Parallel Ranking and Selection

Susan R. Hunter and Barry L. Nelson

Abstract The Winter Simulation Conference serves as the initial publication venue for many advances in ranking and selection (R&S), including the recently-developed R&S procedures that exploit high-performance parallel computing. We formulate a new stylized model for representing parallel R&S procedures, and we provide an overview of existing R&S procedures under the stylized model. We also discuss why designing R&S procedures for a parallel computing platform is nontrivial and speculate on the future of parallel R&S procedures. In this chapter, "parallel computing" means multiple processors that can execute distinct simulations independently, rather than vector or array processors designed to speed up vector-matrix calculations.

1 Introduction

The term *Ranking and Selection (R&S)* broadly refers to solution methods developed to solve the R&S problem. The R&S problem is a stochastic optimization problem in which the decision-maker wishes to choose the "best" among a finite set of design points, or "systems," when the performance of each system can only be observed with error. The R&S problem can be considered a special case of the more general simulation optimization (SO) problem, which is a (usually) nonlinear optimization problem whose objectives and constraints, if present, can only be observed with error as output from a stochastic simulation (see, e.g., Pasupathy and Ghosh [52] and Fu [20] for overviews). Among SO problems, the R&S problem is unique in the sense that it engenders interesting research questions only in the stochastic context: The deterministic black-box analogue of the R&S problem

Susan R. Hunter
Purdue University, West Lafayette, IN, USA e-mail: susanhunter@purdue.edu

Barry L. Nelson
Northwestern University, Evanston, IL USA, e-mail: nelsonb@northwestern.edu

is complete enumeration. In contrast, when the system performance measures are defined implicitly through a black-box stochastic simulation model, the decision-maker can only observe each system's performance by constructing an estimator whose precision depends on the simulation budget expended. In this context, interesting methodological questions arise. For example, two key questions are (a) how does one guarantee that at the end of simulating, the estimated best system is truly the best system with high probability, and (b) how does one allocate a finite simulation budget across systems to efficiently identify the best system? For more than sixty years, researchers have sought answers to these questions, resulting in a large body of R&S literature. For overviews and entry points into this literature, see Bechhofer et al. [4] and Gupta and Panchapakesan [26] for origins, and Goldsman and Nelson [25], Kim and Nelson [36], and Branke et al. [6] for the stochastic simulation perspective.

R&S was originally developed by the statistics community during the 1950's, 1960's, and 1970's [3, 14, 15, 54, 57], but following its appearance at the Winter Simulation Conference (WSC), the nexus of research shifted from the statistics community to the stochastic simulation community sometime in the 1980's and 1990's. There were two key reasons for this shift: (a) optimizing a function embedded in a stochastic simulation was a natural goal for simulation practitioners, and early SO researchers borrowed existing methods from the statistics community; and (b) efficiency in R&S often comes from sequential sampling and comparison of systems, and the barrier to sequential algorithms was far lower in computer experiments than in the industrial and biostatistics applications for which R&S was invented. Further, while the statistics community was often concerned with having procedures for different (non-normal) populations, the emphasis in the simulation community was on designing procedures for larger and larger numbers of alternatives, with normality being plausible due to averaging, e.g., via batch means [59].

During this shift, WSC became a key venue joining the statistics and simulation communities. To the best of our knowledge, R&S first appeared at WSC in a 1976 session entitled, "Statistical Basis for Selection Among Alternatives," which contained the work of Turnquist and Sussman [61] and Dudewicz [13]. R&S became an increasingly popular topic after Goldsman published a survey paper in the 1983 WSC *Proceedings* [24]. Since then, WSC has served as the initial publication venue for many key advances in the R&S literature, with the area still active at WSC 2016 (e.g., [12]).

The significant advances in computing power over the last forty years, and particularly the recent proliferation of parallel computing platforms, is one reason why R&S is still an active research topic at WSC today.[1] Originally developed with serial computing platforms in mind, R&S procedures are now being redesigned for deployment as parallel procedures — a surprisingly nontrivial endeavor. Serial R&S procedures of the 1990's and early 2000's measure efficiency as the total number of

[1] In this paper "parallel computing platform" means multiple processors that can independently execute simulation experiments and communicate with each other via message passing or shared memory. We use the term "processors" to refer to cores or threads that can complete computing tasks, so the total number of processors is cores $\times$ (threads / core).

simulation observations required on a single processor, ensure efficiency by being fully sequential, and tend to work well when the number of systems is small. On a parallel platform, appropriate efficiency measures include processor utilization, wall-clock time, and monetary cost to rent processors. Fully sequential procedures may require bottleneck-inducing synchronization. Further, while today's computing power ensures we can solve small problems fast, it should also enable us to solve much bigger problems. Thus parallel computing platforms require procedures that minimize new measures of efficiency, guard against the bottlenecks that can arise in a parallel setting, and can handle a large number of systems. A handful of such parallel R&S procedures exist; all have made their debut at WSC.

Since R&S was originally designed to select among a small number of categorial or unordered alternatives, one may wonder, when do large R&S problems arise? First, large problems with categorical choices arise naturally in some applications, such as drug discovery and plant breeding. The respective goals in these applications are to find the best drug molecule among many potential drug molecules [42], and to find the best plant breeding pairs to produce a progeny population with desirable properties [32]. Second, problems with a very-large-but-finite number of alternatives on an ordered space would seem to be most naturally solved by using algorithms that can exploit the spatial structure, such as R-SPLINE [62] or COMPASS [29, 63]. However, because of its simplicity and ability to provide a statistical guarantee that the selected system is truly the global best, R&S is often a go-to method for practitioners. For example, R&S can be applied to large problems that are created by considering all feasible combinations of a set of decision variables. Xu et al. [63] describe a SO problem with 500^6 feasible solutions obtained by considering all 500 possible values of 6 order-up-to levels in a supply chain problem. A characteristic of such problems is that many (most) of the feasible solutions are substantially inferior to the better ones, and R&S procedures can exploit this tendency. Even when a search algorithm such as R-SPLINE or COMPASS is employed first, R&S can be used to provide a statistical guarantee as to which of the visited solutions is the best [5].

In this chapter, we discuss the current state of the art in R&S for large problems solved on parallel computing platforms. We assume the reader is familiar with serial R&S procedures, at a broad level. In rethinking R&S procedures for parallel implementation, we provide a new stylized model for representing parallel R&S methods (§2), discuss mathematical and computational formulations of existing serial R&S procedures under the stylized model (§3 and §4, respectively), discuss design principles for efficiency and validity of parallel R&S procedures (§5), discuss existing parallel R&S procedures (§6), and speculate on the future of parallel R&S procedures (§7 and §8).

1.1 Problem Setting and Notational Conventions

R&S addresses the following SO problem: Let the true expected performances of the k competing systems be denoted

$$\mu_1 \leq \mu_2 \leq \cdots \leq \mu_{k-1} \leq \mu_k,$$

where a larger mean is better, and let $\mathcal{S} = \{1, 2, \ldots, k\}$ denote the set of indices of all systems. We refer to system k, or any system tied with system k, as the best. Often, the best system is assumed to be unique, in which case $\mu_{k-1} < \mu_k$. Recall that we are unable to observe the true expected performances directly. Then suppose we are given a simulation oracle that can provide us with random variables $Y_{i1}, Y_{i2}, \ldots, Y_{in}$, where Y_{ir} is a random variable representing the performance of system i on the rth simulation replication, $r = 1, 2, \ldots, n$, $i \in \mathcal{S}$. For all systems $i \in \mathcal{S}$, we estimate the value of μ_i with a consistent estimator such as the sample mean $\bar{Y}_i(n) := \sum_{r=1}^{n} Y_{ir}/n$.

An *R&S procedure* is an algorithm that attempts to return the best system using only the estimators of the expected system performances. In this chapter the estimators of the expected system performances after obtaining $n_i \geq 1$ simulation replications from each system $i \in \mathcal{S}$ are $\{\bar{Y}_i(n_i) \colon i \in \mathcal{S}\}$. We assume an R&S procedure returns the system with the largest estimated mean as the estimated best system, so that $\hat{K} := \text{argmax}_{i \in \{1,2,\ldots,k\}} \{\bar{Y}_i(n_i) \colon i \in \mathcal{S}\}$ is the estimated best system. (See, e.g., [55] for an example in which a system other than the one with the largest estimated mean is returned.) Since we may only assess each system with a finite computational budget, there is always a positive probability that an R&S procedure will return some system other than the best. Thus R&S procedures are usually created to satisfy some form of mathematical or statistical objectives, which we discuss in §3.

1.2 Scope

We classify as R&S any procedures that include three key ingredients: (a) they are applied to a finite number of systems whose expected performance can only be observed with error as simulation output, (b) the procedure will simulate all of the systems and construct consistent estimators of their expected performance, and (c) the decision-maker wishes to select the best by comparing these systems to each other. While we consider only the single-objective problem formulation, we note that stochastically constrained and multi-objective versions of the R&S problem exist. For example, see [1, 53] for the stochastically constrained case and [17, 18, 31, 37] for the multi-objective case.

R&S is closely related to some versions of best-arm identification in stochastic multi-armed bandit (MAB) problems; see Bubeck and Cesa-Bianchi [7] and Jamieson and Nowak [33]. However, there are differences: MAB is often concerned with online decision-making so as to accumulate the most reward, while R&S is

always an offline optimization problem. Perhaps more critically, the key assessment of an MAB algorithm is its "big-O" computational complexity (convergence rate) when selecting the best, while R&S is concerned with finite-time performance, even when asymptotic methods are used in the analysis. As a result, MAB algorithms tend to be simple, have few distribution-specific assumptions, and their computational complexity is determined up to some unknown constants; as compared to R&S that tries to exploit specific distributions to gain efficiency and to assure validity even when unknown constants must be estimated. We focus exclusively on R&S.

2 A Stylized Computational Model for R&S

Throughout this chapter, we use a stylized computational model to facilitate our discussion of both serial and parallel R&S procedures. In this section, we define and discuss the stylized model.

We formulate a stylized computational model for parallel R&S procedures by breaking all simulation and calculation tasks that must be completed during an R&S procedure into *jobs*. All R&S procedures contain two primary tasks for processors to complete: (a) performing simulation replications, and (b) calculations completed after simulation replication output is obtained, such as comparing the performances of systems to each other to select the estimated best. Thus we define job j as the ordered list comprised of obtaining simulation replications and performing calculations,

$$J_j := \{(\mathcal{Q}_j, \Delta_j, \mathcal{U}_j), (\mathcal{P}_j, \mathcal{C}_j)\},$$

where

- $\mathcal{Q}_j \subseteq \mathcal{S}$ a set containing the indices of systems to be simulated;
- $\Delta_j = \{\Delta_{ij}\}$ specifies how many samples to take from each system $i \in \mathcal{Q}_j$;
- $\mathcal{U}_j$ is the assigned block of random numbers with which to perform the simulation replications;
- $\mathcal{P}_j$ is a list of jobs whose termination must precede the calculation $\mathcal{C}_j$, if any, and
- $\mathcal{C}_j$ is a list of non-simulation calculations or operations to perform.

We allow $(\mathcal{Q}_j, \Delta_j, \mathcal{U}_j)$ or $(\mathcal{P}_j, \mathcal{C}_j)$ to be null, so that a job can consist of just simulations or just calculations. Since J_j is ordered, we assume that the simulation replications in $(\mathcal{Q}_j, \Delta_j, \mathcal{U}_j)$ are completed before the calculation $\mathcal{C}_j$ begins. In the presence of only one processor, the list of jobs is usually created and performed dynamically by a single processor. In the presence of multiple processors, the list of jobs must be coordinated to preserve precedence requirements. For example, some simulation replications must be obtained from each system before their performances can be compared to each other.

When the number of processors $p \geq 2$, we broadly assume that parallel algorithms operate in what is known as a *master-worker* framework. In this framework, one *master* processor coordinates the activities of one or more *worker* processors. The workers execute jobs determined by the master, and report results back to the

master. Communication may occur through shared memory or via message passing. A master-worker framework can also be implemented in multiple tiers, in which each worker acts as a master to, and coordinates the tasks of, one or more sub-workers. For simplicity and ease of exposition, we assume only one such tier for now.

Remark 1. We acknowledge the existence of various parallel computing architectures and frameworks for parallel algorithm design (see, e.g., [2]). We take a higher-level approach that enables us to focus on broad R&S procedure design concepts, instead of the details related to the underlying parallel computing architecture.

When a master sends a job to a worker, we assume that all data required to do the job is also transferred, or is otherwise accessible by shared memory. Likewise, when a worker completes a task, we assume relevant data is transferred back to the master. Ensuring efficient data transfer is an important part of designing parallel algorithms, however for exposition, we suppress data transfer information in our framework. Thus while a worker's job may entail performing simulation replications and calculating statistics such as a sample mean, we do not explicitly denote whether the worker transfers just the sample mean back to the master, or the sample mean and all data used to compute the sample mean.

In the master-worker framework, we assume the (possibly dynamic) list of jobs

$$\mathcal{J} := \{J_j : 1 \leq j \leq M\},$$

is created and maintained by the master processor, where job $1 \leq M \leq \infty$ is some (possibly random) terminal job; $M = \infty$ denotes the list of jobs for a non-terminating algorithm. When a worker processor completes a job or becomes idle, it communicates any results back to the master processor and requests a new job. Henceforth, let $0 < T_j < \infty$ be the wall-clock time that job J_j finishes, so that

$$T_e(\mathcal{J}) = \max_{j=1,2,\ldots,M} T_j$$

is the (possibly random) ending time of the procedure.

Remark 2. We assume the master creates jobs that can be sent to the workers for execution. Some jobs, particularly jobs containing only calculations, may be executed by the master. Since only the master creates jobs, we do not consider the creation of jobs to be a job.

In modern computing environments, R&S procedures may be completed by purchasing processing power from a service. Since cores may often be purchased in increments such as 4, 8, 16, 40, or 64, with the price per hour varying by the type of processing power provided, we formulate the general cost to purchase p processors for s time units as a function $c(p,s)$. For a total budget b, we require $c(p,s) \leq b$. Define the function $t(p,b)$ as the maximum amount of time we purchase on p processors, so that

$$t(p,b) := \max\{s\colon c(p,s) \leq b\}.$$

3 Mathematical Formulations of Existing R&S Procedures

Recall that because we cannot simulate every system infinitely often, upon termination, R&S procedures have some positive probability of selecting a system other than the true best. However, most R&S procedures are designed to control this error probability. In this section, we formulate the common goals of existing R&S procedures using the stylized model in §2.

First, we note that most R&S procedures are in some way concerned with the optimality gap between the true best system and the estimated best system, $\mu_k - \mu_{\hat{K}}$. We say that a *correct selection* (CS) event occurs if this optimality gap is zero, and $\mu_k = \mu_{\hat{K}}$. *An ideal R&S procedure would always deliver a CS for any computational budget* $n \geq k$. Since this ideal is impossible in the presence of noise, compromises are made, and the chosen compromise affects the nature of the procedure. R&S procedures may be classified by a number of different approaches and compromises, although these boundaries are not always sharp (see also [12, 52]):

Fixed-precision vs. fixed-budget guarantee: Fixed-precision procedures execute until some form of guarantee holds, usually on the optimality gap between the selected and true best systems. Fixed-budget procedures attempt to allocate a fixed computational budget in a way that minimizes a loss function that penalizes an incorrect selection event.

Finite-sample vs. asymptotic validity: Finite-sample procedures provide some provable guarantee within a finite sample size, such as achieved probability of correct selection (PCS). Asymptotic validity procedures achieve guarantees only in some meaningful limit.

Frequentist vs. Bayesian guarantee: Frequentist probabilistic guarantees are averaged over (conceptually) repeated applications of the procedure. Bayesian probabilistic guarantees are conditioned on the data and averaged over the sources of parameter uncertainty.

In the next two subsections, we discuss some of the standard compromises and approaches for creating R&S procedures. Later, we argue that the relevant compromises may be affected by the decision to implement the procedure in a parallel computing environment. In our discussion, we group procedures by whether they are fixed-precision or fixed-budget procedures, which often, but not always, determines the computational formulation of the R&S procedure, as we discuss in §4.

3.1 Mathematical Formulation of Fixed-Precision Guarantees

Ideally, fixed-precision R&S procedures are guaranteed to deliver the optimal solution with a pre-specified frequentist probability, which we denote by $1-\alpha$ for $1-\alpha \in (1/k, 1)$. This guarantee is called the probability of correct selection (PCS) guarantee, and is expressed as

$$\mathbb{P}\{\mu_{\hat{K}} = \mu_k\} \geq 1-\alpha.$$

If there are multiple optima, or several solutions with close performance, delivering this guarantee can be computationally infeasible. As a result, making one of the following additional compromises is typical.

- One can assume that the best is unique, and accept the possibility of substantial computation before termination, as in Fan et al. [16].
- One can allow for a practically significant difference $\delta > 0$, also called an *indifference-zone (IZ)* parameter, and instead require

  $$\mathbb{P}\{\hat{K} = k \mid \mu_k - \mu_{k-1} \geq \delta\} \geq 1-\alpha.$$

  The IZ compromise has been widely adopted.
- One can be satisfied with returning a good solution with optimality gap no larger than a user specified δ:

  $$\mathbb{P}\{\mu_k - \mu_{\hat{K}} \leq \delta\} \geq 1-\alpha.$$

  Some of the procedures that deliver a guaranteed PCS also deliver a guaranteed probability of good selection (PGS), but this is not always the case.
- One can be satisfied with

  $$\mathbb{P}\{\hat{K} \in [k, k-1, k-2, \ldots, k-m+1]\} \geq 1-\alpha.$$

  That is, one can be satisfied with selecting a top-m solution based on rank order, irrespective of the actual optimality gap. This is the compromise behind ordinal optimization (see, e.g., Chen and Lee [8]).
- One can be satisfied with a subset $\hat{\mathcal{S}} \subseteq \mathcal{S}$ such that

  $$\mathbb{P}\{k \in \hat{\mathcal{S}}\} \geq 1-\alpha.$$

  Subset procedures are closely related to multiple comparison procedures that provide simultaneous confidence intervals on some set of differences, and in particular to multiple comparisons with the best (MCB, Hsu [30]). Subset guarantees can often be delivered with weak assumptions, but the conclusion may also be weak if the subset is large. Subset procedures may be used within other R&S procedures for *screening*, or removing systems from consideration that are estimated as inferior.

While all R&S procedures strive to be efficient, fixed-precision procedures require statistical guarantees to hold. Thus we formulate the objective of fixed-precision procedures by placing a hard constraint on the guarantee, but we wish to purchase processors p and create a job schedule $\mathcal{J}$ such that we minimize the expected (scaled) completion time of the procedure plus the (scaled) monetary cost of the R&S procedure. Under the stylized model in §2, we formulate this problem as

$$\text{minimize}_{p,\mathcal{J}} \quad \mathbb{E}[\beta_t T_e(\mathcal{J}) + \beta_c c(p, T_e(\mathcal{J}))] \quad \text{s.t.} \quad \mathbb{P}\{G\} \geq 1-\alpha,$$

where $\beta_t \geq 0$ and $\beta_c \geq 0$ are scaling coefficients, and the event G denotes a "good event" upon termination of the procedure, in whatever form. For example, $G = (\hat{K} = k \mid \mu_k - \mu_{k-1} \geq \delta)$ for an IZ compromise, and $G = (k \in \hat{\mathcal{S}})$ for a subset selection compromise. Usually, $\beta_t \in \{0,1\}$ and $\beta_c = 1-\beta_t$, so that only expected wall-clock time or only expected cost is minimized, depending on the cost structure of the parallel computing environment. To ensure the probabilistic guarantee constraint is satisfied, we require purchasing as many processor-hours as the procedure requires to terminate at time $T_e(\mathcal{J})$; thus the monetary budget for purchasing processor hours should be $b = \infty$.

3.2 Mathematical Formulation of Fixed-Budget Guarantees

In contrast with fixed-precision procedures, in which the simulation budget is determined in part by the required precision, the goal of most fixed-budget procedures is to identify the best system efficiently under a fixed simulation budget. Thus most fixed-budget procedures provide an "efficiency guarantee," which we formulate as

$$\text{minimize}_{p,\mathcal{J}} \quad \mathbb{E}[\mathcal{L}(G^c, \mathcal{J})] \quad \text{s.t.} \quad t(p,b) \leq t^*,$$

where the function $\mathcal{L}$ is some type of loss function that depends on an undesirable event (which, loosely speaking, we denote as G^c) upon termination of the procedure, and t^* is a fixed limit on processor hours we purchase. This formulation implies that we wish to choose the processors and the job configuration to minimize the expected loss associated with an incorrect decision, subject to a hard budgetary constraint on the amount of processor hours. The budgetary constraint on the amount of processor hours differs from the traditional constraint on the total number of simulation replications; this formulation provides a more accurate way to measure cost in a parallel computing setting. Note that equivalently, we could formulate the constraint in terms of monetary cost instead of time.

Several prominent fixed-budget guarantee methods include those provided by OCBA (Optimal Computing Budget Allocation) [9], the Bayesian Expected Value of Information (EVI) [11] and Knowledge Gradient (KG) [19] methods, and the frequentist SCORE (Sampling Criteria for Optimization using Rate Estimators) framework [53], which generalizes the work of Glynn and Juneja [23] and has a close relationship with OCBA and EVI [58].

3.3 Guarantees Require Standard Assumptions

Whether assuring a desired PCS or minimizing an expected loss, there is an underlying output distribution with respect to which the PCS or expected loss is evaluated. This underlying distribution may be derived from a strong assumption about the simulation output data, or hold asymptotically under weaker conditions. Establishing that these probability guarantees hold in small samples usually requires strong distribution assumptions. Asymptotic analysis (e.g., as $\delta \to 0$) can establish attainment in a large-sample sense. In either case, the actual distribution depends on both (a) the simulation model itself and (b) the sequence of jobs executed. Dependence on (b) is typically not a concern when there is only a single processor, but as discussed in §5, it is critical when jobs are executed in parallel.

To ensure the guarantees from the previous two sections hold, we define the standard output assumptions as follows. Recall that Y_{ir} is a random variable representing the performance of the ith system on the rth simulation replication, for each $r = 1, 2, \ldots$ and all $i \in \mathcal{S}$.

Definition 1. The *standard output assumptions* comprise the following:

1. *(Within)* for all systems $i \in \mathcal{S}$, the random variables $Y_{ir},\ r = 1, 2, \ldots$ are i.i.d. normally distributed with finite variance, and
2. *(Between)* for all pairs of systems $i, i' \in \mathcal{S}$, the random variables Y_{ir} and $Y_{i'r'}$ are independent for all $r = 1, 2, \ldots$ and all $r' = 1, 2, \ldots$.

The validity of a serial R&S procedure can usually be established under the standard output assumptions. However, these assumptions may be overly stringent. We now provide several common relaxations to the standard output assumptions.

Within Relaxations: for all systems $i \in \mathcal{S}$, the random variables $Y_{ir},\ r = 1, 2, \ldots$, (a) are i.i.d. with finite variance; (b) are stationary with finite variance; or (c) appropriately standardized, satisfy a Functional Central Limit Theorem.

Between Relaxations: for all pairs of systems $i, i' \in \mathcal{S}$, the random variables Y_{ir} and $Y_{i'r}$ are positively correlated for all $r = 1, 2, \ldots$, where the positive correlation is induced by the use of common random numbers (CRN).

CRN is a rule for assigning a set of jobs $\{J_j,\ j \in \mathcal{B}^{(b)}\}$ a "common" block of random numbers $\mathcal{U}_j = \mathcal{U}^{(b)}$ for all $j \in \mathcal{B}^{(b)}$, so that the blocks $b = 1, 2, \ldots$ exhaust all jobs that require simulation replications in $\mathcal{J}$. The use of CRN across systems to induce a positive correlation and thereby reduce the variance of the difference $\bar{Y}_i(n) - \bar{Y}_{i'}(n)$ has long been a staple of R&S methods to improve statistical efficiency; see for instance Nelson [43]. Because CRN can in fact increase variance if simulation outputs are not appropriately paired with equal numbers of observations across systems, the use of CRN imposes an additional coordination problem when there are multiple processors. As a result, CRN has not yet been central to parallel R&S procedures. Therefore, we assume independent blocks of random numbers for each job from here on unless specifically indicated. For simplicity in our stylized

model, whenever the blocks of random numbers are independent, we drop the the specification of $\mathcal{U}_j$ and instead write

$$J_j := \{(\mathcal{Q}_j, \Delta_j), (\mathcal{P}_j, \mathcal{C}_j)\}.$$

4 Computational Formulations of Existing Serial R&S Procedures

Once we have a mathematical formulation of the goals of the R&S procedure, we require a computational formulation of the procedure that can be implemented on one or more processors. To naïvely implement existing serial R&S procedures in a parallel computing setting, we require an assignment of jobs to processors such that the standard assumptions from the original serial procedure, in whatever form they exist, are still satisfied. The simplest way to accomplish this goal is to parallelize only the parts of the procedure that can be completed in an embarrassingly parallel fashion, and complete all other tasks in the original sequence. As is common in the parallel computing literature, we use the term *embarrassingly parallel* to refer to jobs that are trivially implemented in parallel and require no coordination or synchronization.

Before we provide naïve parallel computational formulations of existing serial fixed-precision and fixed-budget R&S procedures, we define the concepts of *coupled operations* and *stages*, which are concepts that assist with ordering jobs. First, recall that $\mathcal{C}_j$ is a list of calculations that are performed as part of a job j. Typical calculations that arise in R&S procedures include

- determining the sample mean and sample variance of the ith system, $\bar{Y}_i(n)$ and $S_i^2(n) := (n-1)^{-1}\sum_{r=1}^{n}(Y_{ir} - \bar{Y}_i(n))^2$, respectively,
- performing a pairwise comparison $\bar{Y}_i - \bar{Y}_{i'}$ for two systems i and i',
- determining the paired sample variance

$$S_{i,i'}^2(n) := (n-1)^{-1}\sum_{r=1}^{n}(Y_{ir} - Y_{i'r} - (\bar{Y}_i(n) - \bar{Y}_{i'}(n)))^2,$$

- updating a sample allocation rule $\mathfrak{R}$ in a fixed-budget procedure like OCBA, and
- updating a posterior distribution in a Bayesian R&S procedure.

Since operations like pairwise comparisons and calculating paired sample variances require the simulation output of two or more systems, we refer to these operations as *coupled*.

Definition 2. We define the following:

- A *coupled operation* or *coupling* is an operation or calculation in which the simulation output of two or more systems is required.
- A *fully coupled operation* or *full coupling* is an operation or calculation that requires the simulation output of all systems still in contention at that point in the procedure.

Thus coupled operations occur when the estimated system performances must be compared to each other, as in a pairwise comparison, or when some key quantity must be calculated that requires the compilation of simulation output from multiple systems. For example, calculating the estimated best system $\hat{K} = \text{argmax}_{i \in \mathcal{S}} \{\bar{Y}_i(n_i)\}$ is a fully coupled operation. Coupling is distinct from the concept of *synchronization* in parallel algorithms, since coupling is across systems, and synchronization is usually across processors. However, when there is a cost to switch from simulating one system to another, it may make sense to assign processors to simulating particular systems, in which case a coupled operation may require synchronization of simulation output across processors.

Since R&S procedures consist of simulations and comparisons, they are usually implemented in what are called *stages*. While the definition of the term "stage" has not always been consistent in the R&S literature, in the context of our stylized model, we define a stage as follows:

Definition 3. A *stage* is a portion of an R&S procedure that begins with the first simulation output obtained after initialization or after the last fully coupled operation, and ends when the next fully coupled operation terminates.

While the individual calculations required in the full coupling may be split into jobs that are carried out by multiple processors, when the final calculation of the full coupling is complete, then the stage is over. When variances are unknown, the minimum number of stages is two [14]; the variances are estimated in the first stage. Thus, the first stage almost always consists of obtaining $n_0 \geq 2$ observations from each system, and ends with a fully coupled calculation of key information for implementing the next stage.

In the computational formulations that follow, our goal is to demonstrate the coupling structure of naïve parallelization of each type of serial procedure. Thus we provide only a straightforward formulation of jobs J_j. We acknowledge that many such formulations exist; some are more efficient than others.

4.1 Computational Formulation of Fixed-Precision Procedures

To create a computational formulation for fixed-precision procedures, we begin by formulating existing serial procedures using the stylized model described in §2. We provide a basic formulation of two prominent versions of fixed-precision procedures that have different coupling structures: *two-stage* procedures, which have exactly two stages with two full couplings, and *fully-sequential* procedures, which have many stages and frequent full couplings.

The first stage of a two-stage procedure usually begins with obtaining $n_0 \geq 2$ simulation replications from each system, and ends with fully-coupled operations that use rules to screen systems and to calculate second-stage sample sizes such that desired statistical guarantees hold. The screening and sampling rules, which we denote as rules $\mathfrak{R}$, are often functions of the user-specified parameters α and δ, and

the variances of the system performances. The simulation replications in each of the two stages can be farmed out to worker processors in an embarrassingly parallel fashion, while the master completes all coupled operations. The full coupling at the end of each stage requires a full synchronization across all processors. A naïve two-stage fixed-precision procedure with an optional subset selection step is provided in Algorithm 1. Prominent two-stage procedures include Rinott [57] and NSGS [44]; Ni et al. [50] provide a parallel version of NSGS that is slightly different from Algorithm 1, called NSGS$_p$.

Algorithm 1 Naïve Two-Stage Fixed-Precision Procedure (Less Coupling)

procedure TWOSTAGE($\alpha, \delta, n_0, \mathfrak{R}_1, \mathfrak{R}_2$) ▷ Inputs: problem parameters $\alpha \in (1/k, 1)$ and $\delta \in (0, \infty)$, first-stage sample size $n_0 \geq 2$, (optional) rule $\mathfrak{R}_1$ for subset selection, and rule $\mathfrak{R}_2$ for second-stage sample size determination.
  Initialize: Set $\mathcal{Q} = \mathcal{S}$. Master creates jobs $J_i = \{(i, n_0), (\emptyset, \{\bar{Y}_i(n_0), S_i^2(n_0)\})\}$ for all $i \in \mathcal{Q}$.
  Stage 1: *(Simulate: Embarrassingly Parallel)* Workers complete jobs $\{J_i : i \in \mathcal{Q}\}$.
  (Subset Selection: Fully Coupled) Master eliminates inferior systems from $\mathcal{Q}$ using $\mathfrak{R}_1$. If $|\mathcal{Q}| = 1$, return the system in $\mathcal{Q}$ as the estimated best, $\hat{K}$. Otherwise, continue.
  (Calculation: Fully Coupled) Master determines second-stage sample sizes $N_{i,2}$ using $\mathfrak{R}_2$.
  For each $i \in \mathcal{Q}$, Master creates $J_{i,2} = \{(i, \max\{0, N_{i,2} - n_0\}), (\emptyset, \bar{Y}_i(N_{i,2}))\}$.
  Stage 2: *(Simulate: Embarrassingly Parallel)* Workers complete jobs $\{J_i : i \in \mathcal{Q}\}$.
  (Compare: Fully Coupled) Master returns $\hat{K} = \text{argmax}_{i \in \mathcal{Q}} \{\bar{Y}_i(N_{i,2})\}$.
end procedure ▷ Algorithm inspired by [52, p. 127].

While two-stage procedures can be completed using mostly embarrassingly parallel computation with little synchronization, they are less efficient than fully-sequential procedures in terms of the expected total number of simulation replications required. Fully-sequential procedures gain sampling efficiency by frequent comparisons and screening. Arguably, fully-sequential procedures have the maximum number of stages and hence a "maximal" coupling structure. In each Stage 2+, one simulation replication is obtained from each system still in contention, sample means and paired variances are updated, and inferior systems are screened out. Since the number of simulation replications per system are equal across surviving systems in each stage of the procedure, that is, $n_i = n_{i'}$ for all $i, i' \in \mathcal{Q}$, some fully sequential procedures such as $\mathcal{KN}$ can be implemented with CRN, further enhancing efficiency. However, screening is an inherently coupled operation — especially when screening requires all pairwise comparisons between systems. Thus when adapting R&S procedures to a parallel computing platform, there exists a tension between sampling efficiency gained by frequent screening, and the potential inefficiency of attempting to perform frequent coupled screening operations across many processors. A generic, naïvely parallelized fully-sequential procedure is provided in Algorithm 2. Prominent fully-sequential procedures include $\mathcal{KN}$ [35].

Algorithm 2 Naïve Fully Sequential Fixed-Precision Procedure (More Coupling)

procedure FULLYSEQUENTIAL($\alpha, \delta, n_0, \mathfrak{R}_s$) ▷ Inputs: problem parameters $\alpha \in (1/k, 1)$ and $\delta \in (0, \infty)$, first-stage sample size $n_0 \geq 2$, a screening rule $\mathfrak{R}_s$.
  Initialize: Set the total samples per system $n = n_0$ and the systems in contention $\mathcal{Q} = \mathcal{S}$. Master creates jobs $J_i = \{(i, n_0), (\emptyset, \{\bar{Y}_i(n_0)\})\}$ for all $i \in \mathcal{Q}$.
  Stage 1+: *(Simulate: Embarrassingly Parallel)* Workers complete jobs $\{J_i : i \in \mathcal{Q}\}$.
  (Calculation: Coupled) Master computes $S^2_{i,i'}(n)$ for all $i, i' \in \mathcal{Q}$, $i \neq i'$.
  (Screen: Fully Coupled) Master uses rule $\mathfrak{R}_s$ on $(\bar{Y}_i(n), S^2_{i,i'}(n))$, $i, i' \in \mathcal{Q}$ to eliminate inferior systems from $\mathcal{Q}$. If $|\mathcal{Q}| = 1$, return the system in $\mathcal{Q}$ as the estimated best, $\hat{K}$. Otherwise, set $n = n + 1$. Master creates new jobs $J_{i,n} = \{(i, 1), (\emptyset, \bar{Y}_i(n))\}$ for each $i \in \mathcal{Q}$.
  Go to the next stage, Step 3.
end procedure ▷ Algorithm inspired by [52, p. 129].

4.2 Computational Formulation of Fixed-Budget Procedures

Fixed-budget procedures often take a similar computational structure, outlined in Algorithm 3. As in the fixed-precision procedures, the first stage begins by obtaining $n_0 \geq 2$ simulation replications from each system, and ends with a (usually) fully-coupled operation that determines how to allocate the Δ simulation replications in the next stage, using a sampling rule $\mathfrak{R}$. This process of obtaining Δ simulation replications per stage and updating the sampling rule is repeated until the total simulation time has been exhausted. Since the frequency of the coupling and the number of stages is determined by the parameter Δ, these procedures tend to have a flexible coupling frequency. We note that some procedures are designed for myopic sampling, such that $\Delta = 1$, while other procedures are more flexible in the choice of Δ.

Algorithm 3 Fixed-Budget Procedure (Flexible Coupling Frequency)

procedure EFFICIENCY($n_0, \Delta, t^*, \mathfrak{R}$) ▷ Inputs: initial simulation budget $n_0 \geq 2$, stagewise simulation budget Δ, limit on total effort $t^* > 0$, and rule $\mathfrak{R}$ for stagewise allocation.
  Initialize: Set stage $\ell = 1$, sample sizes $n_{i,\ell} = n_0$ for all $i \in \mathcal{S}$, total effort $t_0 = 0$, and Master creates jobs $J_i = \{(i, n_0), (\emptyset, \{\bar{Y}_i(n_0), S^2_i(n_0)\})\}$ for all $i \in \mathcal{S}$.
  while $t_{\ell-1} \leq t^*$ **do**
    Stage ℓ: *(Simulate: Embarrassingly Parallel)* Workers complete jobs $\{J_i\}$.
    (Calculation: Fully Coupled) Master applies rule $\mathfrak{R}$ to statistical history to find next-stage sample allocation $\{n_{i,\ell+1} : \sum_{i \in \mathcal{S}} n_{i,\ell+1} = \Delta\}$. Master creates jobs $J_i = \{(i, n_{i,\ell+1}), (\bar{Y}_i(n_{i,\ell+1}), S^2_i(n_{i,\ell+1}))\}$ for all $i \in \{i' : i' \in \mathcal{S}, n_{i',\ell+1} \geq 1\}$.
    Master updates total effort expended so far t_ℓ.
    Master sets $\ell = \ell + 1$.
  end while
  return $\hat{K} = \text{argmax}_{i \in \mathcal{S}} \{\bar{Y}_i(\sum n_{i,\ell})\}$.
end procedure ▷ Algorithm inspired by [52, p. 132].

5 Parallelization: Efficiency and Validity

Having presented fairly straightforward parallel computational frameworks for existing serial R&S procedures that should preserve the standard assumptions from the original serial procedure, one may wonder, why not simply use these procedures? While such procedures surely can be implemented, they are unlikely to scale well to larger problem instances and to achieve the levels of speedup and efficiency we would like to see from a parallel R&S algorithm. The concepts of speedup and efficiency (or scalability) are defined in [2] as follows. Suppose we are handed a parallel algorithm that requires t_p wall-clock time to be run on p identical processors in parallel, and t_s wall-clock time to be run on only one of the processors. Then the speedup is defined as the ratio of the sequential time to the parallel time, $\textit{speedup} := t_s/t_p$. Given $p \geq 1$ processors, the *efficiency* is defined as the scaled speedup, $\textit{efficiency} := s/p = t_s/(pt_p)$. Thus speedup gives a measure of how beneficial it is to execute the algorithm in parallel, while efficiency measures the utilization of the available processors. Efficiency of 1 corresponds to *linear speedup*, in which case the speedup is equal to the number of processors, p. Embarrassingly parallel jobs that are appropriately load-balanced across cores tend to achieve almost linear speedup.

Since embarrassingly parallel implementations achieve almost linear speedup, it seems that two-stage procedures would perform the best in parallel. However, recall that two-stage procedures require more simulation replications, on average, than those that have more frequent coupled operations like screening. Thus it may benefit the procedure to introduce more frequent coupled operations. However, we have two potential forms of idleness that arise: (a) the master may be idle waiting for every simulation replication to complete before it completes any coupled operations — especially if simulation replication completion times are random — and (b) if screening and fully-coupled operations take significant computational effort on the master, many worker processors must wait for the master to create new jobs.

Then, we may wish to design a procedure that does not require the processors to wait for each other. Unfortunately, *the standard assumptions are most easily assured by one-job-at-a-time execution.* Estimators can become biased if we do not wait for all parallelized simulation replications to complete; such bias was investigated by Heidelberger [27] and Glynn and Heidelberger [21, 22]. Further, serious violations of the standard assumptions can occur if jobs are executed in parallel but output data are used as-available and without enforcing conformance with single-processor execution. These include the following, as described in Luo et al. [40] and Ni et al. [49, 51].

Random sample size: The number of observations from system i when the nth overall observation is obtained may be random if job execution time is variable.

Not i.i.d.: The observations n_i from system i may not be i.i.d. if the order in which jobs complete is not the order in which the jobs were dispatched, and there is a dependence between returned value and execution time.

Dependence across systems: A difficult-to-characterize dependence across systems' outputs can be induced if elimination of system i by system i' frees processors that affect the number of observations obtained from other systems.

These issues suggest that we must employ output coordination strategies that ensure all calculations $(\mathcal{P}_j, \mathcal{C}_j)$ across jobs j are executed as they would be if there were only a single processor. However, this still leads to a potential degradation of efficiency and speedup from the two forms of idleness: master waiting for simulation replications, and workers waiting for the master's calculations.

Based on this analysis, efficient parallel R&S requires procedures with one or more of the following characteristics: (a) they implement careful load balancing to retain the standard assumptions without significant idling and overwhelming communication; or (b) they are valid under weaker assumptions than the standard ones; or (c) the procedure uses a combination of the strategies above. We discuss existing parallel R&S procedures of both types in the next section.

6 Existing Parallel Ranking and Selection Procedures

Existing parallel R&S procedures overcome some of the shortcomings of the naïve parallelization of existing serial R&S procedures. We now discuss the state-of-the-art in both fixed-precision and fixed-budget parallel ranking and selection procedures.

6.1 Parallel Fixed-Precision Procedures

We describe four parallel fixed-precision procedures and formulate each procedure in terms of the types of jobs created and deployed to the workers. First, Luo et al. [39, 40] extend the $\mathcal{KN}$ procedure [34, 35], which provides a PCS guarantee, to the parallel setting in two distinct ways: a conservative vector-filling procedure (VFP) that strictly enforces the standard assumptions, and an aggressive asymptotic parallel selection (APS) procedure that is valid under weaker assumptions. Both algorithms resemble Algorithm 2 in that they use elimination at every stage; their key difference is in how they define the completion of a stage. Then, we discuss a simple divide-and-conquer approach by Chen [10] for when the number of processors is small. Finally, a substantial extension of the divide-and-conquer approach is provided by the good selection procedure (GSP) of Ni et al. [49], which provides a PGS guarantee.

Of these procedures, we highlight two procedures for their strategies related enhancing efficiency and maintaining validity. First, APS never allows the master to idle waiting for simulation replications, but maintains its validity under conditions weaker than the standard ones. Second, GSP maintains validity under the standard assumptions, but performs careful load balancing to maintain efficiency.

6.1.1 VFP: Vector-Filling Procedure

In VFP, the master creates/executes three types of jobs:

1. Initialization jobs:

$$\mathcal{J}_0 = [\{(1,n_0),(\emptyset)\},\{(2,n_0),(\emptyset)\}\ldots,\{(k,n_0),(\emptyset)\},\{(\emptyset),(\mathcal{P}_0,\mathcal{C}_0)\}]$$

   where the set $\mathcal{P}_0$ includes the k preceding simulation jobs, and the calculations $\mathcal{C}_0$ include computing the variance of all pairwise differences, making pairwise comparisons of the sample means of all k systems, and possibly eliminating some systems.
2. Round-robin simulation jobs: Conceptually, there is an infinite set of sets of simulation jobs

$$\mathcal{J}_\ell = [\{(1,1),(\emptyset)\},\{(2,1),(\emptyset)\}\ldots,\{(k,1),(\emptyset)\}],\ell = 1,2,\ldots$$

   that obtain one additional replication from each system. However, if at stage ℓ' system i' is eliminated, then all $\{(i',1),(\emptyset)\}$ jobs are elminated from the unexecuted simulation job set. Upon completion of a simulation job, a worker pulls the next simulation job in the sequence to execute.
3. Elimination jobs: Stage ℓ is defined by an elimination job $\{(\emptyset),(\mathcal{P}_\ell,\mathcal{C}_\ell)\}$, where $\mathcal{P}_\ell$ contains all simulation jobs $\mathcal{J}_\ell$, and $\mathcal{C}_\ell$ performs pairwise comparisons of all systems that have not been eliminated at an earlier stage. The elimination jobs are executed by the master.

The VFP terminates when there is only one system that has not been eliminated. The term "vector filling" is appropriate because the VFP enforces the standard assumptions by associating each simulation output with its job set $\mathcal{J}_\ell$, and only performing a full coupling for stage ℓ when all jobs in $\mathcal{J}_\ell$ have completed. For this reason, outputs from later job sets, say $\mathcal{J}_{\ell+1}$, that complete before jobs in $\mathcal{J}_\ell$ must be held in a vector for later elimination calculations.

6.1.2 APS: Asymptotic Parallel Selection

The APS procedure is superficially similar to the VFP, but a small change makes its computational profile quite different.

1. Initialization jobs:

$$\mathcal{J}_0 = [\{(1,n_0),(\emptyset)\},\{(2,n_0),(\emptyset)\}\ldots,\{(k,n_0),(\emptyset)\},\{(\emptyset),(\mathcal{P}_0,\mathcal{C}_0)\}]$$

   where the set $\mathcal{P}_0$ includes the k preceding simulation jobs, and the calculations $\mathcal{C}_0$ include computing the marginal variance of all k systems, making pairwise comparisons of the sample means of all k systems, and possibly eliminating some systems. (These initialization jobs are the same as those in VFP.)

2. Round-robin simulation jobs: Conceptually, there is an infinite set of sets of simulation jobs

$$\mathcal{J}_\ell = [\{(1,1),(\emptyset)\},\{(2,1),(\emptyset)\}\ldots,\{(k,1),(\emptyset)\},\{(\text{phantom},0),(\emptyset)\}],\ell = 1,2,\ldots$$

that obtain one additional replication from each real system, and no replications from a "phantom" system. Again, if at stage ℓ' real system i' is eliminated, then all $\{(i',1),(\emptyset)\}$ jobs are elminated from the unexecuted simulation job set. Upon completion of a simulation job, a worker pulls the next simulation job in the sequence to execute, which could be a phantom.
3. Elimination jobs: Stage ℓ is defined by an elimination job $\{(\emptyset),(\mathcal{P}_\ell,\mathcal{C}_\ell)\}$, where $\mathcal{P}_\ell$ is the ℓth phantom job. The calculation $\mathcal{C}_\ell$ updates marginal variances and performs pairwise comparisons of all systems that have not been eliminated at an earlier stage *using all available simulation output data.* The elimination jobs are executed by the master.

The APS procedure defines a stage as the completion of a phantom job, but otherwise makes no attempt to process simulation jobs in any order. Thus, it is aggressive in that the master never idles waiting for a particular real simulation job to complete, but it is subject to all of the violations of standard assumptions described in Section 5. The validity of APS is asymptotic, as $\delta \to 0$, with the key insight being that since there are $p < \infty$ processors the simulation jobs may only be out of order by an asymptotically negligible p jobs.

6.1.3 Simple Divide-and-Conquer

An early paper by Chen [10] descibes a simple approach that is sensible when the number of processors p is small; GSP below can be considered a substantial extension of this idea. There are two types of jobs:

1. Group R&S jobs: The k systems are divided as evenly as possible into p nonoverlapping groups of systems, say $\mathcal{G}_1,\mathcal{G}_2,\ldots\mathcal{G}_p$, and p jobs are formed

$$J_j = \{(\mathcal{G}_j,\Delta_j),(\mathcal{G}_j,\mathcal{C}_j)\},\ j = 1,2,\ldots,p$$

where each job j is a complete R&S procedure that returns a group-best selected system $\widehat{i}_j$ along with its accumulated output data.
2. Final R&S job: Let $\mathcal{Q} = \{\widehat{i}_1,\widehat{i}_2,\ldots,\widehat{i}_p\}$, the group bests. Then the final job is

$$J_{p+1} = \{(\mathcal{Q},\Delta_{p+1}),(\mathcal{Q},\mathcal{C}_{p+1})\},\ j = 1,2,\ldots,p$$

which performs a R&S procedure on the group-best systems $\mathcal{Q}$ starting with their previously accumulated data and $\mathcal{C}_{p+1}$ computes the sample means and selects the best.

Chen [10] suggests some specific R&S procedures for each type of job, but the framework is flexible. The simplicity of this strategy is appealing, but it will lose effectiveness when $k \gg p$ so that the group R&S jobs themselves are challenging.

6.1.4 GSP: Good Selection Procedure

The GSP procedure of [49] (also see [48,50,51]) provides a PGS guarantee, instead of the usual PCS guarantee, under the standard output assumptions. GSP exhibits good speedup and efficiency using careful load-balancing and reduced computation. Several key strategies of GSP include: (a) distributing screening tasks to the workers in a divide-and-conquer fashion to avoid overwhelming the master with screening calculations; (b) using only a reduced number of pairwise comparisons instead of completing all pairwise comparisons; (c) carefully constructing load-balanced jobs of large-enough size to prevent overwhelming the master with communication. As a result, when implemented in a high-performance computing (HPC) environment in C with MPI, the master is idle most of the time. However in its idleness, the master usually is ready to communicate and can ensure the workers are *not* idle most of the time.

GSP has three stages and one "phase," which is a sequential portion of the algorithm containing multiple stages. GSP's stages and phases are: an optional load-balancing stage; an initialization stage with screening; a sequential phase that contains multiple stages and is somewhat similar in structure to Algorithm 2 after initialization; and a Rinott stage, similar in structure to Algorithm 1. The sequential phase is intended to harness the efficiency of sequential screening to create a subset of contender systems likely to contain the best. In the Rinott stage, the appropriate sample sizes for the remaining systems are calculated, and the simulations for remaining competitive systems are completed in an embarrassingly parallel fashion.

In this subsection, we assume we have p worker processors, where the zeroth processor is the master. In each stage or phase, the master creates the following types of jobs:

1. Optional load-balancing jobs: The master randomly permutes the systems in $\mathcal{S}$ and assigns an approximately equal number of systems to groups $\mathcal{G}_0^1, \ldots, \mathcal{G}_0^p$, for each processor. Then the master creates jobs

$$\mathcal{J}_0 = \{(\mathcal{G}_0^w, n_0^*), (\emptyset, \{\bar{T}_i : i \in \mathcal{G}_0^w\})\}_{w=1}^p,$$

   where $\bar{T}_i$ is the average simulation completion time across all replications n_0^* from simulating system i. After calculating statistics $\bar{T}_i$, the simulation output is thrown away, due to potential dependence between the output random variable Y_{ir} and the simulation replication completion time T_{ir}.
2. Initialization jobs: Using information from the optional load-balancing step if available, the master partitions the systems in $\mathcal{S}$ into load-balanced simulation groups $\mathcal{G}_1^1, \ldots, \mathcal{G}_1^p$ for each processor. Then the master creates jobs

$$\mathcal{J}_1 = \{(\mathcal{G}_1^w, n_0), (\emptyset, \{(\bar{Y}_i(n_0), S_i^2(n_0), \mathcal{C}_1) : i \in \mathcal{G}_1^w\})\}_{w=1}^p,$$

where $\mathcal{C}_1$ is a screening calculation that only reports the surviving systems and their sufficient statistics to the master. The master updates the surviving systems $\mathcal{Q}$.

3. Sequential phase jobs: The master divides the systems into approximately load-balanced screening groups $\mathcal{G}_2^1, \ldots, \mathcal{G}_2^p$ using rule $\mathfrak{R}_1^{\text{GSP}}$, so that each processor always screens the same set of systems. The master also uses rule $\mathfrak{R}_2^{\text{GSP}}$ to determine an appropriate "batch size" b_i of simulation replications to obtain from each system $i \in \mathcal{Q}$ in each simulation job, so that the master is not overwhelmed with communication. The sequential phase ends when a pre-determined maximum number of batches has been simulated, or when $|Q| = 1$.

   a. Simulation jobs: The master creates and maintains an ordered list of batched simulation jobs for each system $i \in \mathcal{Q}$. Whenever a worker becomes idle and the master indicates that some systems in its screening group are not ready for screening, the worker requests the next simulation job in the list, for any system $i \in \mathcal{Q}$. For each system still in contention $i \in \mathcal{Q}$, the νth simulation job is
   $$J_{i,\nu} = \{(i, b_i), (\emptyset, \bar{Y}(b_i))\}, \nu = 1, 2, \ldots.$$

   b. Within-group and best-across-processor screening jobs: Whenever a processor becomes idle and the νth simulation batch has completed for all systems in its screening group, the processor pulls the "screening job" for its group from the master,
   $$J_\nu^w = \{\emptyset, (\{J_{i,\nu} : i \in \mathcal{G}_2^w\}, \mathcal{C}_\nu^w)\},$$
   where $\mathcal{C}_\nu^w$ is an all-pairwise screening job within group $\mathcal{G}_2^w$, as well as among the best systems who have completed batch ν from the other screening groups. The processor then reports the indices of eliminated systems to the master, who updates the set of systems still in contention, $\mathcal{Q}$. Note that the νth screening must occur before the $(\nu+1)$th screening, and so on. Per Definition 3, each within-group screening that uses the best systems from screening groups on *all* the other processors constitutes the end of a stage.

4. Rinott stage jobs: If $|\mathcal{Q}| > 1$, the master uses a rule $\mathfrak{R}$ to determine the Rinott-stage sample sizes $N_{i,4}$ for all remaining systems $i \in \mathcal{Q}$. Let N_i be the total number of simulation replications observed from each system $i \in \mathcal{Q}$ so far before the Rinott stage, and define $N_{i,4}^+ := \max\{0, N_{i,4} - N_i\}$ as the number of additional simulation replications required from system i. The master then arranges the required additional simulation replications for each system into load-balanced "batched" jobs; for ease of exposition, we omit the batching notation in this stage. Then for all $i \in \mathcal{Q}$ such that $N_{i,4}^+ > 0$, the master creates the jobs
   $$J_{4,i} = \{(i, N_{i,4}^+), (\emptyset, \bar{Y}_i(N_{i,4}))\}.$$

After all simulation replications terminate, the master updates the sample means with the latest data and returns the estimated-best system $\hat{K}$.

6.2 Parallel Fixed-Budget Procedures

Luo et al. [41] is the first reported effort to parallelize an R&S procedure, specifically OCBA in the fixed-budget setting. Their base algorithm resembles Algorithm 3, and they assume a master-worker environment with a small number of workers ($p \leq 3$ in their experiments).

The master creates/executes three types of jobs:

1. Initialization jobs:

$$[\{(1,n_0),(\emptyset)\},\{(2,n_0),(\emptyset)\}\ldots,\{(k,n_0),(\emptyset)\},\{(\emptyset),(\mathcal{P}_0,\mathcal{C}_0)\}]$$

   where the set $\mathcal{P}_0$ includes the k preceding simulation jobs, and the calculations $\mathcal{C}_0$ include computing the marginal sample means and variance of the k systems.
2. Simulation jobs: In the ℓth stage, p jobs $\{(i_j,\Delta),(\emptyset)\}$ for $j = 1,2\ldots,p$ are created, where $i_j \in \{1,2,\ldots,k\}$ denote p distinct systems, each allocated the same number of replications, Δ. These jobs are exectued by the p workers in parallel.
3. OCBA jobs: $\{(\emptyset),(\mathcal{P}_\ell,\mathcal{C}_\ell)\}$, where $\mathcal{P}_\ell$ contains all of the simulation jobs from the ℓth simulation stage, and $\mathcal{C}_\ell$ performs the OCBA optimization to find the p systems for whom an allocation of Δ additional replications would most rapidly increase an approximate posterior PCS expression. Simulation jobs are then created for these p systems.

By having the OCBA job hold for the return of all of the ongoing simulation jobs, this algorithm enforces the single-processor assumptions behind OCBA at each stage. As noted by the authors, there is a loss of statistical efficiency by simulating the top-p OCBA systems at each stage, rather than simulating the single best then reevaluating, but there is a gain in computational efficiency. The algorithm terminates when a fixed number-of-replications budget is expended. A related paper by Yoo et al. [64] also applies OCBA in a parallel *search* setting where not all systems are expected to be simulated.

6.3 Available Implementations of Parallel R&S Procedures

To the best of our knowledge, only one commercial simulation product, Simio (www.simio.com), has implemented R&S procedures that exploit parallel computing. Simio has implemented two fixed-precision procedures: $\mathcal{KN}$ [34,35] which uses multiple processors on a local PC, and GSP [49] which is specifically designed to use high-performance or cloud computing. $\mathcal{KN}$ gains efficiency by obtaining repli-

cations in parallel; in every other sense it is the single-processor algorithm and it implements full synchronization at every stage.

There are also public code repositories that contain parallel versions of R&S procedures. In this paragraph, the citations provide links to code repositories that are publicly available at the time of writing. GSP has been implemented in MPI [46], MapReduce [45], and Spark [47]. Code for a parallel version of OCBA is available [38]. As a repository for the simulation optimization community, `simopt.org` [28], also contains test problems for a variety of problem types, as well as an algorithms library with publicly available code.

7 A Future Research Agenda

Effective and efficient parallel R&S procedures of the future seem likely to be obtained by a careful coordination of a number of ideas. Here is a part of the roadmap as we see it.

Assignment of jobs to processors is clearly a type of stochastic parallel-machine scheduling problem as addressed by the operations research literature (see for instance Pinedo [56]). The objective in such problems is often to minimize makespan, which is analogous to our objective in the fixed-precision formulation, and sequencing constraints are similar to our dependence of certain computations on the completion of particular jobs, $(\mathcal{P}_j, \mathcal{C}_j)$. A key difference is that the jobs that need scheduling in parallel R&S may evolve based on the simulation outputs obtained from earlier jobs, rather than being all available in advance or arriving according to some exogeneous stochastic process. Nevertheless, this is a deep literature whose lessons should not be ignored.

Strategies that avoid full coupling seem critical as the number of all-pairwise comparisons grows as $O(k^2)$. Thus, as k increases it becomes computationally prudent to simulate more outputs than strictly needed for, say, correct selection to avoid coupling. This can be done from at least two directions:

1. Distributed screening: Couplings of $k' \ll k$ systems to screen out inferior systems and pass competitors to full couplings, thereby reducing the comparisons to $O((k')^2)$.
2. Distributed killers: Obtaining high-precision estimates of an apparently good solution and distributing it to all or groups of systems to screen out inferior ones; this type of screening is $O(k)$.

The fixed-budget formulation, when expressed as a limit on the number of simulation replications, has always been somewhat artificial. A fixed *monetary* or *time* budget for parallel computation, on the other hand, is both concrete and relevant. To us, the joint choice of number of processors p and jobs to execute $\mathcal{J}$ to minimize expected loss with respect to a monetary budget looks very challenging indeed. We suspect that a strategy that chooses p based on a priori problem characteristics, and then treats it as fixed when optimizing over $\mathcal{J}$, will be the most productive avenue.

Finally, parallel R&S for very large numbers of systems should cause us to revisit the standard R&S objectives as described in Sections 3.1–3.2. For very large k a PGS guarantee seems more relevant and easier to obtain than a PCS guarantee, as it seems likely there are many close competitors. More critically, *any* objective that returns a *single* system $\widehat{K}$ without additional inference about the others seems questionable. Consider an alternative objective:

Suppose, based on previous experience with similar problems, a known standard for "good" performance of $\mu^\star$ can be established. Finding, with high probability, the subset of systems with $\mu_i \geq \mu^\star$ is a fully uncoupled problem that is embarrassingly parallel. A related approach by Singham and Szechtman [60] defines inclusion of inferior systems in the subset as a "false discovery" and sets as the objective bounding the false discovery *rate*. In terms of both conservatism of the inference and growth of computation these ideas scale better than the traditional objectives.

8 WSC 2017

At the time of writing, we are aware of at least one paper on parallel R&S under review for WSC 2017. Thus parallel R&S continues to be an active research area at WSC.

Acknowledgements Hunter's research was partially supported by the National Science Foundation under Grant Number CMMI-1554144. Nelson's research was partially supported by the National Science Foundation under Grant Number CMMI-1537060 and GOALI co-sponsor SAS Institute.

References

1. Andradóttir, S., Kim, S.H.: Fully sequential procedures for comparing constrained systems via simulation. Naval Research Logistics **57**(5), 403–421 (2010). DOI 10.1002/nav.20408
2. Barlas, G., Kaufman, M.: Multicore and GPU Programming: An Integrated Approach. Elsevier (2015)
3. Bechhofer, R.E.: A single-sample multiple decision procedure for ranking means of normal populations with known variances. The Annals of Mathematical Statistics **25**(1), 16–39 (1954)
4. Bechhofer, R.E., Santner, T.J., Goldsman, D.: Design and Analysis of Experiments for Statistical Selection, Screening and Multiple Comparisons. John Wiley and Sons, New York (1995)
5. Boesel, J., Nelson, B.L., Kim, S.H.: Using ranking and selection to 'clean up' after simulation optimization. Operations Research **51**(5), 814–825 (2003)
6. Branke, J., Chick, S.E., Schmidt, C.: Selecting a selection procedure. Management Science **53**(12), 1916–1932 (2007)
7. Bubeck, S., Cesa-Bianchi, N., et al.: Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Foundations and Trends® in Machine Learning **5**(1), 1–122 (2012)
8. Chen, C.H., Lee, L.H.: Stochastic Simulation Optimization: An Optimal Computing Budget Allocation. World Scientific, Hackensack, NJ (2010)

9. Chen, C.H., Lin, J., Yücesan, E., Chick, S.E.: Simulation budget allocation for further enhancing the efficiency of ordinal optimization. Discrete Event Dynamic Systems **10**(3), 251–270 (2000). DOI 10.1023/A:1008349927281
10. Chen, E.J.: Using parallel and distributed computing to increase the capability of selection procedures. In: M.E. Kuhl, N.M. Steiger, F.B. Armstrong, J.A. Joines (eds.) Proceedings of the 2005 Winter Simulation Conference, pp. 723–731. Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2005)
11. Chick, S.E., Branke, J., Schmidt, C.: Sequential sampling to myopically maximize the expected value of information. INFORMS Journal on Computing **22**(1), 71–80 (2010)
12. Dong, J., Zhu, Y.: Three asymptotic regimes for ranking and selection with general sample distributions. In: T.M.K. Roeder, P.I. Frazier, R. Szechtman, E. Zhou, T. Huschka, S.E. Chick (eds.) Proceedings of the 2016 Winter Simulation Conference, pp. 277–288. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2016)
13. Dudewicz, E.J.: Statistics in simulation: how to design for selecting the best alternative. In: H.J. Highland, T.J. Schriber, R.G. Sargent (eds.) Proceedings of the 1976 Bicentennial Winter Simulation Conference, pp. 67–71 (1976). URL http://informs-sim.org/wsc76papers/1976_0012.pdf
14. Dudewicz, E.J., Dalal, S.R.: Allocation of observations in ranking and selection with unequal variances. Sankhya: The Indian Journal of Statistics **B37**, 28–78 (1975)
15. Fabian, V.: Note on Anderson's sequential procedures with triangular boundary. The Annals of Statistics **2**(1), 170–176 (1964)
16. Fan, W., Hong, L.J., Nelson, B.L.: Indifference-zone-free selection of the best. Operations Research **64**(6), 1499–1514 (2016)
17. Feldman, G., Hunter, S.R.: SCORE allocations for bi-objective ranking and selection. Optimization Online (2016). URL http://www.optimization-online.org/DB_HTML/2016/06/5469.html
18. Feldman, G., Hunter, S.R., Pasupathy, R.: Multi-objective simulation optimization on finite sets: optimal allocation via scalarization. In: L. Yilmaz, W.K.V. Chan, I. Moon, T.M.K. Roeder, C. Macal, M.D. Rossetti (eds.) Proceedings of the 2015 Winter Simulation Conference, pp. 3610–3621. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2015). DOI 10.1109/WSC.2015.7408520
19. Frazier, P.I., Powell, W.B., Dayanik, S.: A knowledge-gradient policy for sequential information collection. SIAM J. Control Optim. **47**(5), 2410–2439 (2008)
20. Fu, M. (ed.): Handbook of Simulation Optimization, *International Series in Operations Research & Management Science*, vol. 216. Springer-Verlag, New York (2015). DOI 10.1007/978-1-4939-1384-8
21. Glynn, P.W., Heidelberger, P.: Bias properties of budget constrained simulations. Operations Research **38**(5), 801–814 (1990)
22. Glynn, P.W., Heidelberger, P.: Analysis of parallel replicated simulations under a completion time constraint. ACM Transactions on Modeling and Computer Simulations **1**(1), 3–23 (1991)
23. Glynn, P.W., Juneja, S.: A large deviations perspective on ordinal optimization. In: R.G. Ingalls, M.D. Rossetti, J.S. Smith, B.A. Peters (eds.) Proceedings of the 2004 Winter Simulation Conference, pp. 577–585. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2004). DOI 10.1109/WSC.2004.1371364
24. Goldsman, D.: Ranking and selection in simulation. In: S. Roberts, J. Banks., B. Schmeiser (eds.) Proceedings of the 1983 Winter Simulation Conference, pp. 387–393 (1983). URL http://informs-sim.org/wsc83papers/1983_0017.pdf
25. Goldsman, D., Nelson, B.L.: Statistical screening, selection, and multiple comparison procedures in computer simulation. In: D.J. Medieros, E.F. Watson, J.S. Carson, M.S. Manivannan (eds.) Proceedings of the 1998 Winter Simulation Conference, pp. 159–166. Institute of Electrical and Electronics Engineers, Piscataway, NJ (1998)
26. Gupta, S.S., Panchapakesan, S.: Multiple decision procedures: theory and methodology of selecting and ranking populations. SIAM (2002)
27. Heidelberger, P.: Discrete event simulations and parallel processing: statistical properties. Siam J. Stat. Comput. **9**(6), 1114–1132 (1988)

28. Henderson, S.G., Pasupathy, R.: Simulation optimization library (2016). URL http://www.simopt.org
29. Hong, L.J., Nelson, B.L.: Discrete optimization via simulation using COMPASS. Operations Research **54**(1), 115–129 (2006)
30. Hsu, J.C.: Constrained simultaneous confidence intervals for multiple comparisons with the best. Annals of Statistics **12**, 1136–1144 (1984)
31. Hunter, S.R., Applegate, E.A., Arora, V., Chong, B., Cooper, K., Rincón-Guevara, O., Vivas-Valencia, C.: An introduction to multi-objective simulation optimization. Optimization Online (2017). URL http://www.optimization-online.org/DB_HTML/2017/03/5903.html
32. Hunter, S.R., McClosky, B.: Maximizing quantitative traits in the mating design problem via simulation-based Pareto estimation. IIE Transactions **48**(6), 565–578 (2016). DOI 10.1080/0740817X.2015.1096430
33. Jamieson, K., Nowak, R.: Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In: Information Sciences and Systems (CISS), 2014 48th Annual Conference on, pp. 1–6. IEEE (2014)
34. Kim, S., Nelson, B.L.: On the asymptotic validity of fully sequential selection procedures for steady-state simulation. Operations Research **54**(3), 475–488 (2006). DOI 10.1287/opre.1060.0281
35. Kim, S.H., Nelson, B.L.: A fully sequential procedure for indifference-zone selection in simulation. ACM Transactions on Modeling and Computer Simulation **11**(3), 251–273 (2001)
36. Kim, S.H., Nelson, B.L.: Selecting the best system. In: S.G. Henderson, B.L. Nelson (eds.) Simulation, Handbooks in Operations Research and Management Science, Volume 13, pp. 501–534. Elsevier, Amsterdam, The Netherlands (2006)
37. Lee, L.H., Chew, E.P., Teng, S., Goldsman, D.: Finding the non-dominated Pareto set for multi-objective simulation models. IIE Transactions **42**, 656–674 (2010). DOI 10.1080/07408171003705367
38. Li, H.: Parallel OCBA (2017). URL https://gitlab.com/o2des_dev/ParallelOCBA
39. Luo, J., Hong, L.J.: Large-scale ranking and selection using cloud computing. In: S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, M. Fu (eds.) Proceedings of the 2011 Winter Simulation Conference, pp. 4051–4061. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2011). DOI 10.1109/WSC.2011.6148094
40. Luo, J., Hong, L.J., Nelson, B.L., Wu, Y.: Fully sequential procedures for large-scale ranking-and-selection problems in parallel computing environments. Operations Research **63**(5), 1177–1194 (2015). DOI 10.1287/opre.2015.1413
41. Luo, Y.C., Chen, C.H., Yücesan, E., Lee, I.: Distributed web-based simulation optimization. In: J.A. Joines, R.R. Barton, K. Kang, P.A. Fishwick (eds.) Proceedings of the 2000 Winter Simulation Conference, pp. 1785–1793. Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2000). DOI 10.1109/WSC.2000.899170
42. Negoescu, D.M., Frazier, P.I., Powell, W.B.: The Knowledge-Gradient algorithm for sequencing experiments in drug discovery. INFORMS Journal on Computing **23**(3), 346–363 (2011)
43. Nelson, B.: Foundations and methods of stochastic simulation: a first course. Springer Science & Business Media (2013)
44. Nelson, B.L., Swann, J., Goldsman, D., Song, W.: Simple procedures for selecting the best simulated system when the number of alternatives is large. Operations Research **49**(6), 950–963 (2001)
45. Ni, E.C.: MapRedRnS: Parallel ranking and selection using MapReduce (2015). URL https://bitbucket.org/ericni/mapredrns
46. Ni, E.C.: mpiRnS: Parallel ranking and selection using MPI (2015). URL https://bitbucket.org/ericni/mpirns
47. Ni, E.C.: SparkRnS: Parallel ranking and selection using Spark (2015). URL https://bitbucket.org/ericni/sparkrns

48. Ni, E.C., Ciocan, D.F., Henderson, S.G., Hunter, S.R.: Comparing Message Passing Interface and MapReduce for large-scale parallel ranking and selection. In: L. Yilmaz, W.K.V. Chan, I. Moon, T.M.K. Roeder, C. Macal, M.D. Rossetti (eds.) Proceedings of the 2015 Winter Simulation Conference, pp. 3858–3867. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2015). DOI 10.1109/WSC.2015.7408542
49. Ni, E.C., Ciocan, D.F., Henderson, S.G., Hunter, S.R.: Efficient ranking and selection in parallel computing environments. Operations Research **65**(3), 821–836 (2017). DOI 10.1287/opre.2016.1577
50. Ni, E.C., Henderson, S.G., Hunter, S.R.: A comparison of two parallel ranking and selection procedures. In: A. Tolk, S.D. Diallo, I.O. Ryzhov, L. Yilmaz, S. Buckley, J.A. Miller (eds.) Proceedings of the 2014 Winter Simulation Conference, pp. 3761–3772. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2014). DOI 10.1109/WSC.2014.7020204
51. Ni, E.C., Hunter, S.R., Henderson, S.G.: Ranking and selection in a high performance computing environment. In: R. Pasupathy, S. Kim, A. Tolk, R. Hill, M.E. Kuhl (eds.) Proceedings of the 2013 Winter Simulation Conference, pp. 833–845. IEEE, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ (2013). DOI 10.1109/WSC.2013.6721475
52. Pasupathy, R., Ghosh, S.: Simulation optimization: a concise overview and implementation guide. In: H. Topaloglu (ed.) TutORials in Operations Research, chap. 7, pp. 122–150. INFORMS, Catonsville, MD (2013). DOI 10.1287/educ.2013.0118
53. Pasupathy, R., Hunter, S.R., Pujowidianto, N.A., Lee, L.H., Chen, C.: Stochastically constrained ranking and selection via SCORE. ACM Transactions on Modeling and Computer Simulation **25**(1), 1–26 (2014). DOI 10.1145/2630066
54. Paulson, E.: A sequential procedure for selecting the population with the largest mean from k normal populations. The Annals of Mathematical Statistics **35**(1), 174–180 (1964)
55. Peng, Y., Chen, C., Fu, M.C., Hu, J.: Dynamic sampling allocation and design selection. INFORMS Journal on Computing **28**(2), 195–208 (2016). DOI 10.1287/ijoc.2015.0673
56. Pinedo, M.: Scheduling. Springer (2015)
57. Rinott, Y.: On two-stage selection procedures and related probability-inequalities. Communications in Statistics – Theory and Methods **A7**, 799–877 (1978)
58. Ryzhov, I.O.: On the convergence rates of expected improvement methods. Operations Research **64**(6), 1515–1528 (2016). DOI 10.1287/opre.2016.1494
59. Schmeiser, B.: Batch size effects in the analysis of simulation output. Operations Research **30**(3), 556–568 (1982)
60. Singham, D.I., Szechtman, R.: Multiple comparisons with a standard using false discovery rates. In: Proceedings of the 2016 Winter Simulation Conference, pp. 501–511. IEEE Press (2016)
61. Turnquist, M.A., Sussman, J.M.: A Bayesian approach to the design of simulation experiments. In: H.J. Highland, T.J. Schriber, R.G. Sargent (eds.) Proceedings of the 1976 Bicentennial Winter Simulation Conference, pp. 59–64 (1976). URL http://informs-sim.org/wsc76papers/1976_0011.pdf
62. Wang, H., Pasupathy, R., Schmeiser, B.W.: Integer-ordered simulation optimization using R-SPLINE: Retrospective Search using Piecewise-Linear Interpolation and Neighborhood Enumeration. ACM Transactions on Modeling and Computer Simulation **23**(3) (2013). DOI 10.1145/2499913.2499916
63. Xu, J., Nelson, B.L., Hong, L.J.: Industrial Strength COMPASS: A comprehensive algorithm and software for optimization via simulation. ACM Transactions on Modeling and Computer Simulation **20**, 1–29 (2010)
64. Yoo, T., Cho, H., Yücesan, E.: Web services-based parallel replicated discrete event simulation for large-scale simulation optimization. Simulation **85**(7), 461–475 (2009)