phi\left(\frac{\bar{\mu}_i(t) - \bar{\mu}_{k(t)}(t)}{\sqrt{\lambda_i(t)}}\right)$$

for $i \in \mathcal{S}_{k(t)}$, whereas $\partial \mathrm{CEI}_i(t)/\partial r_j(t) = 0$ for $j \in \mathcal{S} \backslash \{i, k(t)\}$. Notice that $\partial \mathrm{CEI}_i(t)/\partial r_j(t) < 0$ for $j \in \{i, k(t)\}$, reflecting that additional replications at either i or $k(t)$ are expected to reduce the CEI of system i. Which system to simulate next is chosen based on the following condition:

$$\sum_{i \in \mathcal{S}_{k(t)}} \frac{\partial \mathrm{CEI}_i(t)}{\partial r_{k(t)}(t)} \overset{?}{\leq} \min_{i \in \mathcal{S}_{k(t)}} \frac{\partial \mathrm{CEI}_i(t)}{\partial r_i(t)} = \frac{\partial \mathrm{CEI}_{g(t)}(t)}{\partial r_{g(t)}(t)}, \tag{3}$$

where

$$g(t) = \arg\min_{i \in \mathcal{S}_{k(t)}} \frac{\partial \mathrm{CEI}_i(t)}{\partial r_i(t)}$$

with ties broken arbitrarily. If Equation (3) holds, then the total impact of simulating $k(t)$ is potentially greater than simulating $g(t)$, and thus simulating $k(t)$ is preferred to $g(t)$, i.e., $x(t) = k(t)$. However, if Equation (3) does not hold, then simulating $g(t)$ is preferred, i.e., $x(t) = g(t)$. Therefore, the gCEI acquisition function is

$$AF_{\mathrm{gCEI}}(\mathcal{H}^t) = \begin{cases} k(t), & \text{if } \sum_{i \in \mathcal{S}_{k(t)}} \partial \mathrm{CEI}_i(t)/\partial r_{k(t)}(t) \leq \partial \mathrm{CEI}_{g(t)}(t)/\partial r_{g(t)}(t), \\ g(t), & \text{otherwise.} \end{cases}$$

Since the corresponding derivatives do not exist when a system has not yet been simulated, we initially set those derivatives to $-\infty$, which makes the gCEI procedure simulate each system once

in the first k iterations. When the variances are unknown the derivatives are set to $-\infty$ until at least two outputs are obtained for the corresponding system to be able to compute the sample variances.

Note that R&S treats the systems as categorical, implying no spatial structure to exploit. In discrete-decision-variable simulation optimization problems that do exhibit spatial structure, this relationship is often modeled via spatial correlation among feasible solutions. The gradient of CEI can be computed and is still relevant when there is such correlation.

Let $\boldsymbol{\alpha}(t) = (\alpha_1(t), \ldots, \alpha_k(t))$ denote the empirical allocation at iteration t, where $\alpha_i(t) = r_i(t)/t$ for all $i \in \mathcal{S}$. Also, let $\boldsymbol{\alpha}^* = (\alpha_1^*, \ldots, \alpha_k^*)$ denote the unique rate-optimal allocation in Glynn and Juneja [2004] such that $\alpha_i^* > 0$, $\sum_{i \in \mathcal{S}} \alpha_i^* = 1$,

$$\frac{(\mu_i - \mu_k)^2}{\sigma_i^2/\alpha_i^* + \sigma_k^2/\alpha_k^*} = \frac{(\mu_j - \mu_k)^2}{\sigma_j^2/\alpha_j^* + \sigma_k^2/\alpha_k^*}, \ \forall i, j \in \mathcal{S}_k, \tag{4}$$

and

$$\sum_{j \in \mathcal{S}_k} \left(\frac{\alpha_j^*}{\sigma_j}\right)^2 = \left(\frac{\alpha_k^*}{\sigma_k}\right)^2. \tag{5}$$

Assuming that the best system is unique, i.e., $\mu_{k-1} < \mu_k$, and the variances are known, Avci et al. [2021] provide a sketch of the proof that $\boldsymbol{\alpha}(t)$ converges to $\boldsymbol{\alpha}^*$ almost surely under the gCEI procedure. In this article, we provide the complete proof for a more general result that includes Theorem 1 below as a special case. All proofs are in the appendices. Theorem 1 states that the gCEI procedure asymptotically achieves $\boldsymbol{\alpha}^*$ when the variances are known, or when the variances are unknown but continually updated via plug-in estimators $\hat{\sigma}_i^2(t)$.

Theorem 1. *If the best system is unique, then the gCEI procedure obtains $\boldsymbol{\alpha}(t) \to \boldsymbol{\alpha}^*$ almost surely as $t \to \infty$. This result still holds when σ_i^2 is replaced with $\hat{\sigma}_i^2(t)$ for all $i \in \mathcal{S}$ at each iteration t.*

3.2 mCEI Acquisition Function

Chen and Ryzhov [2019] present the mCEI acquisition function:

$$AF_{\text{mCEI}}(\mathcal{H}^t) = \begin{cases} k(t), & \text{if } \left(r_{k(t)}(t)/\sigma_{k(t)}\right)^2 < \sum_{i \in \mathcal{S}_{k(t)}} \left(r_i(t)/\sigma_i\right)^2, \\ c(t), & \text{otherwise,} \end{cases}$$

where $c(t) = \arg\max_{i \in \mathcal{S}_{k(t)}} \text{CEI}_i(t)$ and prove that the mCEI procedure asymptotically achieves the rate-optimal allocation, $\boldsymbol{\alpha}^*$, when the variances are known, or when the variances are unknown but continually updated via plug-in estimators.

3.3 Insights on gCEI

We briefly summarize what Avci et al. [2021] observe empirically by running the gCEI procedure in several numerical experiments under the fixed-budget framework. In all of those experiments, the outputs are assumed to be independent and normally distributed with known variances.

The first observation is that under the gCEI procedure, the frequentist PCS, estimated by averaging the correct selection across the macro-replications, converges to one as expected. Moreover, the gCEI procedure converges to the rate-optimal allocation of Glynn and Juneja [2004] as expected in the long run.

When the performance of the gCEI procedure is compared with three procedures from the literature—the mCEI procedure of Chen and Ryzhov [2019], AOMAP of Peng and Fu [2017], and TTTS of Russo [2020]—it appears that the gCEI procedure performs as well or better than these

ALGORITHM 2: Generic Adaptive Procedure in a Synchronous Environment

1: Use the first b_0 batch iterations to simulate each system at least twice.
2: Let $t \leftarrow b_0 p$ and initialize the history.
3: **repeat**
4: Choose to simulate $x(t), \ldots, x(t+p-1)$ by calling a BAF.
5: Obtain $Y(t), \ldots, Y(t+p-1)$ by simulating $x(t), \ldots, x(t+p-1)$ in parallel.
6: Update the history by calling a HUF and $t \leftarrow t+p$.
7: **until** Stopping condition satisfied.
8: Return $k(T) = \arg\max_{i \in \mathcal{S}} \{\bar{\mu}_i(T)\}$ as the selected best system.

other procedures based on how fast PCS converges to one. Even though both gCEI and mCEI procedures are proven to converge to the rate-optimal allocation of Glynn and Juneja [2004], the gCEI procedure tends to allocate less to the best system than the mCEI procedure for finite samples.

Another observation in experiments is that in the "slippage configuration," where all inferior systems have the same mean, the PCS converges to one faster for the gCEI procedure than an unrealistic procedure that employs the (assumed-to-be-known) rate-optimal allocation from the beginning. This observation emphasizes that the rate-optimal allocations address large-sample, not small-sample, behavior. Therefore, there are advantages to employing a procedure like gCEI, which aggressively improves PGS initially but converges to the rate-optimal allocation in the limit.

4 SYNCHRONOUS PARALLEL ENVIRONMENT

In this section, we extend acquisition-function-based procedures into a synchronous parallel environment with p processors. Let $\mathcal{W} = \{0, 1, \ldots, p-1\}$ denote the set of processors. In the synchronous environment, systems are simulated and the corresponding outputs are obtained in batches of size p. In particular, at the beginning of each batch iteration $b = 0, 1, \ldots$, the systems $\{x(bp+w),\ w \in \mathcal{W}\}$ are chosen to simulate next and each system $x(bp+w)$ is assigned to processor w. The systems $x(bp), \ldots, x(bp+p-1)$ need not all be different. Then, the corresponding outputs $\{Y(bp+w),\ w \in \mathcal{W}\}$ are accumulated before the beginning of the next batch iteration. In other words, the outputs are collected only at iterations $t = p, 2p, \ldots$. Therefore, the history is updated only at those iterations, and thus $\mathcal{H}^t = \mathcal{H}^{bp} = \{(x(\tau), Y(\tau))\}_{\tau=0}^{bp-1}$ for $t = bp, \ldots, bp+p-1$.

Using the same stopping and selection rules as in Section 3, Algorithm 2 presents a generic adaptive procedure for the synchronous environment. Each system is initially simulated at least twice in the first b_0 batch iterations so the sample means and variances can be computed, where $b_0 \geq 2k/p$. Then, at each batch iteration $b = b_0, b_0+1, \ldots$, the systems to simulate next are chosen by calling a **batch acquisition function (BAF)**, and the history is updated with the obtained outputs by calling a **history update function (HUF)**. In the following subsections, we introduce two procedures that adopt different BAFs and HUFs: caching and credit.

To facilitate the discussion, we first slightly modify the definition of $AF(\cdot)$ as a function that takes a list of system-output pairs, not necessarily $\mathcal{H}^t$, as the input and returns a single system to simulate as the output. The statistics used by $AF(\cdot)$ are conditional on and computed from the input. For example, slightly abusing notation, $AF(\mathcal{H})$ may require computing

$$r_i(\mathcal{H}) = \sum_{(\mathsf{x},\mathsf{Y}) \in \mathcal{H}} \mathbb{I}(\mathsf{x} = i),\ \bar{Y}_i(\mathcal{H}) = \frac{1}{r_i(\mathcal{H})} \sum_{(\mathsf{x},\mathsf{Y}) \in \mathcal{H}} \mathbb{I}(\mathsf{x} = i)\,\mathsf{Y}$$

and

$$\hat{\sigma}_i^2(\mathcal{H}) = \frac{1}{r_i(\mathcal{H}) - 1} \sum_{(\mathsf{x},\mathsf{Y}) \in \mathcal{H}} \mathbb{I}(\mathsf{x} = i) \left(\mathsf{Y} - \bar{Y}_i(\mathcal{H})\right)^2,$$

where (x, Y) are generic elements of $\mathcal{H}$.

Both caching and credit procedures call $AF(\cdot)$ sequentially to choose systems $\{x(bp+w),\ w \in \mathcal{W}\}$ at batch iteration b; the performance of the procedures depends on the choice of $AF(\cdot)$. Since the corresponding outputs $\{Y(bp+w),\ w \in \mathcal{W}\}$ are not available until the current batch iteration ends, the procedures use the corresponding sample means as persistence forecasts to stand in for those outputs. This is equivalent to assuming that an output with value of $\bar{Y}_i(\mathcal{H}^{bp})$ will be obtained if system i is simulated. Once system i is chosen to simulate, $r_i(t)$ is incremented by one while $\bar{Y}_i(t)$ remains the same, and hence $k(t)$ remains unchanged as well. We define $\mathcal{F}^t$ as a "forecast" at iteration t, which includes system-output pairs to use as persistence forecasts; namely, $\mathcal{F}^{bp} = \emptyset$ and $\mathcal{F}^t = \{(x(\tau), \bar{Y}_{x(\tau)}(\mathcal{H}^{bp}))\}_{\tau=bp}^{t-1}$ for $t = bp+1, \ldots, bp+p-1$.

Persistence forecasts are a simple, fast way to predict the impact of simulating more than one system in parallel. However, to benefit from persistence forecasts, the $AF(\cdot)$ must account for the benefit of increasing the number of replications within the batch, because the sample means will not change. Since gCEI is the *gradient* with respect to the number of replications, $AF_{\text{gCEI}}(\cdot)$ is a good candidate. Notice that with gCEI a system can be chosen to be simulated more than once within the batch, which would not be the case if, say, the $AF(\cdot)$ selected the p solutions with the largest CEIs. Choosing solutions and allocating replications is typically a challenge for acquisition-function-based procedures in a parallel environment; gCEI naturally makes both decisions in concert. The details of the caching and credit procedures are presented in Section 4.1 and Section 4.2, respectively.

4.1 Caching Procedure

If simulations were instantaneous, the serial procedures such as gCEI would be desirable because of their good finite-sample performances and asymptotic consistencies. We consider the sample path generated by the serial procedure, i.e., when $p = 1$, as the "desired path." Motivated to build the desired path independent of p, we propose the caching procedure that keeps obtained-but-not-yet-needed outputs in a "cache" until they can be used to construct the desired path. While choosing systems to simulate, this procedure uses only a subset of the history, called the "desired history."

Let $\mathcal{C}^t$ and $\mathcal{D}^t$ denote the cache and desired history, respectively, at iteration t. Together, they partition the history, i.e., $\mathcal{C}^t \cap \mathcal{D}^t = \emptyset$ and $\mathcal{C}^t \cup \mathcal{D}^t = \mathcal{H}^t$. Since the outputs are accumulated only at iterations $t = p, 2p, \ldots$, the cache and desired history are updated only at those iterations, and thus $\mathcal{C}^t = \mathcal{C}^{bp}$ and $\mathcal{D}^t = \mathcal{D}^{bp}$ for $t = bp, \ldots, bp+p-1$ at each batch iteration b. To initialize the sample means and variances at the beginning of batch iteration $b = b_0$, we set $\mathcal{C}^{b_0 p} = \emptyset$ and $\mathcal{D}^{b_0 p} = \{(x(\tau), Y(\tau))\}_{\tau=0}^{b_0 p-1}$. Then, at the end of each batch iteration $b = b_0, b_0+1, \ldots$, we update the cache and desired history after obtaining the outputs. Using temporary cache $\mathcal{C}$ and desired history $\mathcal{D}$ to choose systems to include in the desired history, Algorithm 3 presents the HUF for the caching procedure.

Algorithm 4 presents the BAF for the caching procedure in which the outputs in the cache are prioritized while choosing the systems to simulate. In particular, for any chosen system, the caching procedure first utilizes the outputs in the cache, if there are any, and then uses the corresponding sample means as the persistence forecasts. Meanwhile, the cache $\mathcal{C}^{bp}$ and desired history $\mathcal{D}^{bp}$ are not updated, leaving the task to the HUF (Algorithm 3), which is called in Step 6 of Algorithm 2. Notice that $\mathcal{C}^{bp}$ does not include any outputs of the chosen system $x(bp)$ to simulate for $w = 0$. They instead would have been included already in $\mathcal{D}^{bp}$ by the HUF because $\mathcal{F}^{bp} = \emptyset$.

When $p = 1$, the caching procedure with $AF(\cdot)$ reduces to the procedure adopting $AF(\cdot)$ in the serial environment, since $x(t) = AF(\mathcal{H}^t)$ as $\mathcal{D}^t = \mathcal{H}^t$ and $\mathcal{F}^t = \emptyset$ for all $t = 0, 1, \ldots$. For example, the caching procedure with $AF_{\text{gCEI}}(\cdot)$ is gCEI itself when $p = 1$. Suppose that the procedure

ALGORITHM 3: HUF for the Caching Procedure in a Synchronous Environment

Input: $\mathcal{C}^{bp}$, $\mathcal{D}^{bp}$ and $\{(x(\tau), Y(\tau))\}_{\tau=bp}^{bp+p-1}$
Output: $\mathcal{C}^{(b+1)p}$ and $\mathcal{D}^{(b+1)p}$
Let $\mathcal{C} \leftarrow \mathcal{C}^{bp} \cup \{(x(\tau), Y(\tau))\}_{\tau=bp}^{bp+p-1}$ and $\mathcal{D} \leftarrow \mathcal{D}^{bp}$.
$i \leftarrow AF(\mathcal{D})$.
while $\mathcal{C}_i \neq \emptyset$ **do** ▷ $\mathcal{C}_i = \{(x(s), Y(s)) \in \mathcal{C} : x(s) = i\}$
  Let $\tau \leftarrow \min\{s : (x(s), Y(s)) \in \mathcal{C}_i\}$.
  Update $\mathcal{C} \leftarrow \mathcal{C} \backslash \{(x(\tau), Y(\tau))\}$ and $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x(\tau), Y(\tau))\}$. ▷ $x(\tau) = i$
  $i \leftarrow AF(\mathcal{D})$.
end while
Let $\mathcal{C}^{(b+1)p} \leftarrow \mathcal{C}$ and $\mathcal{D}^{(b+1)p} \leftarrow \mathcal{D}$.

ALGORITHM 4: BAF for the Caching Procedure in a Synchronous Environment

Input: $\mathcal{C}^{bp}$ and $\mathcal{D}^{bp}$
Output: $x(bp), \ldots, x(bp+p-1)$
Let $t \leftarrow bp$, $\mathcal{C} \leftarrow \mathcal{C}^{bp}$, $\mathcal{D} \leftarrow \mathcal{D}^{bp}$ and $\mathcal{F}^t \leftarrow \emptyset$.
for $w \in \mathcal{W}$ **do**
  $i \leftarrow AF(\mathcal{D} \cup \mathcal{F}^t)$.
  while $\mathcal{C}_i \neq \emptyset$ **do** ▷ $\mathcal{C}_i = \{(x(s), Y(s)) \in \mathcal{C} : x(s) = i\}$
    Let $\tau \leftarrow \min\{s : (x(s), Y(s)) \in \mathcal{C}_i\}$.
    Update $\mathcal{C} \leftarrow \mathcal{C} \backslash \{(x(\tau), Y(\tau))\}$ and $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x(\tau), Y(\tau))\}$. ▷ $x(\tau) = i$
    $i \leftarrow AF(\mathcal{D} \cup \mathcal{F}^t)$.
  end while
  $x(t) \leftarrow i$.
  Update $\mathcal{F}^{t+1} \leftarrow \mathcal{F}^t \cup \{(x(t), \bar{Y}_{x(t)}(\mathcal{D}))\}$ and $t \leftarrow t+1$.
end for

adopting $AF(\cdot)$ in the serial environment converges to some allocation $\tilde{\boldsymbol{\alpha}} = (\tilde{\alpha}_1, \ldots, \tilde{\alpha}_k)$, with $\tilde{\alpha}_i > 0$ and $\sum_{i \in \mathcal{S}} \tilde{\alpha}_i = 1$. Assume that the sequence of simulation outputs for each system is not affected by the procedure. In other words, the output of a replication of a system depends only on the corresponding system and how many times the system has been simulated so far. Under this assumption, we prove that the caching procedure with $AF(\cdot)$ also converges to the same allocation $\tilde{\boldsymbol{\alpha}}$ if the cache size is finite almost surely. This result holds when the variances are known, or when the variances are unknown but continually updated via plug-in estimators, i.e., σ_i^2 is replaced with $\hat{\sigma}_i^2(\mathcal{D}^{bp})$ when $r_i(\mathcal{D}^{bp}) \geq 2$ for $t = bp, \ldots, bp+p-1$.

Theorem 2. *If the sequence of simulation outputs for each system is not affected by the procedure, $|\mathcal{C}^t| < \infty$ almost surely for all t and $\boldsymbol{\alpha}(t) \to \tilde{\boldsymbol{\alpha}}$ almost surely as $t \to \infty$ when $p = 1$, then the synchronous caching procedure obtains $\boldsymbol{\alpha}(t) \to \tilde{\boldsymbol{\alpha}}$ almost surely as $t \to \infty$ when $p > 1$.*

Remark. As a practical matter, the "not affected" condition will be satisfied if each system is assigned a distinct random generator seed or stream, effectively assigning a distinct block of (pseudo) random numbers to the simulation of each system.

The proof of Theorem 2 is straightforward from the definition of the desired history and the assumption of unaffected simulation output sequences. In particular, the desired history consists only of the outputs of the systems that would have been chosen to simulate if the procedure was implemented in the serial environment, i.e., if we had $p = 1$. Therefore, the allocation on the

ALGORITHM 5: BAF for the Credit Procedure in a Synchronous Environment

Input: $\mathcal{H}^{bp}$
Output: $x(bp), \ldots, x(bp+p-1)$
1: Let $t \leftarrow bp$ and $\mathcal{F}^t \leftarrow \emptyset$.
2: **for** $w \in \mathcal{W}$ **do**
3: $x(t) \leftarrow AF(\mathcal{H}^{bp} \cup \mathcal{F}^t)$.
4: Update $\mathcal{F}^{t+1} \leftarrow \mathcal{F}^t \cup \{(x(t), \bar{Y}_{x(t)}(\mathcal{H}^{bp}))\}$ and $t \leftarrow t+1$.
5: **end for**

desired history $\mathcal{D}^t$ converges to $\tilde{\boldsymbol{\alpha}}$ almost surely as $t \to \infty$. Since the cache size is assumed to be finite almost surely, it does not have any impact on the asymptotic allocation. Thus, $\boldsymbol{\alpha}(t) \to \tilde{\boldsymbol{\alpha}}$ almost surely as $t \to \infty$.

The cache size can be bounded explicitly for each system by slightly modifying the BAF in Algorithm 4. In particular, Steps 9 and 10 could be performed only if the output of the corresponding replication of system i can be cached (once it is accumulated at the end of the batch iteration) without violating a fixed cache size limit. Otherwise, only the forecast is updated $\mathcal{F}^t \leftarrow \mathcal{F}^t \cup \{(i, \bar{Y}_i(\mathcal{D}))\}$ without assigning system i to a processor and increasing the iteration counter t. The version of BAF for the caching procedure with limited cache size is presented in Appendix A.

Since the caching procedure with $AF_{\text{gCEI}}(\cdot)$ reduces to gCEI when $p = 1$, it is immediate from Theorems 1 and 2 that it converges to rate-optimal allocation $\boldsymbol{\alpha}^*$ for any $p \geq 1$.

Corollary 1. *If the sequence of simulation outputs for each system is not affected by the procedure, $|\mathcal{C}^t| < \infty$ almost surely for all t and the best system is unique, then the synchronous caching procedure with $AF_{gCEI}(\cdot)$ obtains $\boldsymbol{\alpha}(t) \to \boldsymbol{\alpha}^*$ almost surely as $t \to \infty$.*

4.2 Credit Procedure

Unlike the caching procedure, the credit procedure uses all information currently available whenever it makes a simulation decision by fully utilizing the obtained outputs and persistence forecasts. As the history includes all of the obtained outputs, it is initialized as $\mathcal{H}^{b_0 p} = \{(x(\tau), Y(\tau))\}_{\tau=0}^{b_0 p-1}$ at batch iteration $b = b_0$ and then updated at the end of each batch iteration by calling the following HUF:

$$\mathcal{H}^{(b+1)p} \leftarrow \mathcal{H}^{bp} \cup \{(x(\tau), Y(\tau))\}_{\tau=bp}^{bp+p-1}.$$

Presented in Algorithm 5, the BAF for the credit procedure passes both history and forecast to $AF(\cdot)$ to choose a single system to simulate at each iteration t within the batch, that is, $x(t) = AF(\mathcal{H}^{bp} \cup \mathcal{F}^t)$ for $t = bp, \ldots, bp+p-1$.

Under the uniqueness assumption of the best system, we prove that the credit procedure with $AF_{\text{gCEI}}(\cdot)$ converges to the rate-optimal allocation $\boldsymbol{\alpha}^*$ for any $p \geq 1$.

Theorem 3. *If the variances are known and the best system is unique, then the synchronous credit procedure with $AF_{gCEI}(\cdot)$ obtains $\boldsymbol{\alpha}(t) \to \boldsymbol{\alpha}^*$ almost surely as $t \to \infty$.*

Theorem 3 is shown under the assumption that the variances are known. To extend this convergence result to the unknown-variance case, for each system i, we first replace σ_i^2 with $\hat{\sigma}_i^2(\mathcal{H}^t)$ when $r_i(t) \geq 2$. Next, we show that the limiting allocation remains the same when the variances are unknown but continually updated via the sample variances.

Corollary 2. *If the best system is unique, then the synchronous credit procedure with $AF_{gCEI}(\cdot)$ that replaces σ_i^2 with $\hat{\sigma}_i^2(\mathcal{H}^t)$ for all $i \in \mathcal{S}$ at each iteration t obtains $\boldsymbol{\alpha}(t) \to \boldsymbol{\alpha}^*$ almost surely as $t \to \infty$.*

Similar to the caching procedure, when $p = 1$, the credit procedure with a given acquisition function $AF(\cdot)$ reduces to the procedure adopting $AF(\cdot)$ in the serial environment, since $x(t) = AF(\mathcal{H}^t)$ as $\mathcal{F}^t = \emptyset$ for all $t = 0, 1, \ldots$. For example, the credit procedure with $AF_{\text{gCEI}}(\cdot)$ is gCEI itself when $p = 1$. Therefore, following Corollary 2, the gCEI procedure in the serial environment with estimated variances converges to the rate-optimal allocation $\boldsymbol{\alpha}^*$; see Theorem 1.

Recall that the persistence forecasts assume that the next output of the to-be-simulated system will have the value of the corresponding sample mean. Clearly, this assumption is false and may make the credit procedure deviate from the sample path of system-output pairs generated by the serial procedure, especially, when p is large. Nevertheless, the credit procedure using gCEI as the acquisition function still achieves the rate-optimal allocation $\boldsymbol{\alpha}^*$ in the limit.

5 ASYNCHRONOUS PARALLEL ENVIRONMENT

In this section, we study the R&S problem in an asynchronous parallel environment with p identical processors. Recall that $\mathcal{W} = \{0, 1, \ldots, p-1\}$ denotes the set of processors. Different from the synchronous environment, we avoid idling a processor in this environment to obtain a speedup in terms of wall-clock time. Instead of waiting for all the currently running replications to complete, whenever a running replication on some processor completes, the corresponding output is obtained and a new system is assigned to the processor to simulate.

At the beginning of each iteration $t = 0, 1, \ldots$, a single system $x(t)$ is chosen and assigned to an idle processor to simulate next. If there is another idle processor, then we move to the next iteration without waiting to obtain the corresponding output $Y(t)$. Otherwise, we wait for a running replication on some processor to complete, making the processor idle, then move to the next iteration. This newly idle processor is not necessarily the processor that was tasked to simulate $x(t)$. In other words, even though $x(t)$ is chosen to simulate at the beginning of iteration t, its corresponding output $Y(t)$ may be obtained at iteration $t' \neq t$. Since the simulation times may vary across systems and replications, the order in which the systems are chosen to simulate may be different from the order in which the outputs are obtained.

To keep track of the currently running replications at iteration t, whose outputs are yet to be available, let $t^w(t)$ denote the iteration when the currently running replication on processor w started, i.e., when $x(t^w(t))$ was chosen to simulate. At the beginning of each iteration t, for all $w \in \mathcal{W}$, we set $t^w(t)$ to t if a new system is assigned to processor w and to $t^w(t-1)$ otherwise; consequently $t^w(t) \leq t$. For notational simplicity, we suppress the dependence of t^w on t. Therefore, t^w is updated only when a new system is assigned to processor w.

Using the same stopping and selection rules in Sections 3 and 4, Algorithm 6 presents a generic adaptive procedure for the asynchronous environment. Each system is initially simulated at least twice in the first t_0 iterations to be able to compute the sample means and variances, where $t_0 \geq 2k$. Then, for each $w \in \mathcal{W}$, $x(t_0 + w)$ is chosen and assigned to processor w to simulate. Here, t^w is initialized as $t_0 + w$. Once the replication of $x(t_0 + p - 1)$ starts on the last processor at the beginning of iteration $t = t_0 + p - 1$, all processors become busy, and we wait for some running replication to complete.

At iteration $t = t_0 + p - 1$, suppose that the currently running replication on processor w completes. In other words, the corresponding output $Y(t^w)$ of system $x(t^w)$ is obtained. The history is updated with $(x(t^w), Y(t^w))$ by calling a HUF, and we move to the next iteration $t + 1 = t_0 + p$. As processor w becomes idle, updating t^w to $t_0 + p$, a new system $x(t^w)$ is chosen to simulate next and assigned to processor w. The same steps are repeated for $t = t_0 + p, t_0 + p + 1, \ldots$ until the procedure terminates.

We modify the caching and credit procedures for the asynchronous environment in Sections 5.1 and 5.2, respectively. Similar to the synchronous versions, the procedures utilize the information

ALGORITHM 6: Generic Adaptive Procedure in an Asynchronous Environment

1: Use the first t_0 iterations to simulate each system at least twice.
2: Initialize the history.
3: **for** $w \in \mathcal{W}$ **do**
4: Let $t^w \leftarrow t_0 + w$. Choose to simulate $x(t^w)$ via an acquisition function and assign it to processor w.
5: **end for**
6: Let $t \leftarrow t_0 + p - 1$.
7: **repeat**
8: Wait until some processor $w \in \mathcal{W}$ obtains $Y(t^w)$.
9: Update the history with $(x(t^w), Y(t^w))$ by calling a HUF and $t \leftarrow t + 1$.
10: Update $t^w \leftarrow t$. Choose to simulate $x(t)$ via an acquisition function and assign it to processor w.
11: **until** Stopping condition satisfied.
12: Return $k(T) = \arg\max_{i \in \mathcal{S}} \{\bar{\mu}_i(T)\}$ as the selected best system.

ALGORITHM 7: Choosing a Single System to Simulate under the Caching Procedure in an Asynchronous Environment

Input: $\mathcal{C}^t$, $\mathcal{D}^t$ and $\mathcal{F}^t$
Output: $x(t)$
1: Let $\mathcal{C} \leftarrow \mathcal{C}^t$ and $\mathcal{D} \leftarrow \mathcal{D}^t$.
2: $i \leftarrow AF(\mathcal{D} \cup \mathcal{F}^t)$.
3: **while** $\mathcal{C}_i \neq \emptyset$ **do** ▷ $\mathcal{C}_i = \{(x(s), Y(s)) \in \mathcal{C} : x(s) = i\}$
4: Let $\tau \leftarrow \min\{s : (x(s), Y(s)) \in \mathcal{C}_i\}$.
5: Update $\mathcal{C} \leftarrow \mathcal{C} \backslash \{(x(\tau), Y(\tau))\}$ and $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x(\tau), Y(\tau))\}$. ▷ $x(\tau) = i$
6: $i \leftarrow AF(\mathcal{D} \cup \mathcal{F}^t)$.
7: **end while**
8: $x(t) \leftarrow i$.

about the currently running replications in the asynchronous environment by using the corresponding sample means as persistence forecasts to stand in for the unavailable outputs of those replications. Therefore, supposing that $x(t)$ is chosen for processor w at iteration t, we modify the forecast as

$$\mathcal{F}^t = \begin{cases} \{(x(\tau), \bar{Y}_{x(\tau)}(\mathcal{H}^t))\}_{\tau=t_0}^{t-1}, & \text{if } t_0 \le t \le t_0 + p - 1 \text{ (in Step 4 of Algorithm 6)}, \\ \{(x(t^\rho), \bar{Y}_{x(t^\rho)}(\mathcal{H}^t))\}_{\rho \in \mathcal{W} \backslash \{w\}}, & \text{if } t \ge t_0 + p \text{ (in Step 10 of Algorithm 6)}, \end{cases}$$

which includes all currently running replications on the other processors. Notice that $\mathcal{F}^t$ is a function of w, but we suppress w in the notation for simplicity.

5.1 Caching Procedure

The caching procedure in Section 4.1 can be modified for the asynchronous environment. Similar to the synchronous version, aiming to build the sample path as in the serial environment, the asynchronous caching procedure utilizes the cache and desired history as well as the persistent forecasts while choosing a single system to simulate next. Algorithm 7 presents the steps of choosing $x(t)$ at iteration t.

ALGORITHM 8: HUF for the Caching Procedure in an Asynchronous Environment

Input: $\mathcal{C}^t$, $\mathcal{D}^t$ and $(x(t^w), Y(t^w))$
Output: $\mathcal{C}^{t+1}$ and $\mathcal{D}^{t+1}$
1: Let $\mathcal{C} \leftarrow \mathcal{C}^t \cup \{(x(t^w), Y(t^w))\}$ and $\mathcal{D} \leftarrow \mathcal{D}^t$.
2: $i \leftarrow AF(\mathcal{D})$.
3: **while** $\mathcal{C}_i \neq \emptyset$ **do** ▷ $\mathcal{C}_i = \{(x(s), Y(s)) \in \mathcal{C} : x(s) = i\}$
4: Let $\tau \leftarrow \min\{s : (x(s), Y(s)) \in \mathcal{C}_i\}$.
5: Update $\mathcal{C} \leftarrow \mathcal{C} \backslash \{(x(\tau), Y(\tau))\}$ and $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x(\tau), Y(\tau))\}$. ▷ $x(\tau) = i$
6: $i \leftarrow AF(\mathcal{D})$.
7: **end while**
8: Let $\mathcal{C}^{t+1} \leftarrow \mathcal{C}$ and $\mathcal{D}^{t+1} \leftarrow \mathcal{D}$.

The cache and desired history are initialized as $\mathcal{C}^{t_0} = \emptyset$ and $\mathcal{D}^{t_0} = \{(x(\tau), Y(\tau))\}_{\tau=0}^{t_0-1}$, respectively, at iteration $t = t_0$. Then, letting $\mathcal{C}^t = \mathcal{C}^{t_0}$ and $\mathcal{D}^t = \mathcal{D}^{t_0}$ for $t = t_0, \ldots, t_0 + p - 2$, they are updated at each iteration $t = t_0 + p - 1, t_0 + p, \ldots$ by calling the HUF presented in Algorithm 8. In fact, Algorithm 8 is a special case of Algorithm 3 when $p = 1$, as the history is updated with a single output at each iteration.

We can extend the convergence results in the synchronous environment, i.e., Theorem 2, to the asynchronous environment.

Corollary 3. *If the sequence of simulation outputs for each system is not affected by the procedure, $|\mathcal{C}^t| < \infty$ almost surely for all t and $\boldsymbol{\alpha}(t) \to \tilde{\boldsymbol{\alpha}}$ almost surely as $t \to \infty$ when $p = 1$, then the asynchronous caching procedure obtains $\boldsymbol{\alpha}(t) \to \tilde{\boldsymbol{\alpha}}$ almost surely as $t \to \infty$ when $p > 1$.*

As with the proof of Theorem 2, the proof of Corollary 3 is straightforward from the assumption that the simulation output sequence is unaffected for each solution.

5.2 Credit Procedure

The credit procedure in Section 4.2 can be modified for the asynchronous environment. Similar to the synchronous version, the asynchronous credit procedure utilizes the entire history as well as the persistence forecasts. That is, at each iteration t,

$$x(t) = AF(\mathcal{H}^t \cup \mathcal{F}^t).$$

The history is initialized as $\mathcal{H}^{t_0} = \{(x(\tau), Y(\tau))\}_{\tau=0}^{t_0-1}$ at iteration $t = t_0$ and remains the same for $t = t_0, \ldots, t_0 + p - 2$. Then, it is updated with $(x(t^w), Y(t^w))$ for some $w \in \mathcal{W}$ at each iteration $t = t_0 + p - 1, t_0 + p, \ldots$ by calling the following HUF:

$$\mathcal{H}^{t+1} \leftarrow \mathcal{H}^t \cup \{(x(t^w), Y(t^w))\}.$$

Similar to the caching procedure, the convergence results of the synchronous credit procedure, i.e., Corollary 2, can be extended to the asynchronous version.

Corollary 4. *If the best system is unique, then the asynchronous credit procedure with $AF_{gCEI}(\cdot)$ that replaces σ_i^2 with $\hat{\sigma}_i^2(\mathcal{H}^t)$ for all $i \in \mathcal{S}$ at each iteration t obtains $\boldsymbol{\alpha}(t) \to \boldsymbol{\alpha}^*$ almost surely as $t \to \infty$.*

Again, the proof of Corollary 4 is omitted, as it is straightforward.

6 NUMERICAL EXPERIMENTS

Avci et al. [2021] compare the finite-sample behavior of serial gCEI to three well-established acquisition functions: mCEI, TTTS, and AOMAP; both gCEI and mCEI converge to the rate-optimal allocation in the limit, while TTTS and AOMAP converge to other limiting allocations. gCEI performs

as well as any of these and often significantly better in terms of probability of correct selection. These results led to our focus on gCEI for the parallel implementation. See also the discussion in Section 3.3 and Appendix C.1.

For the numerical experiments here, we consider three different problems: the airline-reservation example [Goldsman et al. 1991], an M/M/1 queue example (from an online Masterclass at http://users.iems.northwestern.edu/~nelsonb/RSMasterclass.html), and a stylized example with normally distributed simulation outputs. For these problems, the objective function can be evaluated analytically, enabling us to identify the good systems. Also, simulation execution times can be adjusted by the choice of problem parameters. Clearly the stylized example conforms perfectly to our output-distribution assumptions for gCEI. The more-realistic airline-reservation and M/M/1 queue examples do not have normally distributed outputs, but normal-theory R&S procedures have been shown to be robust when either a very large number of replications will be needed before termination (airline reservation) or the replication output is the average of a large number of more basic non-normal outputs (M/M/1 queue). In all experiments, we assume that the variances are unknown but continually updated via plug-in estimators.

In the airline-reservation example, there are $k = 4$ systems, which differ in the parameters of the time-to-failure and time-to-repair distributions and thus the expected time to failure, E[TTF]. Each system is modeled by a continuous-time Markov chain with three states, where each state represents a number of functional components. Setting the parameters accordingly, the E[TTF]'s are 828.3, 902.0, 910.9, and 1,002.0 days. Aiming to find the system with the maximum E[TTF], we set the tolerance $\delta = (1{,}002.0 - 910.9)/2$ so only the best system is considered "good."

In the M/M/1 queue example, system $i \in \mathcal{S}$ has an arrival rate of 1 and a service rate of $20i/k + 1$. Given service cost of 1 and waiting cost of 36, the expected cost of a system is estimated by averaging the waiting time of 1,000 customers in each replication. Aiming to find the system with the minimum cost (or maximum negative cost) among $k = 40$ systems, we set the tolerance $\delta = 0.192$ so only the best system is considered "good."

For these two problems, we first evaluate the performances of the caching and credit procedures in terms of efficiency (e.g., the total number of outputs obtained and the computation time) and effectiveness (i.e., the good selection). Varying the number of processors, we perform the performance evaluation in both synchronous and asynchronous environments. Then, we compare the caching and credit procedures to APS of Luo et al. [2015], which is an extension of KN [Kim and Nelson 2001] to the asynchronous environment by making elimination decisions only when specific simulations complete. We also compare the performance of caching and credit procedures when they are combined with mCEI instead of gCEI in Appendix C.1; for both procedures, gCEI achieves a higher empirical PGS with earlier stopping.

In all experiments, we initially allocate 100 and 30 replications to each system before applying any procedure on the airline-reservation and M/M/1 queue examples, respectively. The procedures stop when the stopping criterion introduced in Section 2 is met for the confidence level $P^* = 0.95$. We set the number of macro-replications $M = 5{,}000$ to be able to estimate PGS to two decimal places over a range of values. Letting T_m be the total number of outputs obtained (i.e., iterations) and $k(T_m)$ be the selected system, in the mth macro-replication, we introduce

$$\widehat{\text{PGS}}_1 = \frac{1}{M}\sum_{m=1}^{M} \mathbb{I}(\mu_{k(T_m)} > \mu_k - \delta) = \frac{1}{M}\sum_{m=1}^{M} \mathbb{I}(k(T_m) = k),$$

which is an estimate of the frequentist PGS, by averaging the good selection across the macro-replications. We also introduce $\widehat{T}_{mean}$, $\widehat{T}_{se}$, $\widehat{T}_{median}$, and $\widehat{T}_{max}$ to report the mean, standard error, median, and maximum of the total number of outputs obtained across macro-replications,

Table 1. Caching (Cache) vs. Credit (Crdt) Procedure on the Airline-reservation Example in Terms of the Total Number of Outputs Obtained and the Good Selection

		synchronous environment						asynchronous environment					
	$p = 1$	$p = 2$		$p = 4$		$p = 8$		$p = 2$		$p = 4$		$p = 8$	
	gCEI	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt
$\widehat{T}_{mean}$	1,039.2	1,040.4	1,078.2	1,058.7	1,077.5	1,100.5	1,084.2	1,070.4	1,053.9	1,084.2	1,049.6	1,085.6	1,102.1
$\widehat{T}_{se}$	9.4	9.6	9.8	9.6	9.7	9.9	9.9	9.6	9.7	10.0	9.4	9.5	9.7
$\widehat{T}_{median}$	847.0	832.0	871.0	852.0	876.0	896.0	872.0	869.0	850.5	877.0	845.0	883.0	908.0
$\widehat{T}_{max}$	5,187	5,394	5,748	5,820	7,152	5,328	5,024	4,985	6,071	5,845	4,439	5,471	5,617
$\widehat{\text{PGS}}_1$	0.928	0.923	0.928	0.927	0.921	0.927	0.936	0.928	0.924	0.923	0.920	0.928	0.931

Table 2. Caching (Cache) vs. Credit (Crdt) Procedure on the M/M/1 Queue Example in Terms of the Total Number of Outputs Obtained and the Good Selection

		synchronous environment						asynchronous environment					
	$p = 1$	$p = 2$		$p = 4$		$p = 8$		$p = 2$		$p = 4$		$p = 8$	
	gCEI	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt
$\widehat{T}_{mean}$	1,433.9	1,436.0	1,440.6	1,446.9	1,440.4	1,459.6	1,454.6	1,431.6	1,431.5	1,485.7	1,431.4	1,463.7	1,437.8
$\widehat{T}_{se}$	3.5	3.6	3.6	3.7	3.6	3.7	3.7	3.6	3.5	4.2	3.6	4.3	3.6
$\widehat{T}_{median}$	1,354.0	1,350.0	1,364.0	1,372.0	1,360.0	1,376.0	1,372.0	1,349.0	1,355.0	1,403.0	1,349.0	1,363.0	1,357.0
$\widehat{T}_{max}$	2,951	3,268	3,130	2,980	3,176	2,984	3,288	2,938	3,069	3,621	3,273	3,724	3,194
$\widehat{\text{PGS}}_1$	0.911	0.920	0.921	0.921	0.924	0.924	0.923	0.910	0.915	0.911	0.913	0.914	0.918

respectively; for example, $\widehat{T}_{mean} = \sum_{m=1}^{M} T_m/M$. All computations are executed on a desktop with a Windows 10 operating system, a 2.9 GHz Intel Core i7 CPU, 32 GB of RAM, 8 cores and 16 logical processors. The procedures are implemented in Python using the "concurrent.futures" package for parallel computing.

Tables 1 and 2 exhibit results for the airline-reservation and M/M/1 queue examples, respectively, in terms of the total number of outputs obtained and the PGS for the caching and credit procedures with different number of processors $p \in \{1, 2, 4, 8\}$ in both synchronous and asynchronous environments. Recall that both procedures reduce to the gCEI procedure when $p = 1$. The purpose of this analysis is to see if there is any degradation in number-of-replications efficiency or achieved PGS from parallelizing with persistence forecasts. Based on T and PGS, the performances of the two procedures do not differ significantly. This observation indicates that the credit procedure performs as well as the caching procedure despite not carefully controlling the history as the caching procedure does. Also, neither the environment nor the number of processors have significant impact on the procedures' performances. A possible explanation is that the persistence forecast is effective at predicting the desired path so the procedures tend to adhere closely to the desired path. See Appendix C.2 for a more detailed comparison of the procedures.

The caching and credit procedures (as well as the gCEI procedure in the serial environment) using pPGS stopping does not quite achieve the frequentist PGS $P^* = 0.95$ on PGS; it is slightly but consistently lower in these examples. Recall that a $\text{pPGS}_{\text{Slep}} \geq 0.95$ does not imply a frequentist PGS of 0.95, and it is also the case that the $\text{pPGS}_{\text{Slep}}$ is computed assuming normally distributed output, which is not the case in these examples. Despite all of this the underachievement of frequentist PGS is remarkably slight and not a concern in practice. We also note that any stopping criterion can be adopted for our procedures to improve their performances.

Next, we compare the caching and credit procedures to APS in terms of the speedup relative to APS, and then relative to their serial versions, for different number of processors $p \in \{1, 2, 4, 8\}$ in the asynchronous environment; APS becomes KN when $p = 1$. Different from the caching

Table 3. Wall-clock Execution Time Speedup of Caching (Cache) and Credit (Crdt) Procedures Relative to APS with the Same Number of Processors in the Asynchronous Environment

	M/M/1 queue example			airline-reservation example		
p	Cache	Crdt	APS	Cache	Crdt	APS
1	3.7	3.7	1	3.9	3.9	1
2	3.4	3.5	1	3.3	3.6	1
4	2.8	2.9	1	3.3	3.1	1
8	3.3	3.5	1	3.9	3.9	1

Table 4. Wall-clock Execution Time Speedup of Caching (Cache), Credit (Crdt), and APS Relative to Their Respective Serial Versions in the Asynchronous Environment

	M/M/1 queue example			airline-reservation example		
p	Cache	Crdt	APS	Cache	Crdt	APS
1	1	1	1	1	1	1
2	1.8	1.9	2.0	1.6	1.7	1.9
4	2.8	2.8	3.6	3.0	2.8	3.5
8	4.8	5.1	5.4	5.1	5.1	5.1

and credit procedures, APS does not make strategic decisions about which system to simulate next; instead it simulates all "surviving" systems in round-robin fashion and makes elimination decisions after performing all-pairwise comparisons when specific simulations complete. In these experiments, APS uses its natural frequentist stopping condition for 95% PCS, while for the caching and credit procedures, we use the $\text{pPGS}_{\text{Slep}}$ stopping condition; however, we target 97% $\text{pPGS}_{\text{Slep}}$ rather than 95%. By setting our $\text{pPGS}_{\text{Slep}}$ target to 97%, we achieve at least 95% *frequentist* PGS. This provides a fair comparison for timing, because it forces our procedures to take more replications than they would with a 95% $\text{pPGS}_{\text{Slep}}$ to achieve the same inference provided by APS.

Table 3 shows the speedup relative to APS with common numbers of processors; we define "speedup" as the ratio of wall-clock time for APS divided by wall-clock time for our procedures. When $p = 1$ they are all serial procedures, so the speedup for the caching and credit procedures comes from requiring about one-third the number of replications as APS to achieve 95% PCS. Notice that these speedups hold as we increase the number of processors, and that the speedups are slightly better for the airline-reservation example whose computational cost of simulation relative to the algorithmic overhead is higher than the M/M/1 queue example.

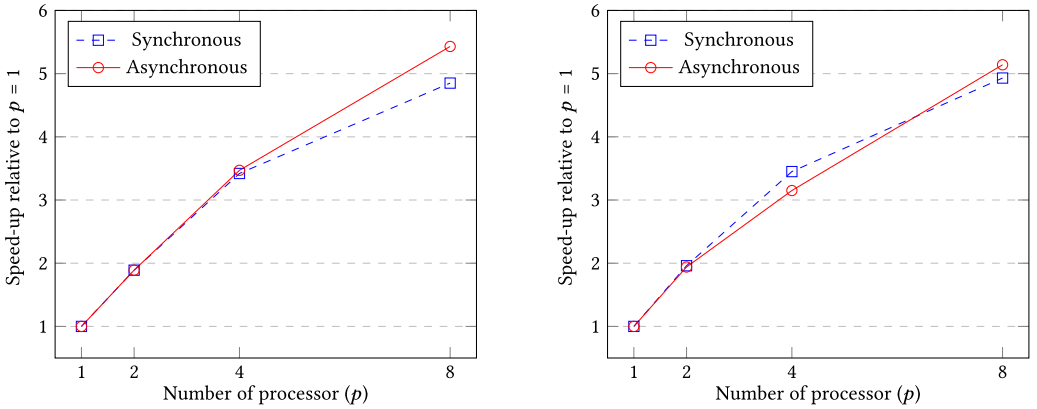
Table 4 shows the speedup of caching, credit, and APS relative to their serial versions as the number of processors increases. Notice again that the speedups are slightly better for the airline-reservation example, which has relatively higher simulation overhead vs. algorithm overhead.

Table 5 shows the estimated frequentist PGS, $\widehat{\text{PGS}}_1$, for caching, credit, and APS. Notice that the desired frequentist PGS, $P^* \geq 0.95$, is achieved in nearly all experiments.

To better compare our procedures to each other and to investigate the impact of a synchronous vs. asynchronous computing environment in terms of the wall-clock time speedup, we increase the simulation execution times. To do this, we simply make the processors simulate 100 replications of each assigned system and record only the simulation outputs of the last replications. We set the number of macro-replications $M = 100$ in these experiments. Also, we fix T to a reasonable deterministic constant (using Tables 1 and 2) for a fair comparison, i.e., the procedures terminate

Table 5. Estimated PGS for Caching (Cache), Credit (Crdt), and APS in the Asynchronous Environment

	M/M/1 queue example			airline-reservation example		
p	Cache	Crdt	APS	Cache	Crdt	APS
1	0.95	0.95	1	0.98	0.98	1
2	0.94	0.95	1	0.99	0.96	1
4	0.94	0.91	1	0.93	0.95	1
8	0.96	0.96	1	0.96	0.98	1



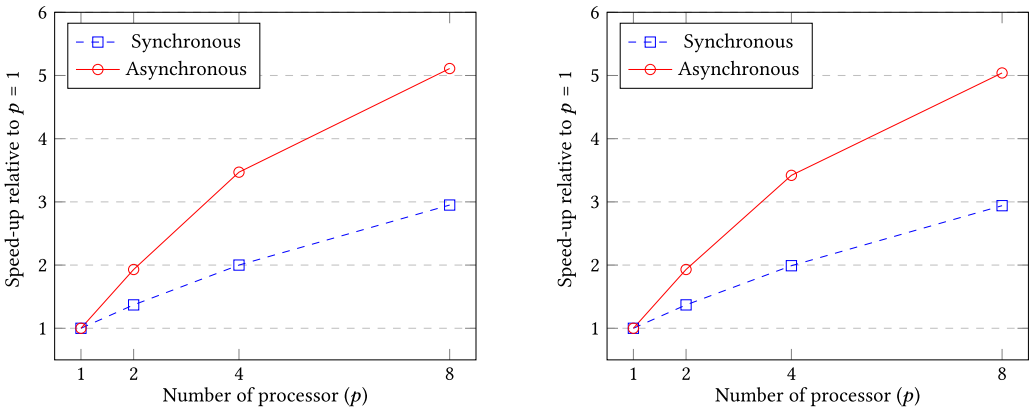
(a) Airline-reservation example (b) M/M/1 queue example

Fig. 1. The credit procedure in the synchronous vs. asynchronous environment in terms of the computation time relative to the serial environment. (The results are similar for the caching procedure.)

after obtaining a certain number of outputs. Figure 1 illustrates the computational benefit of using parallel environments (compared to the serial one) for the credit procedure; we do not exhibit the results for the caching procedure, because it shows a similar performance, nor for APS, because it simply takes too long to run. In particular, the figure reports how much faster the credit procedure is than the gCEI procedure (in the serial environment) for different values of $p \in \{2, 4, 8\}$.

The marginal computational benefit of using a parallel environment decreases as the number of processors increases, because the computational overhead increases as well. In other words, the benefit does not increase linearly in p. When p is larger, the asynchronous version performs better, because the processors in the synchronous environment wait potentially longer for the slowest replication to complete in each batch. Nevertheless, neither of the parallel environments has a substantial advantage over the other, because the simulation execution times do not vary much. Next, we compare the synchronous and asynchronous environment for a case where the simulation execution times vary substantially.

For this comparison, we consider a stylized example with normally distributed simulation outputs. (We explain this stylized example in detail and also use it to provide comprehensive sensitivity analyses in Appendix C.) To control the simulation times in the experiments, we artificially delay obtaining the simulation output. In particular, the simulation execution time for each replication of a system is assumed to follow a bimodal distribution, defined as a mixture of two lognormal distributions with equal weights. The lognormal distributions have means 0.136 and 0.417 and variances 0.014^2 and 0.222^2, respectively. We specify the unit time of simulation execution as the amount of



(a) Caching procedure (b) Credit procedure

Fig. 2. The caching and credit procedures on the slippage configuration with $k = 5$ in the synchronous vs. asynchronous environment in terms of the computation time relative to the serial environment.

time that a processor spends to perform a certain number of elementary mathematical operations (which takes around 1 second in the serial environment). We note that the expected value of the bimodal distribution is approximately 0.276, which is the expected simulation execution time in seconds per replication.

Setting $k = 5$, we ran four experiments in total for the slippage configuration, where all the inferior systems are the same, including four different values of the number of processors, $p \in \{1, 2, 4, 8\}$. In the experiments, we set the number of macro-replications $M = 100$ and $T = 350$, i.e., the procedures terminate after obtaining a certain number of outputs. Figure 2 illustrates the computational benefit of using the asynchronous caching and credit procedures over the respective synchronous versions. Compared to the airline-reservation and M/M/1 queue examples, a clear computational benefit of the asynchronous environment can be seen in this stylized example. It can be attributed to the fact that the simulation execution time is highly variable across replications, making it wasteful to wait for all processors to complete.

7 CONCLUSION

In this article, we tackled a R&S problem in both serial and parallel environments. We first showed that the gCEI procedure, which applies gCEI as the acquisition function in the serial environment, achieves the rate-optimal allocation of Glynn and Juneja [2004] in the limit under the normality assumption.

We also proposed two procedures, caching and credit, that can be combined with any user-selected acquisition function and adapted to both synchronous and asynchronous parallel environments. Both procedures repeatedly apply the acquisition function to choose a single system to simulate for each idle processor. The currently running replications, which are yet to be obtained, are taken into account by considering the corresponding sample means as the persistence forecasts. These procedures differ in the usage of the history of all simulated systems and their replications while choosing systems to simulate. In particular, the caching procedure uses only a subset of the history and caches its complement, whereas the credit procedure uses the whole history. Caching allows the procedure to behave as if it is implemented in a serial environment. Under certain assumptions, the caching procedure extending any acquisition function into the parallel environments achieves the same asymptotic allocation as in the serial environment. A similar

result holds for the credit procedure using gCEI as the acquisition function. The credit procedure empirically performs as well as the caching procedure despite not carefully controlling the history as the caching procedure does.

As parallelizing simulation has gotten easier, parallel R&S has become an active area of inquiry, and many procedures have been created, as noted in Section 1. When considering which procedure to use, issues such as number of systems, computational cost per replication per system, R&S algorithm overhead, communication delays, and the particular parallel architecture all matter, as noted in Hunter and Nelson [2017]. Our caching and credit procedures will be appropriate when the following apply: (a) The computational cost per replication is high, so strategically deciding which system to simulate next dominates all-current-survivors algorithms like KN. (b) The number of systems is small enough (certainly 100 or less) that it is reasonable to keep all systems in play; very large numbers of systems favors eliminating procedures that may be able to discard many systems after they are simulated the first time. (c) The number of parallel processors is modest (say, 64 or less), as we expect persistence forecasts to eventually break down if, for instance, it is needed to forecast hundreds of choices into the future to keep processors busy. Many problems faced by working engineers fit squarely in this domain.

Future research includes modifying the gCEI acquisition function and evaluating its performance when spatial correlations exist among the systems. Implementing the caching and credit procedures for other simulation optimization problems, beyond R&S, is interesting future work.

APPENDICES

A CACHING PROCEDURE WITH LIMITED CACHE SIZE

Let $L_i \geq p$ denote a user-defined limit on the cache size for system i. Also, let C_i denote the cache size (at the end of the batch iteration) for system i. Algorithm 9 presents the BAF for the caching procedure with a cache-size limit. To avoid an infinite loop, the user-selected acquisition function must satisfy a condition: choosing each system infinitely many times in the limit when it is adopted by a serial procedure. A serial procedure could not be based on persistence forecasts.

B PROOFS

We first show the following auxiliary result, which serves as a basis for the proofs of the theorems in this article.

Proposition B.1. *For the credit procedure with $AF_{gCEI}(\cdot)$, $r_i(t) \to \infty$ almost surely as $t \to \infty$ for all $i \in \mathcal{S}$.*

Proof of Proposition B.1. We fix a sample path ω but suppress it in the notation. For each batch iteration $b = 0, 1, \ldots$, recall that $x(t) = AF_{\text{gCEI}}(\mathcal{H}^{bp} \cup \mathcal{F}^t)$ for $t = bp, \ldots, bp + p - 1$. Therefore, the statistics used by the acquisition function at iteration t are conditional on $\mathcal{H}^{bp} \cup \mathcal{F}^t$. To ease notation, we write these statistics as functions of t. For example,

$$r_i(t) = r_i(\mathcal{H}^{bp} \cup \mathcal{F}^t) = \sum_{\tau=0}^{t-1} \mathbb{I}(x(\tau) = i) \text{ and } \bar{Y}_i(t) = \bar{Y}_i(\mathcal{H}^{bp} \cup \mathcal{F}^t) = \frac{1}{r_i(bp)} \sum_{\tau=0}^{bp-1} \mathbb{I}(x(\tau) = i)\, Y(\tau).$$

Let $S = \{i \in \mathcal{S} : r_i(t) \to \infty \text{ as } t \to \infty\}$. Since at least one system must be simulated infinitely often as $t \to \infty$, notice S is non-empty. Assume that $S^c = \mathcal{S} \backslash S$ is also non-empty. Then, the last iteration at which a system in S^c is simulated, $R_1 = \sup\{t + p : x(t-1) \in S^c\}$, is finite. Notice for all $i \in S$ that $\bar{Y}_i(t) \to \mu_i$ by the strong law of large numbers. However, for all $j \in S^c$, $\bar{Y}_j(t) = \bar{Y}_j(R_1)$ and $r_j(t) = r_j(R_1) \leq R_1$ for all $t \geq R_1$.

ALGORITHM 9: BAF for the Caching Procedure with Limited Cache Size in a Synchronous Environment

Input: $\mathcal{C}^{bp}$ and $\mathcal{D}^{bp}$
Output: $x(bp), \ldots, x(bp+p-1)$

Let $t \leftarrow bp$, $\mathcal{C} \leftarrow \mathcal{C}^{bp}$, $\mathcal{D} \leftarrow \mathcal{D}^{bp}$ and $\mathcal{F}^t \leftarrow \emptyset$.
$C_i \leftarrow |\mathcal{C}_i|$, $\forall i \in \mathcal{S}$. ▷ $|\cdot|$ represents the cardinality.
while $t \leq bp+p-1$ **do**
  $i \leftarrow AF(\mathcal{D} \cup \mathcal{F}^t)$.
  while $\mathcal{C}_i \neq \emptyset$ **do** ▷ $\mathcal{C}_i = \{(x(s), Y(s)) \in \mathcal{C} : x(s) = i\}$
    Let $\tau \leftarrow \min\{s : (x(s), Y(s)) \in \mathcal{C}_i\}$.
    Update $\mathcal{C} \leftarrow \mathcal{C} \backslash \{(x(\tau), Y(\tau))\}$ and $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x(\tau), Y(\tau))\}$. ▷ $x(\tau) = i$
    $i \leftarrow AF(\mathcal{D} \cup \mathcal{F}^t)$.
  end while
  if $C_i < L_i$ **then** ▷ Simulating system i will not violate the cache size limit.
    $x(t) \leftarrow i$.
    Update $\mathcal{F}^{t+1} \leftarrow \mathcal{F}^t \cup \{(x(t), \bar{Y}_{x(t)}(\mathcal{D}))\}$ and $t \leftarrow t+1$.
    $C_i \leftarrow C_i + 1$.
  else ▷ Simulating system i will violate the cache size limit.
    Update $\mathcal{F}^t \leftarrow \mathcal{F}^t \cup \{(i, \bar{Y}_i(\mathcal{D}))\}$.
  end if
end while

We first show that there exists an iteration $R_2 \geq R_1$ such that $k(t) \in S$ for all $t \geq R_2$: Assume to the contrary that this is not true. Then there exists a subsequence $(t_n)_{n=1}^{\infty}$ with $t_1 \geq R_1$ on which $k(t_n) \in S^c$. Notice that $k(t_n) = k^c$ is fixed, since $t_n \geq R_1$ so $k^c = \arg\max_{j \in S^c}\{\bar{Y}_j(R_1)\}$. Moreover, for all $j \in S^c \backslash \{k^c\}$, we have $\partial \mathrm{CEI}_j(t_n)/\partial r_j(t_n) = \partial \mathrm{CEI}_j(R_1)/\partial r_j(R_1) = -\varepsilon_j$ for some $\varepsilon_j > 0$, since $\bar{Y}_j(t_n)$, $\bar{Y}_{k^c}(t_n)$, $r_j(t_n)$ and $r_{k^c}(t_n)$ are all fixed for $t_n \geq R_1$. However, for any $i \in S$, $\partial \mathrm{CEI}_i(t_n)/\partial r_i(t_n) \to 0$ as $n \to \infty$ so $\partial \mathrm{CEI}_i(t_n)/\partial r_i(t_n)$ will eventually be greater than $-\max_{j \in S^c \backslash \{k^c\}} \varepsilon_j < 0$. Therefore, there exists an iteration $t_{n'}$ such that

$$\min_{j \in S^c \backslash \{k^c\}} \frac{\partial \mathrm{CEI}_j(t_{n'})}{\partial r_j(t_{n'})} = -\max_{j \in S^c \backslash \{k^c\}} \varepsilon_j < \min_{i \in S} \frac{\partial \mathrm{CEI}_i(t_{n'})}{\partial r_i(t_{n'})}.$$

That is, $g(t_{n'}) \in S^c$. As $k(t_{n'}) \in S^c$ and $g(t_{n'}) \in S^c$, we have $x(t_{n'}) \in S^c$, contradicting the existence of R_1, since $t_{n'} \geq R_1$. Hence, such a subsequence does not exist, and therefore, there exists $R_2 \geq R_1$ such that $k(t) \in S$ for all $t \geq R_2$.

Since $k(t) \in S$ for all $t \geq R_2$, there exists a subsequence $(t_m)_{m=1}^{\infty}$ with $t_1 \geq R_2$ on which $k(t_m) = l$ for some $l \in S$. Notice for all $j \in S^c$ that $\partial \mathrm{CEI}_j(t_m)/\partial r_j(t_m) < -\epsilon_j$ for some $\epsilon_j > 0$, since $r_j(t_m)$ is fixed for $t_m \geq R_2 \geq R_1$. On the other hand, for all $i \in S \backslash \{l\}$, $\partial \mathrm{CEI}_i(t_m)/\partial r_i(t_m) \to 0$ as $m \to \infty$ so it will eventually be greater than $-\max_{j \in S^c} \epsilon_j < 0$. Moreover, $\partial \mathrm{CEI}_i(t_m)/\partial r_l(t_m) \to 0$ for all $i \in \mathcal{S}_l$ as $m \to \infty$ so we will eventually have $\sum_{i \in \mathcal{S}_l} \partial \mathrm{CEI}_i(t_m)/\partial r_l(t_m) > -\max_{j \in S^c} \epsilon_j$. Therefore, there exists an iteration $t_{m'}$ such that

$$\min_{j \in S^c} \frac{\partial \mathrm{CEI}_j(t_{m'})}{\partial r_j(t_{m'})} < -\max_{j \in S^c} \epsilon_j < \min_{i \in S \backslash \{l\}} \frac{\partial \mathrm{CEI}_i(t_{m'})}{\partial r_i(t_{m'})} \quad \text{(i.e., } g(t_{m'}) \in S^c\text{)}$$

and

$$\sum_{i \in \mathcal{S}_l} \frac{\partial \mathrm{CEI}_i(t_{m'})}{\partial r_l(t_{m'})} > -\max_{j \in S^c} \epsilon_j > \min_{j \in S^c} \frac{\partial \mathrm{CEI}_j(t_{m'})}{\partial r_j(t_{m'})} \quad \text{(i.e., } x(t_{m'}) = g(t_{m'})\text{)}.$$

Hence, $x(t_{m'}) = g(t_{m'}) \in S^c$. However, this contradicts the existence of R_1, since $t_{m'} \geq R_2 \geq R_1$. Hence, we conclude that S^c is empty, and thus $S = \mathcal{S}$, i.e., $r_i(t) \to \infty$ as $t \to \infty$ for all $i \in \mathcal{S}$. Since this will occur on almost all sample paths, the convergence is with probability 1. □

Recall that in Section 4.2, we mention that Theorem 1 is a direct consequence of Corollary 2, which follows from Theorem 3. Below, we present the proofs of Theorem 3 and Corollary 2, respectively.

Proof of Theorem 3. We fix a sample path ω but suppress it in the notation. Similar to the proof of Proposition B.1, the statistics used by the acquisition function $AF_{\text{gCEI}}(\cdot)$ at iteration t are conditional on $\mathcal{H}^{bp} \cup \mathcal{F}^t$, but we write these statistics as functions of t to ease notation.

Let ε be a constant such that $0 < \varepsilon < \frac{1}{2}(\mu_k - \mu_{k-1})$. From Proposition B.1, there exists some R_3 such that for all $i \in \mathcal{S}$, we have $|\bar{Y}_i(t) - \mu_i| < \varepsilon$ for $t \geq R_3$ by the strong law of large numbers. By the choice of ε, we have $k(t) = k$ for $t \geq R_3$.

Consider the empirical allocation $(\alpha_1(t), \ldots, \alpha_k(t))$. Since $0 \leq \alpha_i(t) \leq 1$ for each i, we know by the Bolzano-Weierstrass theorem that the allocation must have at least one limit point. We will show that the limit point is unique and is $\boldsymbol{\alpha}^*$. Let $(\alpha_1, \ldots, \alpha_k)$ be an arbitrary limit point of $(\alpha_1(t), \ldots, \alpha_k(t))$. Then, there exists a subsequence $(t_n)_{n=1}^{\infty}$ such that $\alpha_i(t_n) \to \alpha_i$ for all $i \in \mathcal{S}$ as $n \to \infty$. Let $\lambda_i' = (\sigma_i^2/\alpha_i + \sigma_k^2/\alpha_k) = \lim_{n\to\infty} t_n \lambda_i(t_n)$.

Fix an arbitrary pair (i, j) such that $i, j \in \mathcal{S}_k$, and consider a sub-subsequence $(t_{n_m})_{m=1}^{\infty}$ on which $\partial \text{CEI}_i(t_{n_m})/\partial r_i(t_{n_m}) \leq \partial \text{CEI}_j(t_{n_m})/\partial r_j(t_{n_m})$. We know such a sub-subsequence exists because otherwise the gCEI procedure would simulate system i only finitely many times, contradicting Proposition B.1. To simplify notation, we denote the index of the sub-subsequence by τ, i.e., $(\tau)_{\tau=1}^{\infty} \equiv (t_{n_m})_{m=1}^{\infty}$. Then for every τ,

$$\frac{\sigma_i^2/(r_i(\tau))^2}{\sigma_j^2/(r_j(\tau))^2}\sqrt{\frac{\lambda_j(\tau)}{\lambda_i(\tau)}} \times \exp\left\{-\frac{1}{2}\left(\frac{(\bar{Y}_i(\tau) - \bar{Y}_k(\tau))^2}{\lambda_i(\tau)} - \frac{(\bar{Y}_j(\tau) - \bar{Y}_k(\tau))^2}{\lambda_j(\tau)}\right)\right\} \geq 1. \tag{6}$$

Notice that, as $\tau \to \infty$, the non-exponential term will converge to a positive constant, that is,

$$\lim_{\tau\to\infty}\left\{\frac{\sigma_i^2/(r_i(\tau))^2}{\sigma_j^2/(r_j(\tau))^2}\sqrt{\frac{\lambda_j(\tau)}{\lambda_i(\tau)}}\right\} = \lim_{\tau\to\infty}\left\{\frac{\sigma_i^2/(r_i(\tau)/\tau)^2}{\sigma_j^2/(r_j(\tau)/\tau)^2}\sqrt{\frac{\tau\lambda_j(\tau)}{\tau\lambda_i(\tau)}}\right\} = \frac{\sigma_i^2/\alpha_i^2}{\sigma_j^2/\alpha_j^2}\sqrt{\frac{\lambda_j'}{\lambda_i'}} > 0.$$

However, as $\tau \to \infty$, the exponential term will converge to 0 unless

$$\lim_{\tau\to\infty}\left\{\frac{(\bar{Y}_i(\tau) - \bar{Y}_k(\tau))^2}{\tau\lambda_i(\tau)} - \frac{(\bar{Y}_j(\tau) - \bar{Y}_k(\tau))^2}{\tau\lambda_j(\tau)}\right\} = \frac{(\mu_i - \mu_k)^2}{\lambda_i'} - \frac{(\mu_j - \mu_k)^2}{\lambda_j'} \leq 0. \tag{7}$$

We note that the sample means converge to the true means, because they include only a finite number of (at most $p-1$) persistence forecasts at each iteration in addition to all the corresponding outputs obtained up to the current batch iteration.

Next, consider another sub-subsequence on which $\partial \text{CEI}_i(\tau)/\partial r_i(\tau) \geq \partial \text{CEI}_j(\tau)/\partial r_j(\tau)$; we again know that such a sub-subsequence exists from Proposition B.1. This reverses the inequality in (6). Then a similar argument shows that the left-hand side of Equation (7) must be ≥ 0 in the limit, because the exponential term will go to ∞ as $\tau \to \infty$ otherwise. Therefore, equality is required:

$$\frac{(\mu_i - \mu_k)^2}{\lambda_i'} - \frac{(\mu_j - \mu_k)^2}{\lambda_j'} = 0. \tag{8}$$

Since this equality holds for arbitrary $i, j \in \mathcal{S}_k$, the limit point $(\alpha_1, \ldots, \alpha_k)$ satisfies the first of two conditions for the rate-optimal allocation of Glynn and Juneja [2004].

Similarly, for arbitrary $j \in \mathcal{S}_k$, consider a sub-subsequence (that exists from Proposition B.1) on which $\sum_{i\in\mathcal{S}_k} \partial \mathrm{CEI}_i(\tau)/\partial r_k(\tau) \le \partial \mathrm{CEI}_j(\tau)/\partial r_j(\tau)$. For such indices,

$$\sum_{i\in\mathcal{S}_k} \frac{\sigma_k^2/(r_k(\tau))^2}{\sigma_j^2/(r_j(\tau))^2}\sqrt{\frac{\lambda_j(\tau)}{\lambda_i(\tau)}} \times \exp\left\{-\frac{1}{2}\left(\frac{(\bar{Y}_i(\tau)-\bar{Y}_k(\tau))^2}{\lambda_i(\tau)} - \frac{(\bar{Y}_j(\tau)-\bar{Y}_k(\tau))^2}{\lambda_j(\tau)}\right)\right\} \ge 1. \tag{9}$$

Notice that, as $\tau \to \infty$, the exponential term will converge to 1 due to Equation (8). Thus,

$$\lim_{\tau\to\infty}\left\{\frac{\sigma_k^2/(r_k(\tau))^2}{\sigma_j^2/(r_j(\tau))^2}\sum_{i\in\mathcal{S}_k}\sqrt{\frac{\lambda_j(\tau)}{\lambda_i(\tau)}}\right\} = \frac{\sigma_k^2/(\alpha_k)^2}{\sigma_j^2/(\alpha_j)^2}\sum_{i\in\mathcal{S}_k}\sqrt{\frac{\lambda_j'}{\lambda_i'}} \ge 1. \tag{10}$$

Next, consider another sub-subsequence (that again exists from Proposition B.1) on which $\sum_{i\in\mathcal{S}_k} \partial \mathrm{CEI}_i(\tau)/\partial r_k(\tau) > \partial \mathrm{CEI}_j(\tau)/\partial r_j(\tau)$. This reverses the inequality in (9). Then a similar argument shows that the left-hand side of Equation (10) must be ≤ 1. Therefore, equality is required in the limit:

$$\frac{\sigma_k^2/(\alpha_k)^2}{\sigma_j^2/(\alpha_j)^2}\sum_{i\in\mathcal{S}_k}\sqrt{\frac{\lambda_j'}{\lambda_i'}} = 1.$$

Since this equality holds for arbitrary $j \in \mathcal{S}_k$, after some rearrangement, and by summing both sides over $j \in \mathcal{S}_k$, we obtain

$$\sum_{j\in\mathcal{S}_k}\left(\frac{\alpha_j}{\sigma_j}\right)^2\sum_{i\in\mathcal{S}_k}\sqrt{\frac{1}{\lambda_i'}} = \left(\frac{\alpha_k}{\sigma_k}\right)^2\sum_{j\in\mathcal{S}_k}\sqrt{\frac{1}{\lambda_j'}}.$$

Dividing out the common term gives

$$\sum_{j\in\mathcal{S}_k}\left(\frac{\alpha_j}{\sigma_j}\right)^2 = \left(\frac{\alpha_k}{\sigma_k}\right)^2,$$

which is the second condition of Glynn and Juneja [2004]. And, since this will occur on almost all sample paths, the convergence is with probability 1. □

Proof of Corollary 2. First, notice that Proposition B.1 will hold provided that $\hat{\sigma}_i^2(t) > 0$ for all $i \in \mathcal{S}$. Then, we know by the strong law of large numbers that $\hat{\sigma}_i^2(t) \to \sigma_i^2$ almost surely for all $i \in \mathcal{S}$ as $t \to \infty$. Thus, Theorem 1 holds from the continuous mapping theorem. □

Recall that Section 4.1 discusses the proof of Theorem 2.

C SENSITIVITY ANALYSIS

Using a stylized example, we provide comprehensive sensitivity analyses of the proposed procedures in the following subsections. Each system $i \in \mathcal{S}$ has normally distributed simulation outputs with mean $\mu_i = c\eta_i$, where η_i is a prescaled true mean value provided in Table 6 and c is a scaling constant explained below. All systems' outputs are independent from each other. Among the four configurations in Table 6, all systems have equal variances in the slippage and ascending means configurations. In the other two configurations the means are ascending, but the variances are proportional and inversely proportional to the prescaled mean values.

In every experiment, we first allocate 10 replications to each system before applying any procedure. To create sensible cases, we scale the true means so at least r_0 replications are consumed before the difference between the best and second-best systems matches the standard error of their

Table 6. Configurations for Experiments with Normally Distributed Outputs

Configuration	Prescaled true mean values	True standard deviations
Slippage	$\eta_i = -1$ for $i \in \mathcal{S}_k$ and $\eta_k = 0$	$\sigma_i = 1$
Ascending mean	$\eta_i = \log(i)$	$\sigma_i = 1$
Ascending variance	$\eta_i = \log(i+1)$	$\sigma_i = \sqrt{\eta_i}$
Descending variance	$\eta_i = \log(i+1)$	$\sigma_i = 1/\sqrt{\eta_i}$

estimated difference if r_0 replications are allocated according to the rate-optimal ratios in Glynn and Juneja [2004]. Specifically,

$$\mu_k - \mu_{k-1} = c(\eta_k - \eta_{k-1}) = \sqrt{\frac{\sigma_{k-1}^2}{r_0 \alpha_{k-1}^*} + \frac{\sigma_k^2}{r_0 \alpha_k^*}}. \tag{11}$$

Namely, we control how quickly the best system becomes distinguishable from the others. To find c satisfying Equation (11), we first calculate $\boldsymbol{\alpha}^*$ by solving the expressions in Equations (4) and (5) with the η_i's from Table 6. The constant c does not change the optimal allocation, because scaling all μ_i's does not affect $\boldsymbol{\alpha}^*$. We set $r_0 = 20k$.

The procedures stop when the stopping criterion introduced in Section 2 is met for the confidence level $P^* = 0.95$. We consider different levels of tolerance. In particular, for any $n = 1, \ldots, k-1$, we set $\delta_n = \mu_k - \frac{1}{2}(\mu_{k-n} + \mu_{k-n+1})$ so the top n systems are considered as good. Using δ_n, we introduce

$$\widehat{\mathrm{PGS}}_n = \frac{1}{M}\sum_{m=1}^{M} \mathbb{I}(\mu_{k(T_m)} > \mu_k - \delta_n) = \frac{1}{M}\sum_{m=1}^{M} \mathbb{I}(k(T_m) \in \{k, \ldots, k-n+1\})$$

to report the estimate of PGS by averaging the good selection across the macro-replication; we set the number of macro-replications $M = 5{,}000$.

Although we ran several experiments for each comparison, in the following subsections, we report the results for a subset of experiments, because the results for the other experiments are similar to ones that we report unless otherwise stated.

C.1 Comparison of Acquisition Functions: gCEI vs. mCEI

First, we compare the performances of $AF_{\mathrm{gCEI}}(\cdot)$ and $AF_{\mathrm{mCEI}}(\cdot)$ in Algorithms 4 and 5, i.e., under the caching and credit procedures in the synchronous environment. For this comparison, setting $k = 20$ and $p = 4$, we ran experiments for all four configurations in Table 6 but only present one representative result.

Table 7 exhibits results for the slippage configuration. Both caching and credit procedures terminate earlier when the gCEI acquisition function is used. Despite the earlier termination, gCEI yields slightly larger PGS. This result demonstrates the efficiency and effectiveness of gCEI as an acquisition function in the parallel environment. Further, it emphasizes the importance of the choice of the acquisition function for the performance of the caching and credit procedures.

We note that the procedures achieve the PGS target of 0.95 regardless of the acquisition function adapted. This target is also achieved in all experiments with normally distributed outputs, which was not the case in the airline-reservation and M/M/1 queue examples where the simulation outputs are non-normal. This emphasizes the importance of the choice of the stopping criterion depending on the problem characteristics for the performance of the procedures in terms of good selection.

Table 7. gCEI vs. mCEI Acquisition Function on the Slippage Configuration with Tolerance δ_1 in the Synchronous Environment

	Caching procedure		Credit procedure	
	gCEI	mCEI	gCEI	mCEI
$\widehat{T}_{mean}$	1,410.8	2,443.1	1,398.9	2,434.4
$\widehat{T}_{se}$	8.4	66.2	8.0	66.8
$\widehat{T}_{median}$	1,324	1,536	1,308	1,528
$\widehat{T}_{max}$	9,064	103,380	8,832	97,044
$\widehat{\text{PGS}}_1$	0.994	0.978	0.995	0.976

Table 8. Caching (Cache) vs. Credit (Crdt) Procedure on the Ascending Variance Configuration with Tolerance δ_2 and $k = 20$ in the Synchronous Environment

	$p = 1$	$p = 4$		$p = 8$		$p = 16$		$p = 32$		$p = 64$	
	gCEI	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt	Cache	Crdt
$\widehat{T}_{mean}$	392.4	404.2	404.1	414.1	412.3	429.9	427.1	451.3	452.4	490.4	492.4
$\widehat{T}_{se}$	2.0	2.0	2.0	2.1	2.0	2.1	2.1	2.1	2.2	2.3	2.3
$\widehat{T}_{median}$	370	384	380	392	392	408	408	424	424	456	456
$\widehat{T}_{max}$	1,095	1,172	1,020	1,032	1,320	1,096	1,112	1,160	1,192	1,224	1,288
$\widehat{\text{PGS}}_2$	0.960	0.965	0.969	0.972	0.967	0.970	0.972	0.978	0.977	0.981	0.981

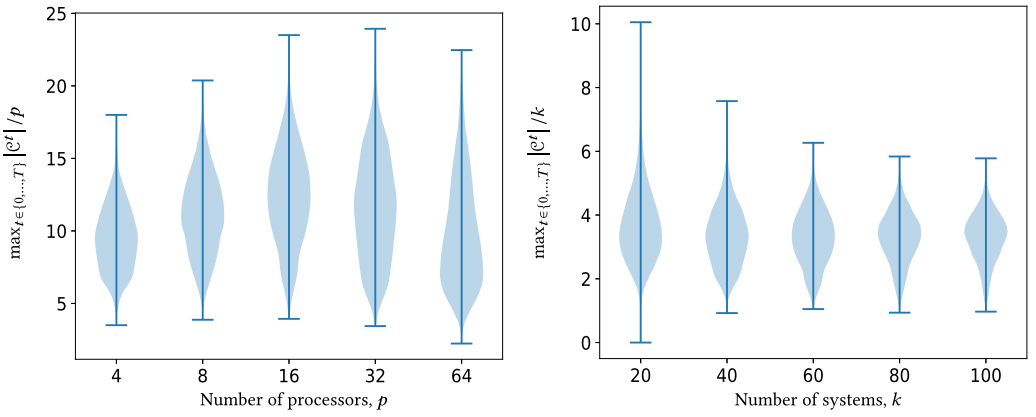
C.2 Credit vs. Caching Procedure

Next, we compare the caching and credit procedures (using the gCEI acquisition function) in the synchronous environment in terms of the total number of outputs obtained and the PGS. Also, we investigate the cache size behavior under the caching procedure for different number of processors. We ran 120 experiments in total for five different values of number of systems $k \in \{20, 40, 60, 80, 100\}$, six different values of number of processors $p \in \{1, 4, 8, 16, 32, 64\}$, and all four configurations in Table 6. Recall that both caching and credit procedures reduce to the gCEI procedure when $p = 1$.

As a representative example, Table 8 exhibits results for the ascending variance configuration with tolerance δ_2 (i.e., the top two systems are considered good) and $k = 20$. Based on T and PGS, both procedures show similar performance. A possible explanation is that the procedures tend to adhere closely to the desired path, as an increases in p does not lead to a significant increase in T for either procedure. This indicates that the persistence forecast is effective at predicting the desired path.

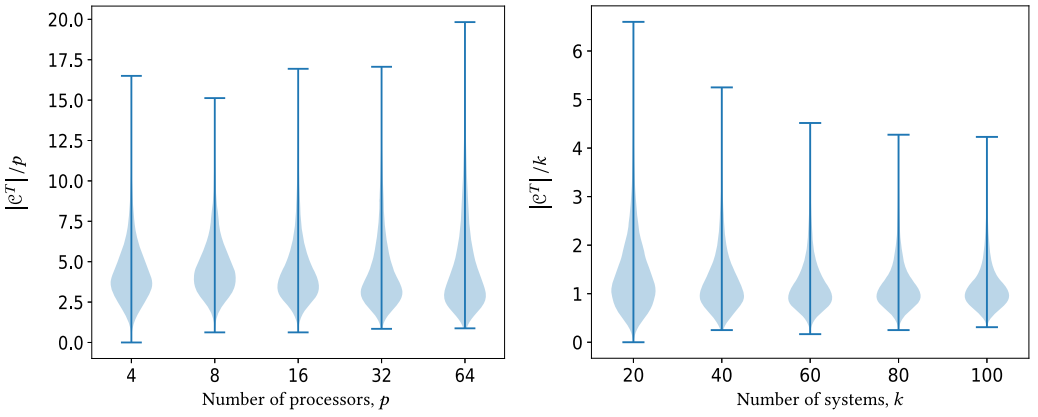
To investigate how the achieved PGS changes as the number of iterations increases, we also ran the same set of 120 experiments under the fixed-budget framework by setting $T = 200k$, i.e., the procedures terminate after obtaining a certain number of outputs. We do not show any results from these experiments but mention that they did not provide any evidence to conclude that either procedure empirically outperforms the other.

Finally, to evaluate how well the caching procedure builds the desired path, Figures 3 and 4 visualize the maximum cache size at any iteration and cache size at termination, respectively, for the slippage configuration by using a violin plot, a combination of a box plot with the addition of a rotated kernel density plot on each side. The violin plots in parts (a) and (b) of these figures show the corresponding cache sizes per processor for a fixed



(a) $k = 60$ (b) $p = 16$

Fig. 3. Maximum cache size per processor/system for the slippage configuration with tolerance δ_1.



(a) $k = 60$ (b) $p = 16$

Fig. 4. Cache size per processor/system at termination for the slippage configuration with tolerance δ_1.

number of systems $k = 60$ and per system for a fixed number of processors $p = 16$, respectively. The maximum cache size per processor first increases and then decreases with the increase in p, whereas the maximum cache size per system is not significantly affected by the change in k, except for a few outliers. The cache size per system/processor at termination shows similar behavior but is less affected by the change in the number of processors/systems. We also observe similar behavior of the cache size per system/processor for the experiments with different configurations. However, the cache sizes are not as large as in the slippage configuration, where all the inferior systems are identical.

REFERENCES

Harun Avci, Barry L. Nelson, and Andreas Wächter. 2021. Getting to "Rate-optimal" in ranking & selection. In *Proceedings of the Winter Simulation Conference*. Retrieved from https://www.informs--sim.org/wsc21papers/236.pdf

Jürgen Branke, Stephen E. Chick, and Christian Schmidt. 2005. New developments in ranking and selection: An empirical comparison of the three main approaches. In *Proceedings of the Winter Simulation Conference*. 708–717.

Jürgen Branke, Stephen E. Chick, and Christian Schmidt. 2007. Selecting a selection procedure. *Manag. Sci.* 53, 12 (2007), 1916–1932.

Chun-Hung Chen, Jianwu Lin, Enver Yücesan, and Stephen E. Chick. 2000. Simulation budget allocation for further enhancing the efficiency of ordinal optimization. *Discr. Event Dynam. Syst.* 10, 3 (2000), 251–270.

E. Jack Chen. 2005. Using parallel and distributed computing to increase the capability of selection procedures. In *Proceedings of the Winter Simulation Conference*. 723–731.

Ye Chen and Ilya O. Ryzhov. 2019. Complete expected improvement converges to an optimal budget allocation. *Adv. Appl. Probabil.* 51, 1 (2019), 209–235.

Peter I. Frazier, Warren B. Powell, and Savas Dayanik. 2008. A knowledge-gradient policy for sequential information collection. *SIAM J. Contr. Optim.* 47, 5 (2008), 2410–2439.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. 2007. A multi-points criterion for deterministic parallel global optimization based on Kriging. In *Proceedings of the International Conference on Non-Convex Programming*.

Peter Glynn and Sandeep Juneja. 2004. A large deviations perspective on ordinal optimization. In *Proceedings of the Winter Simulation Conference*. 577–585.

David Goldsman, Barry L. Nelson, and Bruce Schmeiser. 1991. Methods for selecting the best system. In *Proceedings of the Winter Simulation Conference*. 177–186.

L. Jeff Hong, Weiwei Fan, and Jun Luo. 2021. Review on ranking and selection: A new perspective. *Front. Eng. Manag.* 8, 3 (2021), 321–343.

Susan R. Hunter and Barry L. Nelson. 2017. Parallel ranking and selection. In *Advances in Modeling and Simulation*. Springer, 249–275.

Kevin Jamieson and Robert Nowak. 2014. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *Proceedings of the 48th Annual Conference on Information Sciences and Systems (CISS'14)*. 1–6.

Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient global optimization of expensive black-box functions. *J. Global Optim.* 13, 4 (1998), 455–492.

Bogumił Kamiński and Przemysław Szufel. 2018. On parallel policies for ranking and selection problems. *J. Appl. Stat.* 45, 9 (2018), 1690–1713.

Seong-Hee Kim and Barry L. Nelson. 2001. A fully sequential procedure for indifference-zone selection in simulation. *ACM Trans. Model. Comput. Simul.* 11, 3 (2001), 251–273.

Jun Luo, L. Jeff Hong, Barry L. Nelson, and Yang Wu. 2015. Fully sequential procedures for large-scale ranking-and-selection problems in parallel computing environments. *Oper. Res.* 63, 5 (2015), 1177–1194.

Yuh-Chuyn Luo, Chun-Hung Chen, E. Yücesan, and Insup Lee. 2000. Distributed web-based simulation optimization. In *Proceedings of the Winter Simulation Conference*, Vol. 2. 1785–1793.

Barry L. Nelson and Jeremy Staum. 2006. Control variates for screening, selection, and estimation of the best. *ACM Trans. Model. Comput. Simul.* 16, 1 (2006), 52–75.

Eric C. Ni, Dragos F. Ciocan, Shane G. Henderson, and Susan R. Hunter. 2017. Efficient ranking and selection in parallel computing environments. *Oper. Res.* 65, 3 (2017), 821–836.

Linda Pei, Barry L. Nelson, and Susan R. Hunter. 2018. A new framework for parallel ranking & selection using an adaptive standard. In *Proceedings of the Winter Simulation Conference*. 2201–2212.

Linda Pei, Barry L. Nelson, and Susan R. Hunter. 2020. Evaluation of bi-PASS for parallel simulation optimization. In *Proceedings of the Winter Simulation Conference*. 2960–2971.

Linda Pei, Barry L. Nelson, and Susan R. Hunter. 2022. Parallel adaptive survivor selection. *Oper. Res.* (2022). DOI: https://doi.org/10.1287/opre.2022.2343

Yijie Peng and Michael C. Fu. 2017. Myopic allocation policy with asymptotically optimal sampling rate. *IEEE Trans. Automat. Contr.* 62, 4 (2017), 2041–2047.

Daniel Russo. 2020. Simple Bayesian algorithms for best-arm identification. *Oper. Res.* 68, 6 (2020), 1625–1647.

Peter L. Salemi, Eunhye Song, Barry L. Nelson, and Jeremy Staum. 2019. Gaussian Markov random fields for discrete optimization via simulation: Framework and algorithms. *Oper. Res.* 67, 1 (2019), 250–266.

Mark Semelhago, Barry L. Nelson, Eunhye Song, and Andreas Wächter. 2022. Object-oriented implementation and parallelization of the rapid Gaussian Markov improvement algorithm. In *Proceedings of the Winter Simulation Conference*. IEEE, 3158–3169.

David Slepian. 1962. The one-sided barrier problem for Gaussian noise. *Bell Syst. Technic. J.* 41, 2 (1962), 463–501.

Taejong Yoo, Hyunbo Cho, and Enver Yücesan. 2009. Web services-based parallel replicated discrete event simulation for large-scale simulation optimization. *Simulation* 85, 7 (2009), 461–475.

Enver Yücesan, Yuh-Chuyn Luo, Chun-Hung Chen, and Insup Lee. 2001. Distributed web-based simulation experiments for optimization. *Simul. Pract. Theor.* 9, 1-2 (2001), 73–90.

Ying Zhong and L. Jeff Hong. 2022. Knockout-tournament procedures for large-scale ranking and selection in parallel computing environments. *Oper. Res.* 70, 1 (2022), 432–453.

Received 29 December 2022; revised 22 August 2023; accepted 29 August 2023