Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

EVALUATION OF bi-PASS FOR PARALLEL SIMULATION OPTIMIZATION

Linda Pei
Barry L. Nelson

Dept. of Industrial Engr. & Management Sciences
Northwestern University
Evanston, IL 60208-3119, USA

Susan R. Hunter

School of Industrial Engineering
Purdue University
West Lafayette, IN 47907, USA

ABSTRACT

Cheap parallel computing has greatly extended the reach of ranking & selection (R&S) for simulation optimization. In this paper we present an evaluation of bi-PASS, a R&S procedure created specifically for parallel implementation and very large numbers of system designs. We compare bi-PASS to the state-of-the-art Good Selection Procedure and an easy-to-implement subset selection procedure. This is one of the few papers to consider both computational and statistical comparison of parallel R&S procedures.

1 INTRODUCTION

Simulation optimization (SO) uses simulation output to estimate the performance of feasible solutions or "systems" to identify an optimal—or at least good—system design. Ranking and selection (R&S) SO procedures simulate *every* feasible system and therefore can provide global statistical error control or guarantees. For instance, frequentist R&S procedures often guarantee the probability of correct selection, while Bayesian R&S procedures provide posterior inference about the quality of the selection. See Kim and Nelson (2006) and Frazier (2010) for surveys. This paper considers the frequentist parallel R&S setting when there are a very large number of systems k.

Parallel R&S procedures use multiple processors to solve SO problems more quickly compared to serial procedures run on a single processor. Parallel R&S is not as simple as taking procedures originally created for a single processor and adapting them to a parallel setting, because parallel procedures must coordinate tasks and manage information among all processors.

This paper carefully evaluates a frequentist parallel R&S procedure called the the bisection Parallel Adaptive Survivor Selection (bi-PASS) algorithm. bi-PASS operates in a master-worker framework, exploiting a "master" processor checking systems for elimination and $p > 1$ parallel "worker" processors carrying out simulations. We highlight two key advantages: First, bi-PASS statistically *benefits* rather than suffers from large k, whereas many conventional R&S procedures become prohibitively conservative as k grows. Second, the structure of bi-PASS reaps computational savings by keeping all workers busy.

In this paper, we conduct an empirical evaluation of bi-PASS. Our empirical evaluation demonstrates that bi-PASS maintains its statistical guarantees on large-scale, realistic problems using a practical implementation. Previous work in Pei et al. (2018) and Pei et al. (2020) proves the theoretical soundness of bi-PASS under conditions that require more synchronization than would be computationally prudent. We compare bi-PASS to a parallel version of subset selection based on Nelson et al. (2001), and to the state-of-the-art Good Selection Procedure (GSP) of Ni et al. (2017). We study two large-scale, realistic problems: an inventory optimization problem with 22,500 systems and a throughput maximization problem with 216,600 systems. Since we study three procedures that have different objectives and statistical guarantees, we do not intend to show that bi-PASS dominates with respect to every metric. Rather, we highlight differences in the procedures and obtain insight into the settings in which bi-PASS is advantageous.

The paper is organized as follows: Section 2 introduces the notation and guarantees of bi-PASS, GSP, and subset selection, Section 3 provides pseudocode for the three procedures, then compares them with respect to parallelization, and Section 4 showcases numerical results.

2 SETTING AND STATISTICAL GUARANTEES

Suppose that we have k systems, indexed by $i \in \mathsf{I} = \{1,2,\ldots,k\}$, with simulation output on replication r, Y_{ir}. We assume that Y_{ir} are i.i.d. $\mathrm{N}(\mu_i, \sigma_i^2)$. We are maximizing, so larger means are better, and unknown to us, the true means satisfy $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_{k-1} \geq \mu_k$. Notice that ties are permitted. We have p "worker" processors, and one "master" processor, giving $p+1$ processors in total.

We let $\tau \geq 0$ denote the "global" or wall-clock time elapsed in a trial of an SO algorithm. For each $i \in \mathsf{I}$, we let $n_i(\tau)$ be system i's "local" time, which is the number of replications accumulated for system i by global time τ. The distinction between global and time is important because local time does not account for the wall-clock time that elapses due to calculations, simulation, message passing among processors, etc., all critical components of executing a R&S procedure in parallel. Serial procedures commonly measure efficiency in terms of total number of replications expended, but examining wall-clock time in parallel procedures is important to capture algorithmic overhead.

Further, let $R_i(\tau) = \sum_{r=1}^{n_i(\tau)} Y_{ir}$ be the running sum of system i's output up to global time τ, and let $\bar{Y}_i(\tau) = \sum_{r=1}^{n_i(\tau)} Y_{ir}/n_i(\tau) = R_i(\tau)/n_i(\tau)$ be the corresponding sample mean. Let τ_{end} be the termination time of a procedure. Common to bi-PASS, GSP, and subset selection is a set $\mathsf{G} \subseteq \mathsf{I}$ of "good systems," an acceptable level of error $\alpha \in (0, 1/2)$, and a terminal set of systems delivered by the procedure, $\mathsf{Q}(\tau_{\text{end}}) \subseteq \mathsf{I}$. Each algorithm has a non-increasing set of "contenders" or "surviving systems" $\mathsf{Q}(\tau)$ for $0 \leq \tau \leq \tau_{\text{end}}$ which consists of systems that have not been eliminated by time τ.

- For bi-PASS (Pei et al. 2018), $\mathsf{G} = \{i\colon \mu_i \geq \mu^\star\}$, where $\mu^\star$ is a standard. In this paper $\mu^\star = \mu_1$, the unknown value of the best system's mean, but other definitions are possible. bi-PASS controls the expected false elimination rate $\mathrm{EFER} = 1 - \mathbb{E}|\mathsf{G} \cap \mathsf{Q}(\tau_{\text{end}})|/|\mathsf{G}| \leq \alpha$, which becomes no more difficult to maintain as k increases.
- For GSP (Ni et al. 2017), $\mathsf{G} = \{i\colon \mu_i \geq \mu_1 - \delta\}$, where $\delta > 0$ is an allowable optimality gap. GSP returns a singleton set $|\mathsf{Q}(\tau_{\text{end}})| = 1$ and guarantees the probability of a good selection $\mathrm{PGS} = \Pr\{\mathsf{Q}(\tau_{\text{end}}) \subseteq \mathsf{G}\} \geq 1 - \alpha$. To guarantee a single system, GSP employs Rinott's procedure (Rinott 1978) in its final stage. GSP demands more output data to provide its guarantee as k increases because PGS is a family-wise error statement.
- For subset selection (Nelson et al. 2001), $\mathsf{G} = \{i\colon \mu_i \geq \mu_1\}$; the procedure returns a subset that guarantees the probability of inclusion $\mathrm{PI} = \Pr\{|\mathsf{Q}(\tau_{\text{end}}) \cap \mathsf{G}| \geq 1\} \geq 1 - \alpha$, which is also a family-wise statement.

The quantities above indexed by τ are random and will change from trial to trial of the same procedure, due to stochasticity of simulation output, simulation execution times, and message passing times.

3 PSEUDOCODE AND COMPARISONS

In this section, we provide pseudocode for our bi-PASS, GSP, and subset selection implementations in a master-worker framework, and compare and contrast the computational characteristics of all three algorithms.

In the master-worker framework, the master and workers execute their respective code in parallel, and the master assigns "jobs" to workers, who report back upon job completion (Hunter and Nelson 2017). We assume processors do not share memory, so master-worker message passing is required to share information. A job is either a simulation job or a screening job, and there is exactly one simulation job for each contender and at most one screening job for each worker. A job is characterized by the message that the master sends to the worker, which contains the data required to complete the job. We say a worker is "busy" if processing a simulation or screening job and is "idle" otherwise. We assume that there are many more systems that workers, so that $k \gg p$.

`Send` and `Receive` are the two methods that comprise message passing. Both methods are "blocking" until a message is successfully passed due to another processor posting a matching `Receive` and `Send`, respectively. This means that processors will not continue executing subsequent code until they are able to send their message to an appropriate destination or until they receive a message from an appropriate source. We also use `Receive` to include the action of the receiving master or worker updating its copies of the received statistics. For example, when the master executes `Receive`(i, R_i, n_i), it also updates its (old) copies of R_i and n_i to be the new updated values returned by the worker.

Whenever we refer to a variable in the pseudocode, we refer to its "current" value at the global time at that point in the algorithm. For simplicity, we omit each variable's notational dependence on global time τ. For example, we simplify the running sum $R_i(\tau)$ by denoting it as R_i in the pseudocode.

We assume that any worker can simulate any system i, and that `Simulate`(i,Δ) for $\Delta \in \mathbb{Z}^+$ returns the sum of Δ independent simulation replications from system i. This implies that random number streams are managed properly so that `Simulate`(i,Δ) is independent for all $i \in \mathsf{I}$, for each time the simulation routine is called, and for each worker that calls it.

The three R&S algorithms we evaluate use auxiliary routines, and their pseudocode is given in the appendix. `DataCollectionMaster` and `DataCollectionWorker` obtain statistics from all systems. For bi-PASS and GSP, these routines are used for a "calibration" stage to obtain initial statistics from every system. GSP employs `RinottStageMaster` and `RinottStageWorker` to select a single good system in its endgame.

3.1 bi-PASS Implementation

The bi-PASS algorithm consists of workers asynchronously completing simulation jobs and the master keeping track of statistics on all systems and comparing a system's sample mean to the estimated standard. There is a one-to-one correspondence between simulation jobs and contenders. A contender's job is either in the job queue waiting to be assigned to a worker, or being simulated on exactly one worker. Eliminated systems no longer have jobs. When a worker reports new simulation output after completing job i, the master updates its copies of statistics on system i and returns job i to the back of the job queue. When job i gets to the front of the job queue, the master checks whether system i can be eliminated. If so, job i is deleted, and if not, job i is assigned to the next available worker.

For elimination decisions, we employ the boundary function $g_{\text{PASS}}(t) = g_{\text{PASS}}(t,c) = \sqrt{[c+\log(t+1)]\,(t+1)}$. As described in detail in Fan et al. (2016), a boundary constant $c = c(\alpha, n_0)$ exists that ensures that a mean-zero Brownian motion with variance estimated from n_0 initial observations will remain inside the boundary forever with desired EFER $\leq \alpha$. Since there are significant numerical issues due to truncation error in estimating the false elimination rate across an infinite time horizon, we approximate the value of c. We select a very large cutoff point $n'_{\max}$ and estimate $\hat{c}$ satisfying

$$\mathbb{P}\left\{ \frac{\sum_{r=1}^{n} Z_{ir}}{\widehat{\sigma}^2(n_0)} \leq -g_{\text{PASS}}\left(\frac{n}{\widehat{\sigma}^2(n_0)}, \hat{c}\right) \text{ for some } n \in \mathbb{N},\ n_0 \leq n \leq n'_{\max} \right\} \leq \alpha$$

where $Z_{ir} \overset{\text{i.i.d.}}{\sim} \mathrm{N}(0,1)$ and $\widehat{\sigma}^2(n_0)$ is the sample variance based on the first n_0 observations. We estimate $\hat{c}$ using Monte Carlo simulation and use it in place of c in g_{PASS}.

Algorithms 1 and 2 are asynchronous because the master does not wait for all workers or all jobs to finish before updating the estimated standard and checking elimination for the system at the head of the job queue whenever a worker reports back. Thus, we expect systems to be "out-of-sync," meaning that at any global time τ there will be $i, j \in \mathsf{Q}(\tau)$ such that $n_i(\tau) \neq n_j(\tau)$. Current theoretical results assume perfectly synchronized systems (Pei et al. 2020). However, large k tends to ameliorate the effects of being out-of-sync because the estimated standard will be the average of a large number of replications.

In Algorithm 1, lines 6–14 represent a "pre-loading," in which the master assigns jobs to idle workers. Then the main loop, in lines 15–25, is "worker-driven" because the master reacts to workers reporting back.

Algorithm 1: `bi-PASSMasterRoutine`

```
Input: EFER parameter $\alpha$, initial sample size $n_0$, run length $\Delta$, boundary $g_{\mathrm{PASS}}(t)$ for $t > 0$, number of systems $k$,
       and StoppingCondition.
1  Create job queue $J = \{1,2,\ldots,k\}$ and set of contenders $\mathsf{Q} = \{1,2,\ldots,k\}$
2  DataCollectionMaster(A=$(R_i, \widehat{\sigma}_i^2)$)              // Get running sums and sample variances
3  for $i \in \mathsf{Q}$ do
4      set $n_i = n_0$
5  Compute initial estimated standard $\widehat{\mu}^\star \leftarrow |\mathsf{Q}|^{-1} \sum_{j\in\mathsf{Q}} R_j/n_j$
6  while some worker w idle do   // Assign a job to each idle worker until no jobs or no idling
7      while $|J| > 0$ do
8          Pop first $i$ from head of $J$
9          if $\frac{n_i}{\widehat{\sigma}_i^2}\left(\frac{R_i}{n_i} - \widehat{\mu}^\star\right) \leq -g_{PASS}\left(\frac{n_i}{\widehat{\sigma}_i^2}\right)$ then              // Check elimination
10             remove $i$ from $\mathsf{Q}$ and update $\widehat{\mu}^\star \leftarrow |\mathsf{Q}|^{-1} \sum_{j\in\mathsf{Q}} R_j/n_j$
11         else
12             Send$(i, R_i, n_i, \Delta)$ to worker $w$ and break    // Break inner loop, job has been assigned
13     if $|J| = 0$ then                           // If no more jobs needed, retire idle worker
14         Send(EndAlgorithmMessage) to worker $w$
15 while StoppingCondition not met do                                        // Main bi-PASS loop
16     Receive$(i, R_i, n_i)$ from any worker $w$ that has completed its job $i$
17     Add $i$ to $J$ and update estimated standard $\widehat{\mu}^\star \leftarrow |\mathsf{Q}|^{-1} \sum_{j\in\mathsf{Q}} R_j/n_j$
18     while $|J| > 0$ do
19         Pop first $i$ from head of $J$
20         if $\frac{n_i}{\widehat{\sigma}_i^2}\left(\frac{R_i}{n_i} - \widehat{\mu}^\star\right) \leq -g_{PASS}\left(\frac{n_i}{\widehat{\sigma}_i^2}\right)$ then              // Check elimination
21             remove $i$ from $\mathsf{Q}$ and update $\widehat{\mu}^\star \leftarrow |\mathsf{Q}|^{-1} \sum_{j\in\mathsf{Q}} R_j/n_j$
22         else
23             Send$(i, R_i, n_i, \Delta)$ to worker $w$ and break    // Break inner loop, job has been assigned
24     if $|J| = 0$ then                           // If no more jobs needed, retire idle worker
25         Send(EndAlgorithmMessage) to worker $w$
26 return $\mathsf{Q}$ as set of contenders
```

Algorithm 2: `bi-PASSWorkerRoutine`

```
Input: Simulation models Simulate$(i)$ for $i = 1,2,\ldots,k$ and initial sample size $n_0$ (available on all workers).
DataCollectionWorker(A=$(R_i, \widehat{\sigma}_i^2)$)                // Get running sums and sample variances
while EndAlgorithmMessage not yet received do              // Receive and do simulation jobs
    Receive$(i, R_i, n_i, \Delta)$ from master
    $n_i = n_i + \Delta$
    $R_i = R_i +$ Simulate$(i, \Delta)$
    Send$(i, R_i, n_i)$ to master
```

In our numerical experiments, we also modify bi-PASS slightly so that systems do not get too far out-of-sync: When a job corresponding to system i completes, if it is 2Δ or more replications behind any other system, then it is put at the front of the master queue rather than at the end.

3.2 Modified GSP Implementation

GSP uses a divide-and-conquer approach that assigns each worker a screening *group*. The master distributes simulation jobs to workers, and after all jobs are completed the master sends screening group statistics and "distributed killers" to workers that do pairwise comparisons to eliminate within their respective groups. Later, GSP transitions to a select-the-best stage which applies Rinott's indifference-zone R&S procedure.

The boundary function is $g_{\text{GSP}} = \eta$, where $\eta = \eta(\alpha_1, k)$ is a constant depending on the Type-I error rate α_1 for the sequential elimination stage and the number of systems k. Details for computing η are found in Ni et al. (2017). In the algorithms that follow, a subscript w indicates a worker-specific set. In particular, Q_w is a set of systems assigned to worker w, and E_w is a set of systems eliminated by worker w.

Our implementations in Algorithms 3 and 4 are more synchronized and similar to the `Spark` implementation of GSP in Ni et al. (2017). We use this synchronized version rather than the original for simplicity of implementation and to preserve the one-simulation-job-per-contender structure that allows more straightforward pseudocode comparisons among the algorithms. It is possible that our wall-clock times might be inflated compared to an asynchronous implementation of GSP, although an asynchronous implementation might suffer from increased master computations and message passing. Ni et al. (2017) provide numerical experiments in which their `Spark` GSP implementation is slower than their asynchronous GSP implementation using `MPI` and `C++`, but to our knowledge, timing analysis has not been conducted for a synchronous vs. asynchronous version that both use `MPI` and the same programming language.

Algorithm 3: `GSPMasterRoutine`

```
Input: Good selection probability $1-\alpha$ and error splitting parameters $\alpha_1, \alpha_2$, where $\alpha_1 + \alpha_2 = \alpha$, indifference-zone
       parameter $\delta > 0$, number of distributed killers $d \ll k$, initial sample size $n_0$, maximum number of pre-Rinott
       replications per system $n_{\max}$, and run length $\Delta$.
Create job queue $J = \{1, 2, \ldots, k\}$ and set of contenders $\mathsf{Q} = \{1, 2, \ldots, k\}$
DataCollectionMaster(A=$\bar{T}_i(n_0)$) where $\bar{T}_i(n_0)$ is the average simulation completion time for system $i$ based
  on $n_0$ independent observations                                  // Get timing info for load balancing
DataCollectionMaster(A=$(R_i, \widehat{\sigma}_i^2)$)                       // Get running sums and sample variances
Set common replication counter $n = n_0$
for each $w = 1, 2, \ldots, p$ do
    compute load-balanced screening groups and Send($\mathsf{Q}_w$) to worker $w$
while $n < n_{max}$ do        // Until max Pre-Rinott reps hit, sequentially simulate and screen
    Recreate job queue $J = \{j \colon j \in \mathsf{Q}\}$
    if $p > |\mathsf{Q}|$ then          // If more workers than simulation jobs, retire excess workers
        Send(EndSimulationStageMessage) to any $p - |\mathsf{Q}|$ workers
    while some worker $w$ is idle do                      // Send simulation jobs and receive results
        Send($i, R_i, \Delta$) to worker $w$
    while some worker busy do
        Receive($i, R_i$) from any worker $w$ that has completed its job $i$
        if $|J| > 0$ then
            Send($i, R_i, \Delta$) to worker $w$
        else
            Send(EndSimulationStageMessage) to worker $w$
    Update common replication counter $n = n + \Delta$
    Form the set $\mathsf{D}$ of distributed killers from systems in $\mathsf{Q}$ with the largest $d$ sample means
    for each worker $w$ do                  // Send screening jobs to workers and receive results
        if $|\mathsf{Q}_w| > 0$ then
            Send($\mathsf{D}, \mathsf{Q}_w, \{(i, R_i, n, \widehat{\sigma}_i^2) \colon i \in \mathsf{D}, \mathsf{Q}_w\}$) to worker $w$
        else                    // If worker screening group empty, no screening job needed
            Send(EndScreenStageMessage) to worker $w$
    while some worker busy do
        Receive($\mathsf{E}_w$) from any worker $w$ and update $\mathsf{Q} = \mathsf{Q} \setminus \mathsf{E}_w$ and $\mathsf{Q}_w = \mathsf{Q}_w \setminus \mathsf{E}_w$
        Send(EndScreenStageMessage) to worker $w$
Set $n_i \leftarrow n$ for all $i \in \mathsf{Q}$ RinottStageMaster()
Send(EndAlgorithmMessage) to all $p$ workers
return $\arg\max_{i \in \mathsf{Q}} R_i / N_i$ as the estimated single best system
```

```
Algorithm 4: GSPWorkerRoutine
Input: Simulation models Simulate($i$) for $i = 1,2,\ldots,k$, boundary function $g_{\mathrm{GSP}}$ for $t > 0$, and initial sample size
       $n_0$ (available on all workers).
DataCollectionWorker(A=$\bar{T}_i(n_0)$)                      // Get timing info for load balancing
DataCollectionWorker(A=$(R_i, \widehat{\sigma}_i^2)$)               // Get running sums and sample variances
Receive($\mathsf{Q}_w$) from master
Set elimination status $E_i \leftarrow$ False for all $i \in \mathsf{Q}_w$
while EndAlgorithmMessage not yet received do
    while EndSimulationStageMessage not yet received do    // Receive and do simulation jobs
        Receive($i, R_i, n, \Delta$) from master
        $R_i = R_i +$ Simulate($i, \Delta$)
        Send($i, R_i$) to master
    while EndScreenStageMessage not yet received do           // Receive and do screening job
        Receive($\mathsf{D}, \mathsf{Q}_w, \{(i, R_i, n, \widehat{\sigma}_i^2) \colon i \in \mathsf{D}, \mathsf{Q}_w\}$) from master
        for $i \in \mathsf{Q}_w$ do
            for $j \in \mathsf{D} \cup \mathsf{Q}_w, j \neq i$ do
                if $\frac{n/\sqrt{(n_0-1)n_{max}}}{\sqrt{\widehat{\sigma}_i^2 + \widehat{\sigma}_j^2}} \left(\frac{R_i}{n} - \frac{R_j}{n}\right) \leq -g_{GSP}$ then
                    set $E_i =$ True and break    // Break inner loop, begin checking next system
        Set $\mathsf{E}_w = \{i \in \mathsf{Q}_w : \; E_i = \mathrm{True}\}$
        Send($\mathsf{E}_w$) to master                                   // Report eliminated systems
RinottStageWorker()
```

3.3 Parallel Subset Selection Implementation

Our parallel implementation of subset selection is the simplest of the three procedures we evaluate. Workers complete one round of simulation jobs and then are not used again. After the master receives statistics from all systems, the master does pairwise comparisons to determine which systems to keep in the subset. This is in contrast to bi-PASS and GSP, in which workers successively complete multiple rounds of simulation jobs and communicate back and forth with the master.

The boundary function is simply $g_{\mathrm{Subset}} = t_{(1-\alpha)^{1/(k-1)},\, n_0-1}$, which is the $(1-\alpha)^{\frac{1}{k-1}}$ quantile of the t distribution with $n_0 - 1$ degrees of freedom, as explained in Nelson et al. (2001).

```
Algorithm 5: SubsetSelectionMasterRoutine
Input: Correct inclusion probability $1-\alpha$, initial sample size $n_0$, boundary function $g_{\mathrm{Subset}}$.
Create job queue $J = \{1,2,\ldots,k\}$ and set of contenders $\mathsf{Q} = \{1,2,\ldots,k\}$
Set elimination status $E_i \leftarrow$ False for all $i \in \mathsf{Q}$
DataCollectionMaster(A=$(R_i, \widehat{\sigma}_i^2)$)               // Get running sums and sample variances
for $i \in \mathsf{Q}$ do                                 // Pairwise comparisons to check elimination
    for $j \in \mathsf{Q},\ j \neq i$ do
        if $\sqrt{\frac{n_0}{\widehat{\sigma}_i^2 + \widehat{\sigma}_j^2}} \left(\frac{R_i}{n_0} - \frac{R_j}{n_0}\right) < -g_{Subset}$ then
            set $E_i =$ True and break              // Break inner loop, begin checking next system
Update $\mathsf{Q} = \{i \in \mathsf{Q} \colon E_i = \mathrm{False}\}$
return $\mathsf{Q}$ as selected subset of systems
```

```
Algorithm 6: SubsetSelectionWorkerRoutine
Input: Simulation models Simulate($i$) for $i = 1,2,\ldots,k$ and initial sample size $n_0$ (available on all workers).
DataCollectionWorker(A=$(R_i, \widehat{\sigma}_i^2)$)               // Get running sums and sample variances
```

Table 1: Comparison of computational characteristics of bi-PASS, GSP and subset selection.

	bi-PASS	**GSP**	**Subset Selection**
Simulation	Workers	Workers	Workers
Elimination	Master	Workers	Master
Master Sends	Running sum, # observations, and run length for i	Running sum and run length for i (simulation); Statistics for systems in screening group and distributed killers (screening)	Initial sample size for i
Master Receives	Running sum and # observations for i	Running sum for i (simulation); Systems eliminated by worker (screening)	Running sum and sample variance for i
Master Tasks	Updates estimated standard	Updates distributed killers	Pairwise comparisons

3.4 Discussion

All three procedures make elimination decisions by checking whether a scaled difference between two sample means falls below a boundary.

- bi-PASS eliminates system i if $\frac{n_i}{\widehat{\sigma}_i^2}\left(\frac{R_i}{n_i}-\widehat{\mu}^\star\right) \le -g_{\text{PASS}}\left(\frac{n_i}{\widehat{\sigma}_i^2}\right) = -\sqrt{[c+\log(n_i/\widehat{\sigma}_i^2+1)](n_i/\widehat{\sigma}_i^2+1)}$.
- GSP eliminates system i with system j if $\frac{n/\sqrt{(n_0-1)n_{\max}}}{\sqrt{\widehat{\sigma}_i^2+\widehat{\sigma}_j^2}}\left(\frac{R_i}{n_i}-\frac{R_j}{n_j}\right) \le -g_{\text{GSP}} = -\eta$ and we note that due to our synchronized implementation, the formulation of the elimination condition above is simpler than the one used in Ni et al. (2017).
- Subset selection eliminates system i with system j if $\sqrt{\frac{n_0}{\widehat{\sigma}_i^2+\widehat{\sigma}_j^2}}\left(\frac{R_i}{n_0}-\frac{R_j}{n_0}\right) \le -g_{\text{Subset}} = -t_{(1-\alpha)^{1/(k-1)},\ n_0-1}$.

In Table 1, we compare and contrast computational characteristics of the three procedures. “Simulation” and “elimination,” respectively, refer to whether simulation and elimination are completed on the master or the worker processors. “Master sends” refers to the data passed to a worker when assigning a job i, and “master receives” refers to the data received from a worker which has just completed its job i. Since the master in GSP assigns two types of jobs, we describe simulation job data first and elimination job data second. Notice that for bi-PASS and GSP, when the first n_0 observations are collected, the master will also receive the sample variance $\widehat{\sigma}_i^2$. “Master tasks” refer to the main computations that the master carries out beyond job assignment and completed job data collection.

In terms of message passing overhead per system, GSP suffers the most. Consider the quantity of jobs needed for each system to obtain exactly n_0 observations and check systems for elimination exactly once. For bi-PASS and subset selection, this requires k jobs. For GSP, this requires $k+p$ jobs, due to worker screening. Additionally, the master in GSP must pass a very large amount of data to workers for screening. The other two algorithms only require the master to send a handful of scalar values to each worker for each job. GSP also requires more “maintenance” messages than the other two algorithms due to the master in GSP needing to tell workers when to switch from simulation to screening. Subset selection is clearly very efficient in limiting the amount of message passing, and in fact only k jobs are required for the entire algorithm. In this sense, GSP and subset selection represent extremes: GSP incurs a higher message passing cost in an attempt to eliminate systems quickly, whereas subset selection incurs minimal message passing cost with a naive allocation of observations that likely results in inefficient expenditure of simulation effort.

Next, we discuss worker processor idling. All bi-PASS jobs are of the same type and require minimal data to be passed when sent to workers, and updating the estimated standard is a simple operation. These factors are designed to keep the master mostly idle and the workers mostly busy in bi-PASS. In contrast, during the comparison stage of subset selection the workers are idle and the master does all the work. When there are fewer remaining systems than processors, bi-PASS does not make use of these idle processors. Later in the algorithm, and generally when there are few systems left, GSP moves from sequential simulation and elimination to a Rinott stage in which a final run on each contender is completed and the contender with the largest sample mean is chosen as the best. This move is strategic because when there are few contenders we expect them to be similar in performance, and it might be computationally better to spend less wall-clock time on message passing and more on extra observations to distinguish similar means.

4 EXPERIMENTS

In this section we describe the computing environment, experiment design, and empirical evaluation of bi-PASS, GSP, and subset selection on two large-scale realistic problems.

4.1 Cluster Specifications and Implementation Details

We ran our experiments on Northwestern University's Optimization and Statistical Learning (OSL) group cluster. The cluster runs a Linux CentOS 7.4 operating system. For each experiment trial, we used two compute nodes, each with two 40-core Intel(R) Xeon(R) Gold 6148 processors with 256 GB of RAM and 1 TB of hard drive memory. No other jobs were being executed on the nodes while we ran our experiments, to ensure that the timing of the experiments was not affected by other processes competing for node resources.

We evaluate parallel bi-PASS, GSP, and subset selection based on the pseudocode in the previous section as implemented in Python, and using the package `mpi4py`, which has Python bindings to `Open MPI`, for parallelization.

We skip the load-balancing stage in GSP because the simulation execution times for the systems are about the same, so that our version of GSP already starts with load-balanced groups without incurring the costs of the additional observations normally needed for this stage. This gives GSP an advantage in our comparisons because in the usual implementation all systems are simulated to gain timing information, and then these data are discarded.

4.2 Inventory Cost Minimization Problem

The first set of experiments is applied to an (s,S) inventory problem with 22,500 policies. This problem is outlined in Semelhago et al. (2017). The objective is to maximize the negative expected value of average cost per period over 30 periods, where the decision variables are s and $S-s$, constrained to $s,S-s \in \{1,2,\dots,150\}$. Demand per period is i.i.d. Poisson with mean 25, and the unit costs for holding and backorder are 1 and 5 per period, respectively. The unique best solution of $(s,S) = (18,53)$ has a true mean that is \$106.17 and \$0.01 better than the second-best solution.

We compare how the three procedures perform when given roughly the same amount of replications. We ran 10 independent trials of GSP on the inventory problem; 10 trials made the standard error sufficiently small for our performance statistics. Let $\mathtt{total}_m$ be the total number of replications in trial m of GSP, for $m = 1,2,\dots,10$. For trial m of bi-PASS, once $\mathtt{total}_m$ replications have been reached, jobs in progress are allowed to finish, but the master assigns no new jobs, and the algorithm terminates. For trial m of subset selection, we equally allocate $\mathtt{total}_m$ among all systems and set $n_0 = \lceil \mathtt{total}_m/k \rceil$ for each system.

The parameters for the experiments are as follows. For bi-PASS, GSP, and subset selection, the error parameter is $\alpha = 0.05$. For bi-PASS, $n_0 = 10$ and $\Delta = 10$. For GSP, $\alpha_1 = \alpha_2 = \alpha/2$, $\delta = \$1$, $d = 20$, $n_{\max} = 100$, $n_0 = 10$, and $\Delta = 10$. For subset selection, $n_0 = \lceil \mathtt{total}_m/k \rceil$ as mentioned above. We use $\delta = \$1$ for GSP instead of the true difference of \$0.01 between the best and second best system to be advantageous to GSP. Neither bi-PASS nor subset selection require an indifference-zone parameter δ; the number of observations required in the Rinott stage of GSP is proportional to $1/\delta^2$, so if it had been set

Table 2: Inventory problem performance comparison of bi-PASS, GSP, and subset selection based on 10 trials, shown with standard errors, with a stopping condition for each trial of bi-PASS and subset based on the total reps GSP accumulates.

	Contenders	Total Reps	Time (sec)
bi-PASS	8 ± 1	$1.2e6 \pm 9e4$	45.8 ± 11
GSP Pre-Rinott	580 ± 12	$5.0e5 \pm 2e4$	70.7 ± 0.4
GSP Complete	1	$1.2e6 \pm 9e4$	75.0 ± 0.7
Subset	215 ± 14	$1.2e6 \pm 8e4$	515 ± 3.6

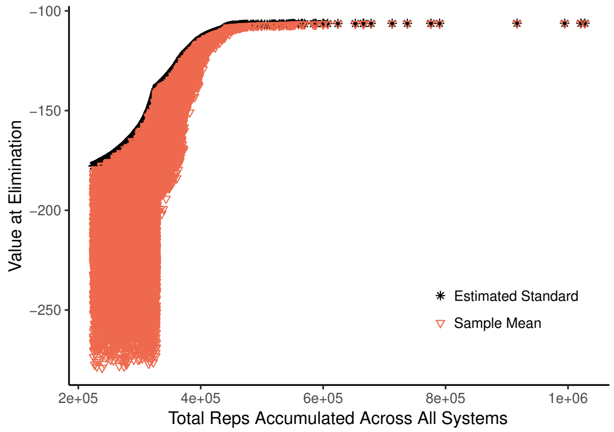

Figure 1: bi-PASS inventory problem sample path: when an elimination occurs, we note the sample mean of the eliminated system, estimated standard value, and total reps.

to, say, \$0.01, this stage would take significantly longer. For $\delta = \$1$, there are 57 “good” systems. Run length Δ was chosen to avoid the master being overwhelmed. Parameters n_0 and $n_{\max}$ were chosen based on previous empirical success in (Ni et al. 2017) using initial sample sizes that are multiples of 10 and a maximum number of “rounds” or jobs of 10, but other combinations of n_0, Δ, and $n_{\max}$ might be effective.

Out of 10 trials, we observed 0 false eliminations for bi-PASS, 0 instances of GSP failing to select a good system, and 0 instances of subset selection failing to include the best system. In 2 of the 10 trials, GSP selected the unique best system.

In Table 2, we describe summary results averaged across the 10 trials with their standard errors. “Contenders” refers to the number of contenders (non-eliminated systems) at trial termination. For GSP, we report statistics for the sequential elimination part of the procedure (called “GSP Pre-Rinott”) as well as the total time elapsed after the final Rinott stage to select a single system (called “GSP Complete”). Compared to both bi-PASS and GSP, subset selection performs much worse on this problem. Subset selection takes an order of magnitude longer to run compared to bi-PASS, due to the master having to execute $O(k^2)$ comparisons, while also leaving behind an order of magnitude more contenders than bi-PASS due to the naive equal allocation. GSP selects exactly 1 system in each trial due to the Rinott stage. GSP pays a price in terms of time, however, and takes about 60% longer than bi-PASS, although bi-PASS has an average of about 8 contenders remaining. The speed of the Rinott stage is due to minimal message passage requirements, since at most one simulation job is needed per contender at this stage.

We conclude the inventory problem discussion with a sample path from bi-PASS. In Figure 1, when a system gets eliminated, we plot the sample mean of that system, the value of the estimated standard, and the total reps accumulated, all at the time of elimination. Note that the estimated standard is always greater than the sample mean of any system it eliminates.

The estimated standard initially increases rapidly, then increases more slowly and starts to level off. Correspondingly, systems are initially eliminated in large swathes, then in less frequent intervals. The plotted bi-PASS sample path terminated after about $1e6$ total reps. At the halfway point, at the first $5e5$ total reps, all but 68 systems were eliminated. This is an order of magnitude less systems compared to the average GSP Pre-Rinott results in Table 2. However, bi-PASS only eliminated 63 systems in the next $5e5$ total reps compared to eliminating 22,432 in the first $5e5$ total reps. As seen in Figure 1, towards the end, eliminations become sporadic and infrequent. In general we expect behavior like this in a bi-PASS sample path, because bi-PASS eliminations slow down significantly when there are few contenders remaining. This

Table 3: Throughput problem performance comparison of bi-PASS and GSP based on 10 trials, shown with standard errors, with a stopping condition based on each contender accumulating 100 reps (i.e. 10 jobs of 10 reps each, including the n_0 job).

	Contenders	**Total Reps**	**Time (sec)**
bi-PASS	2280 ± 8	$3.9e6 \pm 7.5e2$	$5.1e3 \pm 1e2$
GSP	3100 ± 170	$5.7e6 \pm 1.5e5$	$12.1e3 \pm 6e2$

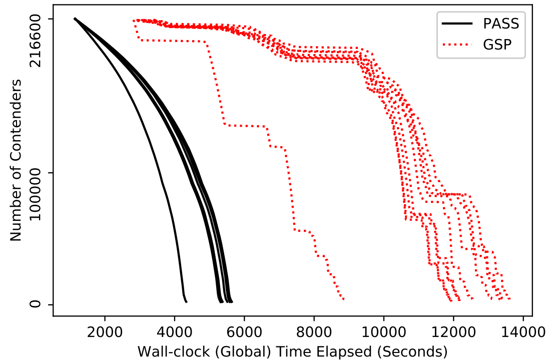

Figure 2: Throughput problem: 10 sample paths of bi-PASS (solid black) and GSP (dotted red) of contenders-vs-time.

underscores that bi-PASS might not be an ideal "endgame" algorithm, because it becomes less efficient towards the end when there are few contenders left.

4.3 Throughput Maximization Problem

The second set of experiments is a throughput maximization problem with 216,600 solutions. This is a larger version of the `Throughput Maximization` problem from `SimOpt.org` for which the constraints were strict equalities; we changed the constraints to be inequalities to enlarge the feasible region without changing the optimal solutions. There are 3 stations in a tandem queue with an infinite number of jobs in front of station 1. Each station i has a single server with exponential service rate r_i for $i = 1,2,3$. Stations 2 and 3 have buffer sizes b_2 and b_3, respectively, with buffer sizes including the server. If Station i has a full buffer, then manufacturing blocking occurs and Station $i-1$ cannot release a finished job. Feasible solutions $(r_1, r_2, r_3, b_2, b_3) \in \mathbb{Z}^+$ satisfy $r_1 + r_2 + r_3 \leq 20$ and $b_2 + b_3 \leq 20$. For each solution, output from one simulation replication is generated by estimating throughput based on 50 jobs after a 2000 job warm-up period. Both $(6,7,7,12,8)$ and $(7,7,6,8,12)$ are optimal with expected throughput of 5.776 per unit time.

For the throughput maximization problem, we compare the screening capabilities of bi-PASS and GSP when given roughly the same stopping condition; subset selection is too slow to be included in this large-scale experiment. We ran 10 independent trials of GSP without Rinott since we are primarily interested in how effectively each procedure can eliminate inferior systems with a modest simulation effort. In contrast to the inventory problem, here we terminate bi-PASS and GSP after each contender has accrued at least 100 observations. Due to the synchronous nature of this implementation of GSP, each contender will have exactly 100 observations at termination.

The parameters for the two algorithms are the same as for the inventory problem, unless explicitly stated. Since we do not use the Rinott stage for this problem, we do not need an indifference-zone parameter δ for GSP, and we do not need to split the error over two stages, so $\alpha_1 = \alpha$ for GSP.

Out of 10 trials, we observed 0 false eliminations or incorrect selection events for both procedures, as in the previous set of experiments. Table 3 records the average time, number of contenders remaining, and total number of replications, as well as their standard errors. bi-PASS is able to reduce the set of contenders more quickly than GSP with more efficient usage of both simulation replications and wall-clock time.

In Figure 2, we plot 10 sample paths of the number of contenders remaining versus wall-clock time elapsed for bi-PASS and GSP. In practice, GSP's load-balancing stage would further delay the start of eliminations. Figure 2's information is slightly different from Table 3, due to increased I/O overhead needed to save the sample paths. The difference between master elimination in bi-PASS and worker group screening in GSP is reflected in the delays and jaggedness of the GSP curves. In bi-PASS, the master checks elimination for each system in the job queue one-by-one. In GSP, the master records the number of contenders after each time a worker finishes a screening job, and often multiple systems get eliminated at once. bi-PASS starts eliminating immediately after each system obtains n_0 observations, and all of its paths begin before the paths of GSP, which is delayed due to waiting for each worker to finish a large number of pairwise comparisons. Also, the bi-PASS curves have much less variance compared to GSP.

ACKNOWLEDGEMENTS

Nelson's work is partially supported by National Science Foundation Grant Number DMS-1854562. Hunter's work is partially supported by National Science Foundation grant CMMI-1554144.

APPENDIX

```
Algorithm 7: DataCollectionMaster(A)
while some worker w idle do
    Pop first $i$ from head of $J$ and Send($i$) to worker $w$
while some worker w busy do
    Receive($i$,A) from any worker $w$ that has completed its job $i$
    if $|J| > 0$ then
        pop first $i$ from head of $J$ and Send($i$) to worker $w$
    else
        Send(EndDataCollectionMessage) to worker $w$
```

```
Algorithm 8: DataCollectionWorker(A)
while EndDataCollectionMessage not yet received do
    Receive($i$) from master
    Initialize set of outputs $L = \emptyset$
    for $n_i = 1, 2, \ldots, n_0$ do
        Simulate($i$, 1) and store in set $L$
    Compute all statistics in A from observations in $L$
    Send($i$,A) to master
```

```
Algorithm 9: RinottStageMaster()
Compute Rinott's constant $h = h(1-\alpha_2, n_0, k)$
for each $i \in$ Q do
    Compute $N_i = \max\{n_{\max}, \lceil (h\widehat{\sigma}_i/\delta)^2 \rceil\}$
while some worker w idle do
    Pop first $i$ from head of $J$ and Send($i, R_i, N_i - n_i$) to worker $w$
while some worker w busy do
    Receive($i, R_i$) from any worker $w$ that has completed its job $i$
    if $|J| > 0$ then
        pop first $i$ from head of $J$ and Send($i, R_i, N_i - n_i$) to worker $w$
    else
        Send(EndRinottStageMessage) to worker $w$
```

```
Algorithm 10: RinottStageWorker()
while EndRinottStageMessage message not yet received do
    Receive($i, R_i, N_i - n_i$) from master
    $R_i = R_i +$ Simulate($i, N_i - n_i$)
    Send($i, R_i$) to master
```

REFERENCES

Fan, W., L. J. Hong, and B. L. Nelson. 2016. “Indifference-Zone-Free Selection of the Best”. *Operations Research* 64(6):1499–1514.

Frazier, P. I. 2010. “Decision-theoretic Foundations of Simulation Optimization”. In *Wiley Encyclopedia of Operations Research and Management Sciences*. New York: Wiley.

Hunter, S. R., and B. L. Nelson. 2017. “Parallel Ranking and Selection”. In *Advances in Modeling and Simulation*, edited by A. Tolk et al., Chapter 12, 249–275. New York: Springer.

Kim, S.-H., and B. L. Nelson. 2006. “Selecting the Best System”. In *Handbooks in Operations Research and Management Science*, edited by S. G. Henderson and B. L. Nelson, Volume 13, Chapter 17, 501–534. New York: Elsevier.

Nelson, B. L., J. Swann, D. Goldsman, and W. Song. 2001. “Simple Procedures for Selecting the Best Simulated System when the Number of Alternatives is Large”. *Operations Research* 49(6):950–963.

Ni, E. C., D. F. Ciocan, S. G. Henderson, and S. R. Hunter. 2017. “Efficient Ranking and Selection in Parallel Computing Environments”. *Operations Research* 65(3):821–836.

Pei, L., S. R. Hunter, and B. L. Nelson. 2020. “Parallel Adaptive Survivor Selection”. Technical report, Department of Industrial Engineering & Management Sciences, Northwestern University, Evanston, IL.

Pei, L., B. L. Nelson, and S. Hunter. 2018. “A New Framework for Parallel Ranking & Selection using an Adaptive Standard”. In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 2201–2212. Piscataway, New Jersey: IEEE.

Rinott, Y. 1978. “On Two-Stage Selection Procedures and Related Probability-Inequalities”. *Communications in Statistics-Theory and methods* 7(8):799–811.

Semelhago, M., B. L. Nelson, A. Wächter, and E. Song. 2017. “Computational Methods for Optimization via Simulation using Gaussian Markov Random Fields”. In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D’Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 2080–2091. Piscataway, New Jersey: IEEE.

AUTHOR BIOGRAPHIES

LINDA PEI is a Ph.D. student in the Department of Industrial Engineering and Management Sciences at Northwestern University. Her research interests are simulation optimization and simulation analytics. Her e-mail address is lindapei2016@u.northwestern.edu.

SUSAN R. HUNTER is an assistant professor in the School of Industrial Engineering at Purdue University. Her research interests include Monte Carlo methods and simulation optimization, especially in the presence of multiple performance measures. Her e-mail address is susanhunter@purdue.edu, and her website is http://web.ics.purdue.edu/~hunter63/.

BARRY L. NELSON is the Walter P. Murphy Professor in the Department of Industrial Engineering and Management Sciences at Northwestern University. He is a Fellow of INFORMS and IIE. His research centers on the design and analysis of computer simulation experiments on models of stochastic systems, and he is the author of *Foundations and Methods of Stochastic Simulation: A First Course*, from Springer. His e-mail address and website are nelsonb@northwestern.edu and http://www.iems.northwestern.edu/~nelsonb/, respectively.