Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Speeding Up Paulson's Procedure for Large-Scale Problem Using Parallel Computing*

Ying Zhong
School of Management and Economics, University of Electronic Science and Technology of China, Chengdu, China, yzhong4@uestc.edu.cn

Shaoxuan Liu, Jun Luo
Antai College of Economics and Management, Shanghai Jiao Tong University, Shanghai, China, liusx@sjtu.edu.cn, jluo_ms@sjtu.edu.cn

L. Jeff Hong†
School of Management and School of Data Science, Fudan University, Shanghai, China, hong_liu@fudan.edu.cn

With the rapid development of computing technology, using parallel computing to solve large-scale ranking-and-selection (R&S) problems has emerged as an important research topic. However, direct implementation of traditionally fully sequential procedures in parallel computing environments may encounter various problems. First, the scheme of all-pairwise comparisons, which is commonly used in fully sequential procedures, has an $\mathcal{O}(k^2)$ computational complexity, and significantly slows down the selection process. Second, traditional fully sequential procedures require frequent communication and coordination among processors, which are also not efficient in parallel computing environments. In this paper, we propose three modifications on one classical fully sequential procedure, Paulson's procedure, to speed up its selection process in parallel computing environments. First, we show that if no common random numbers (CRNs) are used, we can reduce the computational complexity of the all-pairwise comparisons at each round to $\mathcal{O}(k)$. Second, by batching different alternatives, we show that we can reduce the communication cost among the processors, leading the procedure to achieve better performance. Third, to boost the procedure's final-stage selection, when the number of surviving alternatives is less than the number of processors, we suggest to sample all surviving alternatives to the maximal number of observations they should take. We show that after these modifications, the procedure remains statistically valid, and is more efficient compared with existing parallel procedures in the literature.

Key words: ranking-and-selection, fully sequential procedures, parallel computing, computational complexity

History: This version: November 11, 2020

1. Introduction

In computer simulation, the ranking-and-selection (R&S) problem is to select the best from a finite set of alternatives, where the best is defined as the alternative with the largest or smallest mean performance measure. Many R&S procedures have been developed to solve this type of problems. In general, existing procedures can be classified into two categories, frequentist procedures and Bayesian procedures. Interested readers may refer to Kim and Nelson (2006) and Chick (2006) for comprehensive reviews on both theoretical achievements and procedure designs for these two classes of procedures, as well as Hong et al. (2015) and Chen et al. (2015) for additional developments in recent years. In this paper, we take a frequentist point of view to investigate potential issues, from both theoretical and implementational aspects, as solving large-scale R&S problems in parallel computing environments.

Under the frequentist framework, R&S procedures are often designed with the aim to select the true best with a predefined *probability of correct selection* (PCS) even under the least favorable configuration of the means. In order to achieve this goal, various procedures, e.g., either stage-wise procedures of Bechhofer (1954), Rinott (1978), and Nelson et al. (2001) or fully sequential procedures of Paulson (1964), Kim and Nelson (2001), and Hong (2006), have been designed in the literature. Stage-wise procedures typically determine the sample size of each alternative at the beginning of the selection process. Then, the procedures take all the observations at once and simply pick the alternative with the largest or smallest sample mean as the best. Meanwhile, fully sequential procedures often proceed in rounds and sequentially eliminate alternatives upon collecting enough evidence.

When these procedures were first designed, they were used to solve problems with relatively small numbers of alternatives. For instance, the early procedures of Rinott (1978) and Paulson (1964) were applied to problems with fewer than 20 alternatives, and the procedures of Nelson et al. (2001) and Kim and Nelson (2001) were used to solve problems with up to 500 alternatives, as reported in their papers. A critical reason that restricts these R&S procedures from dealing with large-scale problems is the limited computing power of

[*] A preliminary version of this paper, i.e., Hong et al. (2016), was published in the Proceedings of the 2016 Winter Simulation Conference, which investigated only the issue of all-pairwise comparisons for the KN procedure in a single-processor environment.

[†] L. Jeff Hong is the corresponding author.

a single processor. Even though simulating a single observation from each alternative may be very fast, a R&S procedure needs to simulate all alternatives for many times in order to obtain meaningful estimations and elimination decisions. As a result, these procedures are practically only suitable for relatively small-scale problems.

With the rapid development of computing technology, parallel computing environments are becoming prevalent and ready for ordinary users to access. The fact that R&S procedures "make no attempt to exploit relationships among the solutions" (Hong et al. 2015), allows the simulation program from different alternatives to be executed independently on different processors with few synchronizations. It makes parallel computing very attractive for large-scale R&S problems. In terms of the selection structure, stage-wise procedures are much simpler and more suitable for parallel computing environments than fully sequential procedures. For stage-wise procedures, once the sample size of each alternative is determined, the simulation tasks can be independently executed on different processors without any synchronizations. After all simulation tasks are finished, the procedures only need to make one round of sample mean comparisons to select the best. Meanwhile, many existing works focus on designing parallel fully sequential procedures because they allow early eliminations of clearly inferior alternatives and thus require much smaller sample sizes than stage-wise procedures do. As demonstrated by two recent works of Luo et al. (2015) and Ni et al. (2017), with the gain on the computing power in parallel computing environments, the authors can use fully sequential procedures to solve large-scale R&S problems with more than $20,000$ and even up to 10^6 alternatives. In the meantime, these works also discover that new issues from both theoretical and implementational aspects appear as one directly implements fully sequential procedures in parallel computing environments. Some of these issues can greatly affect the procedures' performance.

Among them, the most critical one is the high computational complexity of comparisons. For fully sequential procedures, in order to efficiently eliminate clearly inferior alternatives, whenever every surviving alternative receives a new observation, the procedures compare each alternative with all other surviving ones to check whether it may be eliminated or not, i.e., all-pairwise comparisons. When the problem size, i.e., the number of alternatives, is relatively small, the computational time of comparisons is often negligible since making a comparison between two alternatives is very fast. When the problem size is large, however, the procedures spend a significant amount of time on conducting this type of comparisons,

because the computational complexity of comparisons increases much faster than that of simulations and the comparisons cannot be distributed to multiple processors. As the problem size keeps increasing, the overall comparison time becomes the bottleneck of the procedures, and it cannot be shortened by leveraging the parallel computing.

Besides the high computational complexity of comparisons, another key issue for parallel computing is the communication cost. Notice that the essence of parallel computing is to decompose a big task into small ones and execute them on different processors. While executing these tasks, the processors need to coordinate with each other and spend time on communications. Typically, the way to execute the tasks is not unique. One can assign the small tasks one-by-one to the next available processor (i.e., in a round-robin manner) or batch the small tasks into a few big ones and then assign. Because the completion time of a task is often random, assigning a small-size task at a time can help a procedure balance the workloads of different processors and ensure all processors to finish the tasks at roughly the same time. However, sequentially assigning a large number of tasks often means frequent communications among the processors. Therefore, balancing the workloads of different processors and reducing the frequency of communications are typically conflicting goals in parallel computing. For parallel fully sequential procedures, while assigning simulation tasks at each round, one should carefully consider the trade-off between these two goals. Especially, when the number of alternatives is large, the existing way (see, for example, Luo and Hong (2011) and Luo et al. (2015)) to assign the simulation tasks one-by-one in a round-robin manner can lead to a high communication cost. A good task assignment scheme should reduce the communication cost in this situation.

So far, some efforts have been made to address the aforementioned issues. In the work of Ni et al. (2017), the authors propose a "divide-and-conquer" approach to reduce the computational complexity of comparisons. They also suggest simulating a batch of observations for one alternative at a time to reduce the frequency of communications and thus the communication cost. By doing so, the authors show that the proposed procedure dramatically increases the sizes of solvable problems. However, we want to point out that there is still much room for improvement because these modifications are made at the expense of efficiency loss on the total sample size, and the computational complexity of comparisons in the proposed procedure is still higher than that of simulations.

In this paper, we choose one classical fully sequential procedure, Paulson's procedure (Paulson 1964), and discuss how to extend it to parallel computing environments. We show that if no common random numbers (CRNs) are used, with a few modifications on the procedure, we can significantly improve the efficiency of the procedure in parallel computing environments. First, by rearranging (with slight modifications) the terms in the comparisons, we can significantly reduce the computational complexity of comparisons to the same order as that of simulations, while the feature of all-pairwise comparisons remains intact. Second, by batching of different alternatives (rather than batching of observations from the same alternative as in Ni et al. (2017)), we can significantly reduce the wall-clock time without sacrificing the efficiency on the total sample size. Lastly, we propose to sample all surviving alternatives to the maximal sample size suggested by Paulson's procedure when the number of surviving alternatives is less or equal to the number of processors used to conduct simulations, which can significantly reduce the time spent on a few remaining alternatives empirically observed by Luo et al. (2015). It is worthwhile noting that the first two modifications may be applied to other fully sequential procedures, such as the KN procedure of Kim and Nelson (2001), as well. However, the last modification depends critically on the structure of Paulson's procedure, and it is not clear how to extend it to other fully sequential procedures.

The rest of this paper is organized as follows. In Section 2, we first introduce the formulations of R&S problems and then briefly review the designs of Paulson's procedure. In Sections 3, 4, and 5, we propose three major modifications on the original Paulson's procedure to make it more suitable for parallel computing environments. In Section 6, we list the detailed description of our procedure and show that it is statistically valid after we make these modifications. In Section 7, we numerically compare our procedure with some existing parallel R&S procedures. We conclude in Section 8.

2. Paulson's Procedure

In this section, we briefly review one classical fully sequential procedure, Paulson's procedure in Paulson (1964). It is the building block of our parallel procedure.

2.1. Problem Formulation

We first introduce some standard notations for R&S problems. Let $\mathcal{K} = \{1, 2, \ldots, k\}$ be the set of alternatives in contention at the beginning of the selection process. We use

notation $X_{i,\ell}$ to indicate the ℓth output of simulation replication (observation) from alternative $i \in \mathcal{K}$. Following the convention in the R&S literature, we assume that for $i \in \mathcal{K}$, $\{X_{i,\ell} : \ell = 1, 2, \ldots\}$ are i.i.d. normal random variables with both mean μ_i and variance σ_i^2 unknown. We further assume that the observations generated by different alternatives are independent as well, i.e., common random numbers (CRNs) are not used in this paper. We use $\bar{X}_i(t)$ to denote the sample average of alternative i based on the first t observations taken by the alternative. Without loss of generality, we assume that the means are in ascending order, i.e., $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_k$, and alternative k is the best. Notice that without making an assumption on the configuration of the means, the means of the best and second-best alternatives can be arbitrarily close to each other, leading to a situation where an infinite number of observations is needed to identify the best. To overcome this issue, Paulson's procedure and many other frequentist procedures adopt the indifference-zone (IZ) formulation. It assumes that there exists a minimal gap between the means of the best and the second-best alternatives, i.e., $\mu_{k-1} \leq \mu_k - \delta$, where $\delta > 0$ is the user-specified IZ parameter and the smallest difference worth detecting. Under the IZ formulation, one's objective is to design a procedure that can satisfy the *probability of correct selection* (PCS) guarantee. It requires the procedure to select the best with probability no less than $1 - \alpha$ given that the IZ assumption holds, i.e.,

$$\mathbb{P}\left(\text{select alternative } k | \mu_k - \mu_{k-1} \geq \delta\right) \geq 1 - \alpha,$$

where α is a user-specified upper bound on the procedure's *probability of incorrect selection* (PICS).

2.2. The Procedure

For many traditional fully sequential procedures, during the selection process, pairwise comparisons are conducted between alternatives upon collecting new observations. After the first-stage sampling whose goal is to estimate the sample variances of all alternatives, the procedures construct a continuation region for every pair of competing alternatives. Every time each surviving alternative takes an additional observation, the procedures compare the partial sum difference between any two alternatives with their continuation region, i.e., all-pairwise comparisons. Once the partial sum difference process exits the predetermined continuation region, one alternative can be eliminated accordingly. The procedures

stop when only a single alternative is left and choose it as the best. Notice that not all fully-sequential procedures use such a selection structure. For instance, the Envelope Procedure of Ma and Henderson (2017) only compares two alternatives based on a certain criterion at each round and, therefore, avoiding the all-pairwise comparisons.

We provide the detailed description of Paulson's procedure below. It is worthwhile pointing out that the original Paulson's procedure was designed for the case where all alternatives share a common variance (either known or unknown). However, the procedure can be extended easily to the more general case of unknown and unequal variances. Therefore, one may observe some differences between the procedure provided in this paper and the original one.

Paulson's Procedure

Step 1. (Input): Select the desired PCS $1-\alpha$ $(0<\alpha<1-1/k)$. Set the IZ parameter $\delta>0$, first-stage sample size $n_0\geq 2$, and λ $(0<\lambda<\delta)$. Let

$$h^2=\frac{n_0-1}{4\left(\delta-\lambda\right)}\times\left[\left(\frac{\alpha}{k-1}\right)^{-\frac{2}{n_0-1}}-1\right].$$

Step 2. (Initialization): Let $\mathcal{I}=\{1,2,\ldots,k\}$ be the set of alternatives in contention. For each $i\in\mathcal{I}$, simulate n_0 observations $X_{i,1},X_{i,2},\ldots,X_{i,n_0}$ from alternative i. For all $i\neq j$, compute

$$S_{ij}^2=\frac{1}{n_0-1}\sum_{\ell=1}^{n_0}\left[X_{i,\ell}-X_{j,\ell}-\left(\bar{X}_i\left(n_0\right)-\bar{X}_j\left(n_0\right)\right)\right]^2,$$
$$N_{ij}=\left\lfloor\frac{h^2S_{ij}^2}{\lambda}\right\rfloor+1,$$

and let

$$N_i=\max_{j\neq i}N_{ij}.$$

Set $t=n_0$.

Step 3. (Screening): Set $\mathcal{I}_{\text{old}}=\mathcal{I}$. Let

$$\mathcal{I}=\left\{i:i\in\mathcal{I}_{\text{old}}\text{ and }Z_{ij}\left(t\right)\geq-h^2S_{ij}^2+\lambda t,\forall j\in\mathcal{I}_{\text{old}},j\neq i\right\},\tag{1}$$

where $Z_{ij}(t)=t\left[\bar{X}_i\left(t\right)-\bar{X}_j\left(t\right)\right]$.

Step 4. (Stopping Rule):

a. If $|\mathcal{I}| = 1$, where $|\mathcal{A}|$ denotes the cardinality of a set $\mathcal{A}$, stop and select the alternative whose index is in $\mathcal{I}$ as the best.

b. Else if $t = \max_{i\in\mathcal{I}} N_i$, stop and select the alternative whose index is in $\mathcal{I}$ having the largest $\bar{X}_i(t)$ as the best.

c. Otherwise, take one additional observation $X_{i,t+1}$ from each alternative $i \in \mathcal{I}$, set $t = t+1$, and go to **Screening**.

For the comparison between two alternatives i and j in Paulson's procedure, the continuation region is triangular in shape (as shown in Figure 1). Two lines $g(t) = -h^2S_{ij}^2 + \lambda t$ and $-g(t) = h^2S_{ij}^2 - \lambda t$ determine the decision boundaries of the continuation region and $\{Z_{ij}(t) = t[\bar{X}_i(t) - \bar{X}_j(t)] : t = 1, 2, \ldots\}$ is the partial sum difference process. The procedure conducts all-pairwise comparisons by using the screening process in (1). At $t = \max_{i\in\mathcal{K}} N_i$ (i.e., the maximum sample size), if there is still more than one surviving alternative in set $\mathcal{I}$, the procedure simply picks the alternative with the largest sample mean as the best. For Paulson's procedure, by changing the input parameter λ, one has the option to alter the slopes of the decision boundaries and thus the sizes of the continuation regions. Based on the *mean-path analysis* of Perng et al. (1969), setting a smaller value for λ, the procedure can eliminate clearly inferior alternatives faster. However, the ending point of the continuation becomes larger and the procedure needs more observations to identify the best alternative in the worst case scenario. Typically, we let $\lambda = \delta/2$.

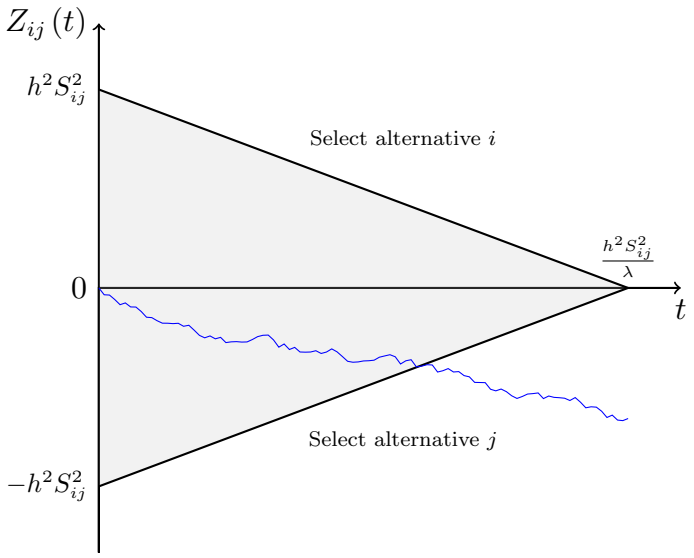


Figure 1 **Continuation region for Paulson's procedure.**

Besides Paulson's procedure, the KN procedure (Kim and Nelson 2001) is another well-known fully sequential procedure. It also has a triangular continuation region and it is often tighter than that of Paulson's procedure. Therefore, the KN procedure is often more efficient in terms of the total sample size than Paulson's procedure. However, to gain efficiency, the KN procedure is more rigid in its structure and it is difficult to modify. As we shall see, in order to improve the efficiency of a fully sequential procedure in parallel computing environments, we require the procedure to take a batch of observations from surviving alternatives when the number of surviving alternatives is smaller than that of the processors. The statistical validity of the KN procedure may fail in this situation. Therefore, in this paper, we focus on modifying Paulson's procedure.

3. Speeding Up Pairwise Comparisons

Similar to the works of Luo et al. (2015) and Ni et al. (2017), in this paper, we adopt the Master/Worker structure for parallel computing. A designated processor is served as the master. The master controls the whole simulation program, including managing the information of all alternatives, assigning simulation tasks to and retrieving the simulation outcomes from the workers, performing all-pairwise comparisons, and other necessary calculations. All other processors are the workers which work in a simple cycle of taking the simulation task from the master, simulating the observation, and submitting the results back to the master.

In this section, we first take Paulson's procedure as an example to show how all-pairwise comparisons can affect the performance of fully sequential procedures in parallel computing environments. We then propose a way to address this issue.

3.1. High Computational Complexity of Comparisons

From the descriptions of Paulson's procedure, it can be seen that, after taking initial n_0 observations for every alternative to estimate the variances (**Step 2**), the procedure iteratively conducts all-pairwise comparisons among the alternatives (**Step 3**) and simulates new observations for surviving alternatives (**Step 4.c**) until it finally decides the best. Typically, simulating an observation takes significantly more time than that of making a comparison. When the problem size k is not very large and there are not many comparisons required at each round, the time spent on simulating observations largely determines how fast a procedure can solve the problem. Therefore, as the first attempt to implement fully

sequential procedures in parallel computing environments, Luo et al. (2015) essentially consider to equally distribute the simulation tasks at each round to all workers so that the overall simulation time can be reduced. By doing so, the authors successfully solve a R&S problem with more than 20,000 alternatives.

However, for such a direct implementation, as the number of alternatives keeps increasing, the marginal effect of adding more workers to the parallel computing environment diminishes. The reasons are as follows. At each round, the computational complexity of simulating observations in **Step 4.c** is $\mathcal{O}(k)$. Meanwhile, the comparisons in **Step 3** require a computational complexity $\mathcal{O}(k^2)$ since each alternative needs to compare with all other surviving alternatives. As the number of alternatives k increases, the computational complexity of comparisons increases much faster than that of simulations. Moreover, because the simulation tasks can be equally distributed to all workers, the overall simulation time actually grows at the rate of $\mathcal{O}(k/m)$ at each round where m is the number of available workers. As a contrast, due to the nature of all-pairwise comparisons, whenever the procedure checks the elimination condition for one alternative, the sampling information on all other alternatives is needed. These comparisons are not parallelizable and can only be done on the master. Therefore, the growth rate of the overall comparison time remains to be $\mathcal{O}(k^2)$. As k increases, the comparison time quickly becomes the bottleneck of the procedure to solve problems and cannot be shortened by adding more workers in the parallel computing environment. One may argue that, in practice, as more and more alternatives are eliminated, the comparison time decreases dramatically at each round. However, due to the high computational complexity, even if there is only one round of all-pairwise comparisons among all k alternatives, as we shall see in the numerical experiments, such comparison time is no longer negligible.

In the rest of this section, we propose our way to address this issue. We show that if no CRNs are used, we can reduce the computational complexity of comparisons in Paulson's procedure to $\mathcal{O}(k)$ at each round, the same order as that of simulations, and this technique can be potentially extended to other fully sequential procedures. It is interesting to point out that the use of CRNs in R&S procedures is often for the purpose of introducing positive correlations among the observations from different alternatives and reducing the total sample size. To use CRNs, one needs to compute and store the sample variance of the partial sum difference between any two alternatives and to ensure that the sample

sizes of any two alternatives in comparisons are the same. This imposes challenges to both computation and memory management. Therefore, for both aforementioned works, i.e., Luo et al. (2015) and Ni et al. (2017), the proposed procedures do not support the use of CRNs.

3.2. Main Idea

To illustrate our main idea, we first consider a special case where all alternatives share a common known variance σ_*^2. Then, for Paulson's procedure, the screening process in (1) changes to,

$$\mathcal{I} = \left\{ i : i \in \mathcal{I}_{\text{old}} \text{ and } Z_{ij}(t) \geq -2h'^2\sigma_*^2 + \lambda t, \forall j \in \mathcal{I}_{\text{old}}, j \neq i \right\}, \tag{2}$$

where h' is a constant determined by λ, δ, k, and α, and the first-stage sample variance S_{ij}^2 is replaced by $2\sigma_*^2$ because $\text{Var}(X_{i,\ell} - X_{j,\ell})$ is known to be $2\sigma_*^2$ in this situation. Since the right hand side of the inequality in (2) is invariant with the change of the alternatives, the screening process (2) is equivalent to letting

$$\mathcal{I} = \left\{ i : i \in \mathcal{I}_{\text{old}} \text{ and } Z_{ii^*}(t) \geq -2h'^2\sigma_*^2 + \lambda t, \right\}, \tag{3}$$

where $i^* = \arg\max_{i \in \mathcal{I}_{\text{old}}} \bar{X}_i(t)$. It suggests that for the case of a common known variance, at each round, one does not need to conduct all-pairwise comparisons among the surviving alternatives; instead one only needs to compare every alternative with alternative i^*, the alternative with the largest sample mean. For the screening process (3), finding alternative i^*, requires $\mathcal{O}(k)$ operations, and comparing each alternative in set $\mathcal{I}_{\text{old}}$ with alternative i^* requires another $\mathcal{O}(k)$ operations. Therefore, at each round, the computational complexity of the screening process (3) reduces to $\mathcal{O}(k)$. Notice that, these modifications do not affect the procedure's efficiency on eliminating inferior alternatives. Thus, the total sample size of the procedure remains unchanged.

This motivates us to think whether there exists an alternative for the case of unknown and unequal variances that works in the same way as alternative i^* does for the common known variance case. To extend this idea, we notice that if no CRNs are used, one can replace the sample variance S_{ij}^2 by $S_i^2 + S_j^2$ where $S_i^2 = \frac{1}{n_0 - 1}\sum_{\ell=1}^{n_0}(X_{i,\ell} - \bar{X}_i(n_0))^2$ is the first-stage sample variance of alternative i. Then the screening process (1) can be rewritten as,

$$\mathcal{I} = \left\{ i : i \in \mathcal{I}_{\text{old}} \text{ and } t\bar{X}_i(t) + h^2 S_i^2 \geq t\bar{X}_j(t) - h^2 S_j^2 + \lambda t, \forall j \in \mathcal{I}_{\text{old}}, j \neq i \right\}. \tag{4}$$

For notational ease, we define,

$$W_i^+(t) = t\bar{X}_i(t) + h^2 S_i^2 \text{ and } W_i^-(t) = t\bar{X}_i(t) - h^2 S_i^2,$$

for $i = 1, 2, \ldots, k$. Notice that for any fixed alternative i,

$$\left\{W_i^+(t) \geq W_j^-(t) + \lambda t, \forall j \in \mathcal{I}_{\text{old}}, j \neq i\right\} \Leftrightarrow \left\{W_i^+(t) \geq \max_{j \in \mathcal{I}_{\text{old}} \setminus \{i\}} W_j^-(t) + \lambda t\right\}. \quad (5)$$

Screening processes (4) and (5) suggest that for the case of unknown and unequal variances, instead of picking the alternative with the largest sample mean, we can define a "best" alternative $i^*_- = \arg\max_{j \in \mathcal{I}_{\text{old}}} W_j^-(t)$ based on both sample mean and sample variance information. Then, all-pairwise comparisons at each round are equivalent to letting every alternative (expect for alternative i^*_-) only compare with the "best" alternative i^*_-. Following the same argument of the common known variance case, one can conclude that the computational complexity of comparisons also reduces to $\mathcal{O}(k)$ under this setting. In the conference paper of Hong et al. (2016), which is a preliminary version of this paper, they demonstrate that this technique applies to the KN procedure as well. Notice that, technically, $S_i^2 + S_j^2$ and S_{ij}^2 have different distributions. In Section 6, we prove that this modification does not change the statistical guarantee of Paulson's procedure. However, after the modification, the procedure's total sample size will be slightly different from that of the original one.

4. Reducing the Frequency of Communications

Different from the single-processor computing, parallel computing additionally requires the processors to spend time communicating with each other. Frequent communications among processors may also delay the selection process.

4.1. One-by-one Task Assignment Scheme

In the existing literature, to simulate observations at each round, a practical way is to let the master assign the tasks of simulating an observation one-by-one to the next available worker, i.e., in a round-robin manner. One advantage of this type of task assignment is that it can finely balance the workloads of different processors. However, frequent communications are needed in this situation. Typically, to simulate an observation, two rounds of communications are needed between the master and a worker. The master first assigns the simulation task to the worker. After simulating the observation, the worker then reports

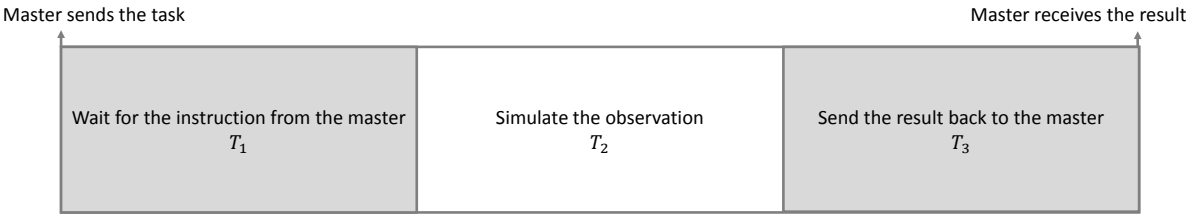


Figure 2 The activities inside a worker regarding the simulation of one observation.

the simulation results back to the master. Therefore, in total, $2k$ rounds of communications are needed to generate an additional observation from k alternatives at each round.

Figure 2 shows the activities inside a worker regarding the simulation of one observation. It can be seen that the worker stays idle when it waits for the instruction from the master, denoted by T_1, and the time after it sends the result back to the master, denoted by T_3. Therefore, when the transmission time of sending a message from one processor to another is relatively long, which is especially true for the case where the master and the worker are in two different machines and the network latency exists, this task assignment scheme can lead to high communication cost relative to the total amount computation cost for simulations. In this situation, assigning the simulation tasks one-by-one in a round-robin manner may not be an optimal choice. In the following subsection, we propose a batching-of-alternatives method to reduce the communication frequency between the master and workers, and thus the communication cost.

4.2. Batching of Alternatives

For parallel computing, an appealing way to reduce the communication cost is batching, i.e., sending a batch of simulation tasks to a worker at a time. Typically, there are two ways to batch the simulation tasks in R&S procedures. One can let a worker either simulate a batch of observations for one alternative at a time (as in Ni et al. (2017)) or simulate observations for a batch of alternatives at a time. Notice that the first way of batching can make the procedure behave more like a stage-wise procedure and may potentially cause a relatively large amount of efficiency loss on the total sample size. In this paper, we choose the second way of batching so that the fully sequential selection structure of the procedure can still be largely maintained. Specifically, at each round, we let the master equally divide the surviving alternatives into m groups and instruct each worker to perform the simulations for one group of alternatives (i.e., simulate an additional observation for every alternative in the group). A worker submits the simulation results back to the master after finishing

the task. Therefore, at each round, every worker only needs to receive the instruction from the master and report the results back to the master once. In this situation, the time that a worker spends on communications is significantly reduced. Moreover, unlike the first way of batching, batching of alternatives keeps the selection structure of the fully sequential procedure and does not sacrifice the sample-size efficiency.

Notice that we are not arguing that our batching method is better than the one-by-one task assignment scheme. Compared with the latter one, there is a drawback of the batching method. Due to the random completion time of simulating an observation, as the master sends out all the simulation tasks at once, the workers which finish the tasks earlier always need to wait for the last one to complete the tasks. It introduces additional idle time on the workers. In the following subsection, we use a mathematical model to compare the wall-clock times of simulating k observations as both task assignment schemes are used. We show that our batching method may be better than the one-by-one task assignment scheme when transmitting a message takes relatively long time and the number of alternatives is large.

4.3. One-by-one v.s. Batching

Consider the tasks of simulating k observations. There are one master processor and m identical workers in the parallel computing environment. In this model, we assume that the transmission time of a message sent from one processor to another is fixed[1] and denoted by t_c. Let Φ_ℓ, for $\ell = 1, 2, \ldots, k$, denote the actual time that a worker requires to perform the simulation of observation ℓ, i.e., T_2 in Figure 2. We assume that,

$$\Phi_\ell = \theta + \varepsilon_\ell,$$

where θ is the expected time of simulating an observation, and $\{\varepsilon_\ell\}$ are i.i.d. σ-subgaussian noise terms (see Definition 4.1 below).

Definition 4.1 *A random variable Z is σ-subgaussian if $\mathbb{E}[Z] = 0$, and $\mathbb{E}[e^{\tau Z}] \leq e^{\tau^2\sigma^2/2}$ holds for all $\tau > 0$.*

[1] In practice, the transmission time can be additionally affected by the size of the message. Since in our case, the size of a message is typically very small, we treat the transmission times of the messages with different sizes as the same in this paper.

REMARK 1. Subgaussian distributions include many commonly used distributions. For example, the normal distributions and the distributions with bounded supports—uniform distributions, Bernoulli distributions, etc. One feature of the subgaussian distribution is that the sum of n i.i.d. σ-subgaussian random variables, $Z_1, Z_2, \ldots, Z_n$, is a $\sqrt{n}\sigma$-subgaussian random variable. This can be simply shown by $\mathbb{E}\left[e^{\tau(Z_1+Z_2+\cdots+Z_n)}\right] \leq e^{n\tau^2\sigma^2/2}$.

We further let $\mathcal{W}_1$ and $\mathcal{W}_2$ denote the wall-clock times of simulating these k observations as the one-by-one task assignment scheme and our batching method are used respectively. In the following analysis, we compare the performance of these two task assignment schemes using the expected values of $\mathcal{W}_1$ and $\mathcal{W}_2$. In the analysis, without loss of generality, we assume that k/m is an integer.

For the one-by-one task assignment scheme, to simulate an observation, the master first sends the task to a worker and after simulating the observation, the worker reports the results back to the master. While estimating $\mathcal{W}_1$, we assume that upon receiving the simulation results from a worker, the master can immediately send the next task to the worker. In this situation, the overall time the worker spends on simulating an observation is $2t_c + \Phi_\ell$, where $2t_c$ represents the idle time that the worker spends on waiting for the instruction from and reporting the results back to the master. Therefore, to simulate all k observations, the total time required across all workers is $2kt_c + \sum_{\ell=1}^{k} \Phi_\ell$. Ideally, under the one-by-one task assignment scheme, this total time can be equally allocated to all m workers. Thus, we can approximate $\mathcal{W}_1$ by,

$$\mathcal{W}_1 = \frac{2kt_c}{m} + \frac{\sum_{\ell=1}^{k} \Phi_\ell}{m}.$$

For the batching method, the master first equally divides the tasks of simulating k observations into m groups and assigns each worker one group of observations. During the implementation, we find that these operations take almost no time. While approximating $\mathcal{W}_2$, we omit the time that the master divides the tasks and assume that the master can send out all the tasks simultaneously. After time t_c, all workers receive the tasks. Every worker then independently simulates k/m observations as assigned. Let Γ_s, for $s = 1, 2, \ldots, m$, denote the actual time that worker s requires to finish simulating k/m observations. For example, if worker 1 handles the simulations for the first k/m observations, then $\Gamma_1 = \sum_{\ell=1}^{k/m} \Phi_\ell$. Then, we can write Γ_s as,

$$\Gamma_s = \frac{k}{m}\theta + \Lambda_s,$$

where Λ_s is the sum of k/m i.i.d. σ-subgaussian random variables and is thus a $\sqrt{k/m}\sigma$-subgaussian noise term. The simulation process stops after the last worker finishes its tasks and spends another t_c amount of time on transmitting the results to the master. Therefore, we can approximate the wall-clock time $\mathcal{W}_2$ by,

$$\mathcal{W}_2 = \max_{s=1,2,\ldots,m} \Gamma_s + 2t_c.$$

In the theorem below, we present the relationship between $\mathbb{E}[\mathcal{W}_1]$ and $\mathbb{E}[\mathcal{W}_2]$ when k is large.

THEOREM 1. *If* $\mathcal{W}_1 = \frac{2kt_c}{m} + \frac{\sum_{\ell=1}^{k}\Phi_\ell}{m}$ *and* $\mathcal{W}_2 = \max_{s=1,2,\ldots,m}\Gamma_s + 2t_c$, *then*

$$\limsup_{k\to\infty} \frac{\mathbb{E}\left[\mathcal{W}_2\right]}{\mathbb{E}\left[\mathcal{W}_1\right]} \leq \frac{\theta}{2t_c+\theta}.$$

Proof. To prove Theorem 1, we first find an upper bound for $\mathbb{E}[\mathcal{W}_2]$. Notice that for $\mathbb{E}[\mathcal{W}_2]$, we have,

$$\begin{aligned}\mathbb{E}\left[\mathcal{W}_2\right] &= \mathbb{E}\left[\max_{s=1,2,\ldots,m}\Gamma_s\right] + 2t_c \\ &= \frac{k}{m}\theta + \mathbb{E}\left[\max_{s=1,2,\ldots,m}\Lambda_s\right] + 2t_c. \end{aligned} \tag{6}$$

We focus on analyzing $\mathbb{E}[\max_{s=1,2,\ldots,m}\Lambda_s]$. For notational simplicity, we let $\mathcal{M}_m = \max_{s=1,2,\ldots,m}\Lambda_s$. Then, for any $\tau > 0$, by Jensen's inequality, we have the following result,

$$\begin{aligned} e^{\tau\mathbb{E}[\mathcal{M}_m]} &\leq \mathbb{E}\left[e^{\tau\mathcal{M}_m}\right] \\ &= \mathbb{E}\left[\max_{s=1,2,\ldots,m} e^{\tau\Lambda_s}\right] \\ &\leq \sum_{s=1}^{m}\mathbb{E}\left[e^{\tau\Lambda_s}\right] \end{aligned} \tag{7}$$

Notice that Λ_s is a $\sqrt{k/m}\sigma$-subgaussian random variable. By the definition of the subgaussian random variable, we have $\mathbb{E}\left[e^{\tau\Lambda_s}\right] \leq e^{k\tau^2\sigma^2/(2m)}$. Plugging this result into Equation (7), we can deduce that,

$$\mathbb{E}\left[\mathcal{M}_m\right] \leq \frac{\log m}{\tau} + \frac{k\tau\sigma^2}{2m}. \tag{8}$$

Letting $\tau = \sqrt{\frac{2m\log m}{k\sigma^2}}$, Equation (8) can be rewritten as,

$$\mathbb{E}\left[\mathcal{M}_m\right] \leq \sigma\sqrt{\frac{2k\log m}{m}}. \tag{9}$$

Plugging (9) into (6), then it yields,

$$\mathbb{E}\left[\mathcal{W}_2\right] = \frac{k}{m}\theta + \mathbb{E}\left[\max_{s=1,2,\ldots,m} \Lambda_s\right] + 2t_c \leq \frac{k}{m}\theta + \sigma\sqrt{\frac{2k\log m}{m}} + 2t_c. \tag{10}$$

Notice that for $\mathbb{E}[\mathcal{W}_1]$, because $\{\Phi_\ell, \ell = 1, 2, \ldots, k\}$ are i.i.d. random variables with mean θ, one can easily compute that

$$\mathbb{E}\left[\mathcal{W}_1\right] = \frac{k}{m}\left(2t_c + \theta\right). \tag{11}$$

With the results in equations (10) and (11), we have,

$$\limsup_{k\to\infty} \frac{\mathbb{E}\left[\mathcal{W}_2\right]}{\mathbb{E}\left[\mathcal{W}_1\right]} \leq \lim_{k\to\infty} \frac{\frac{k}{m}\theta + \sigma\sqrt{\frac{2k\log m}{m}} + 2t_c}{\frac{k}{m}\left(2t_c + \theta\right)} = \frac{\theta}{2t_c + \theta}. \tag{12}$$

It concludes the proof. □

Theorem 1 suggests that, when the number of workers in the parallel computing environments is fixed, as long as the transmission takes time and the number of alternatives is large enough, our batching of alternatives method tends to require smaller wall-clock time to simulate observations than the one-by-one task assignment scheme does.

However, there are some limitations in the analysis of Theorem 1. First, in Theorem 1, we consider a situation where the number of processors is fixed. In practice, as the number of alternatives increases, more and more processors may be added to the parallel computing environments. Then, based on the results of Theorem 1, it is unclear which task assignment scheme is better. To determine which task assignment scheme should be used in this situation, while analyzing the wall-clock time of the one-by-one task assignment scheme in Theorem 1, we assume that master can immediately respond to the message received from a worker. As pointed out by Luo et al. (2015) and Ni et al. (2017), for the one-by-one task assignment scheme, when there are a large number of workers in the parallel computing environments, the master could potentially be overwhelmed with the frequent incoming messages and take a very long time to send a message back. It may significantly slow down the selection process. Therefore, we may still choose the batching method in this situation. Second, according to Equation (12), for the case where $t_c \ll \theta$, if the number of workers in the parallel computing environments is relatively small (i.e., the master would not get overwhelmed for the one-by-one task assignment scheme), the batching method may never show its theoretically superior performance until k is much

larger than the problem size that we can encounter in practice. There is no rule of thumb to determine which method is better in this situation. To practically address this issue, before the selection starts, one may start an initial stage to estimate the wall-clock time needed to generate one round of observations for each task assignment scheme and choose whichever has a smaller wall-clock time. In this paper, because we implement our procedure in a parallel computing environment with up to 96 processors and the processors can be spread across different local servers (t_c is relatively large), we choose to use the batching method to allocate observations at each round. Notice that besides Paulson's procedure, the batching method may be applied to other fully sequential procedures, e.g., the KN procedure, as well.

We close this section with a remark that as a by-product of our batching method, we can further reduce the overall comparison time. After a worker simulates the observations for a group of alternatives, we can additionally request the worker to find the alternative with the largest $W_j^-(t)$ within the group and report this local "best" alternative together with the simulation results to the master. Then, the master only needs to compare all m local "best" alternatives to decide the unique "best" alternative i^*_- and afterward return each worker alternative i^*_- to conduct eliminations. Notice that finding the unique "best" alternative requires $\mathcal{O}(k/m)$ operations in each worker and $\mathcal{O}(m)$ operations in the master, and comparing a local set of alternatives with the unique "best" alternative needs another $\mathcal{O}(k/m)$ operations in a worker. Therefore, the overall comparison time at each round reduces from $\mathcal{O}(k)$ to $\mathcal{O}(k/m+m)$ in this situation.

5. Boosting the Selection at the Final Stage

As the selection process continues, fewer and fewer alternatives are left. All these surviving alternatives tend to be "good" alternatives with similar means. In this situation, with a small increment on the number of observations, it becomes increasingly difficult to eliminate alternatives. Then, conducting comparisons every time that each surviving alternative takes an additional observation becomes less efficient. Moreover, when the number of surviving alternatives is less than the number of workers in the parallel computing environment, maintaining the fully sequential selection structure can lead to processor idling because each worker on average handles the simulation for less than one alternative at each round. To avoid this issue, in this paper, when only m alternatives are left, we suggest sampling every surviving alternative to the maximal number of observations it can

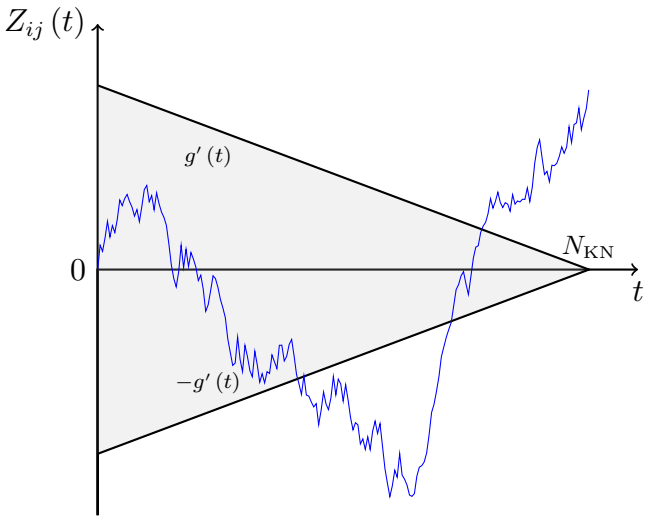


Figure 3 Continuation region of the KN procedure.

take, i.e., $\max_{i\in\mathcal{K}} N_i$, and each worker handles the simulations for one alternative. After the simulations, we simply pick the alternative with the largest sample mean as the best.

Though this modification is very intuitive, it changes the selection structure of a fully sequential procedure, and thus may affect the statistical validity of the procedure. For example, for the comparison between two alternatives in the KN procedure, the procedure allows the user to take multiple observations for each alternative at a time. However, the number of observations can only depend on their own partial sum difference process through its values in previous periods (see Lemma 2 of Hong (2006) and Jennison et al. (1980)). As in our case, the start of the final-stage depends on how many alternatives are left (i.e., all other partial sum difference processes). *The statistical validity of the KN procedure may fail in this situation.*

To better illustrate our point, consider the comparison between two alternatives i and j where $\mu_i - \mu_j = -\delta$. In Figure 3, we plot a sample path of the partial sum difference process $\{Z_{ij}(t) : t = 1, 2, \ldots\}$ between the two alternatives. Notice that the grey area formed by $g'(t)$ and $-g'(t)$ is the continuation region of the KN procedure given that the IZ parameter is δ and the PICS is α. The continuation region ends at N_{KN}. Let Υ denote the continuation region and t_s, for $s = 1, 2, \ldots$, denote the sth time point where we observe the value of partial sum difference and compare it with the continuation region. We further let $\mathcal{T} = \{t_s : s = 1, 2, \ldots \text{ and } t_s \leq \lceil N_{\mathrm{KN}} \rceil\}$. The difference $t_s - t_{s-1}$ represents the number of observations we decide to take for each alternative at time point t_{s-1}. The continuation

region of the KN procedure ensures that if t_s only depends on $Z_{ij}(t)$ through its values in period $[0, t_{s-1}]$ and $\mathbf{T} = \inf\{t \in \mathcal{T} : Z_{ij}(t) \notin \Upsilon\}$, then

$$\mathbb{P}\left(Z_{ij}\left(\mathbf{T}\right) \geq 0\right) \leq \frac{\alpha}{k-1}.$$

Notice that the condition on t_s is crucial here. If t_s depends on other random processes, the inequality above may no longer hold. For example, if there exists another random process $Z'_{ij}(t)$ such that $Z'_{ij}(t)$ is $\lceil N_{\mathrm{KN}} \rceil$ time steps ahead of $Z_{ij}(t)$, i.e., $Z'_{ij}(t) = Z_{ij}\left(t + \lceil N_{\mathrm{KN}} \rceil\right)$, then at $t = 0$ by referring to the process $Z'_{ij}(t)$ in the period $t \in [-\lceil N_{\mathrm{KN}} \rceil, 0]$, we can set

$$\mathcal{T}'_1 = \left\{t : t = 1, 2, \ldots,\ Z'_{ij}(t - \lceil N_{\mathrm{KN}} \rceil) \geq g'(t),\ \text{and } t < \lceil N_{\mathrm{KN}} \rceil\right\},$$

and $\mathcal{T}' = \mathcal{T}'_1 \cup \{\lceil N_{\mathrm{KN}} \rceil\}$. In this situation, if $\mathbf{T}' = \inf\{t \in \mathcal{T}' : Z_{ij}(t) \notin \Upsilon\}$, then based on the definition of $\mathcal{T}'$, one can easily show that

$$\begin{aligned}\mathbb{P}\left(Z_{ij}\left(\mathbf{T}'\right) \geq 0\right) &= \mathbb{P}\left(\left\{\sup_{t=1,2,\ldots,\lceil N_{\mathrm{KN}} \rceil - 1} Z_{ij}\left(t\right) \geq g'(t)\right\} \bigcup \left\{Z_{ij}\left(\lceil N_{\mathrm{KN}} \rceil\right) \geq 0\right\}\right) \\ &\geq \mathbb{P}\left(Z_{ij}\left(\mathbf{T}\right) \geq 0\right).\end{aligned} \tag{13}$$

The inequality holds because by definitions, $\mathbb{P}\left(Z_{ij}\left(\mathbf{T}\right) \geq 0\right)$ is upper bounded by the probability that the random process $\{Z_{ij}(t) : 1 \leq t \leq \lceil N_{\mathrm{KN}} \rceil\}$ ever exits the continuation region from the upper decision boundary $g'(t)$, i.e., the probability of the first event on the right-hand side of (13). Therefore, it is unclear whether $\mathbb{P}\left(Z_{ij}\left(\mathbf{T}'\right) \geq 0\right)$ is still upper bounded by $\alpha/(k-1)$. The continuation region of the KN procedure can no longer provide any statistical guarantee for the partial sum difference process $\{Z_{ij}(t) : t \in \mathcal{T}'\}$ in this situation. As we will later show, the statistical validity of Paulson's procedure will not be affected after we make this modification.

6. New Procedure and Statistical Analysis

In this section, we first present our parallel fully sequential procedure, called the parallelized Paulson's Procedure (PPP), followed by a validation of its statistical guarantee.

6.1. The Procedure

In what follows, we provide detailed description of our procedure. In the descriptions, we let **(M)** denote the jobs that are completed by the master and **(W)** denote the jobs that are completed by workers. Notice that when the master and the workers exchange information

about an alternative, it contains the first-stage sample variance, the most updated sample average, and the sample size of the alternative. Also, for every worker, after an iteration, it will free up its memory and no information will be stored.

The Parallelized Paulson's Procedure (PPP)

Step 1. (Input): Select the desired PCS $1-\alpha$ $(0<\alpha<1-1/k)$. Set the IZ parameter $\delta>0$, first-stage sample size $n_0\geq 2$, λ $(0<\lambda<\delta)$, and number of workers $m\geq 1$. Let

$$h^2=\frac{n_0-1}{4\left(\delta-\lambda\right)}\times\left[\left(\frac{\alpha}{k-1}\right)^{-\frac{2}{n_0-1}}-1\right].$$

Step 2. (Initialization): Let $\mathcal{I}=\{1,2,\ldots,k\}$ be the set of alternatives in contention.
(M) Let $\mathcal{I}^s$ be the set of alternatives assigned to worker s for $s=1,2,\ldots,m$. Equally allocate all k alternatives to m workers so that each worker handles the first-stage simulations for approximately k/m alternatives, e.g., an allocation rule could be:

for $i=1,2,\ldots,k$, do

$$\mathcal{I}^{(i \bmod m)+1}=\mathcal{I}^{(i \bmod m)+1}\bigcup\{i\}.$$

(W) For worker $s=1,2,\ldots,m$, simulate n_0 observations for each alternative $i\in\mathcal{I}^s$ and compute

$$S_i^2=\frac{1}{n_0-1}\sum_{\ell=1}^{n_0}\left[X_{i,\ell}-\bar{X}_i\left(n_0\right)\right]^2,$$
$$i^*_{n_0,s}=\underset{i\in\mathcal{I}^s}{\arg\max}\,W_i^-\left(n_0\right),$$

where $i^*_{t,s}$ denote the local "best" alternative at time t in worker s. Then, report the results back to the master.
(M) Upon receiving the simulation results from all workers, set $t=n_0$.
Step 3. (Screening):
(M) Let

$$i^*_-=\underset{i\in\left\{i^*_{t,1},\ldots,i^*_{t,m}\right\}}{\arg\max}\,W_i^-\left(t\right),$$

and return i^*_- to all workers.
(W) For worker $s=1,2,\ldots,m$, set $\mathcal{I}^s_{\text{old}}=\mathcal{I}^s$, let

$$\mathcal{I}^s=\left\{i:i\in\mathcal{I}^s_{\text{old}}\text{ and }W_i^+\left(t\right)\geq W_{i^*_-}^-\left(t\right)+\lambda t\right\},\tag{14}$$

and report $\mathcal{I}^s$ back to the master.

(M) Upon receiving the comparison results, let

$$\mathcal{I} = \bigcup_{s=1,2,\ldots m} \mathcal{I}^s.$$

If $i^*_- \notin \mathcal{I}$, let $\mathcal{I} = \mathcal{I} \cup \{i^*_-\}$. Compute

$$N_{\max} = \max_{j,i\in\mathcal{I},j\neq i} \left\lfloor \frac{h^2\left(S_i^2 + S_j^2\right)}{\lambda} \right\rfloor + 1.$$

Step 4. (Stopping Rule):

a. If $|\mathcal{I}| \leq m$,

i. If $|\mathcal{I}| > 1$,

(W) For worker $s = 1, 2, \ldots, |\mathcal{I}|$, simulate one alternative in $\mathcal{I}$ to the largest sample size $N_{\max}$.

(M) Retrieve the simulation results from all workers and pick the alternative with the largest $\bar{X}_i\left(N_{\max}\right)$ in $\mathcal{I}$ as the best.

ii. If $|\mathcal{I}| = 1$,

(M) Select the alternative in $\mathcal{I}$ as the best.

b. Else if $t = N_{\max}$,

(M) Stop and select the alternative whose index is in $\mathcal{I}$ having the largest $\bar{X}_i\left(t\right)$ as the best.

c. Otherwise,

(M) Equally assign the alternatives in $\mathcal{I}$ to sets $\mathcal{I}^s$ for $s = 1, 2, \ldots, m$. Set $t = t + 1$.

(W) For each worker $s = 1, 2, \ldots, m$, take one additional observation $X_{i,t}$ from each alternative $i \in \mathcal{I}^s$, find the $i^*_{t,s}$, and report the results back to the master.

Then, go to **Screening**.

Notice that when the communication cost is relatively high and the number of surviving alternatives is not very large, in **Step 3**, one may conduct the screening process solely on the master rather than distribute the screening tasks to the workers. Also, in **Step 4.a.i**, the procedure simulates every surviving alternative to the maximal number. In practice, it can be the case that after the last screening process, there are fewer than m alternatives rather than exactly m alternatives left in **Step 4.a.i**. For this case, we only use $|\mathcal{I}|$ workers to simulate observations for the leftover alternatives. This can cause processor idling.

However, as we previously mentioned, when only a few alternatives are left, the number of surviving alternatives decreases very slowly. It suggests that once $|\mathcal{I}|$ falls below m, the difference between m and $|\mathcal{I}|$ is not large. So the number of idle processors is not large in this situation. In PPP, because we do not assume any prior knowledge on the expected simulation times of an observation for different alternatives, we choose to randomly assign alternatives to workers in **Step 4.c**. In practice, the expected time needed to simulate an observation may vary from one alternative to another and the user may have some foreknowledge on it, e.g., estimating it at the beginning of the selection process. In this situation, one can incorporate this piece of information in the allocation rule in **Step 4.c** to achieve a better load balancing among workers.

6.2. Statistical Validity

To prove that PPP is statistically valid, we first summarize the hitting probability for a partial sum process as follows.

LEMMA 1 **(Paulson (1964))**. *Suppose* $\{Y_1, Y_2, \ldots\}$ *is a sequence of i.i.d. normal random variables with mean* $\Delta < 0$ *and variance* σ_Y^2*. For a positive constant* $a > 0$*, we have,*

$$\mathbb{P}\left(\sum_{i=1}^{n} Y_i > a \text{ for some } n < \infty\right) \leq e^{2\Delta a/\sigma_Y^2}.$$

With the results in Lemma 1, we can proceed our analysis on the statistical validity of PPP. In Theorem 2, we first show that PPP can satisfy the PCS guarantee.

THEOREM 2. *If the means of the alternatives are in ascending order, there exists an optimality gap* $\delta > 0$*, between the means of the best alternative and all other alternatives, i.e.,* $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_k - \delta$*., and the observations generated by different alternatives are independent, then PPP can select alternative* k *with probability at least* $1 - \alpha$*.*

Proof. To show the results, we first decompose the problem of selecting the unique best alternative k into $k-1$ subproblems in which alternative k competes with all other alternatives.

$$\begin{aligned}
\mathbb{P}\left(\text{select alternative (alt.) } k\right) &= \mathbb{P}\left(\text{alt. } 1, 2, \ldots, k-1 \text{ are eliminated}\right) \\
&\geq \mathbb{P}\left(\bigcap_{i=1}^{k-1} \{\text{alt. } k \text{ eliminates alt. } i\}\right) \\
&\geq 1 - \sum_{i=1}^{k-1} \mathbb{P}\left(\text{alt. } i \text{ eliminates alt. } k\right). \qquad (15)
\end{aligned}$$

Then, we focus on analyzing the probability that alternative k is falsely eliminated by each alternative $i = 1, 2, \ldots, k-1$. For the comparison between alternatives i and k, let $\mathbf{N} < N_{\max}$ denote the random time point where the procedure switches to simulate a batch of observations for each surviving alternative to the maximal number, i.e., the time point where **Step 4.a** is initiated. Notice that for $n_0 \le t \le \mathbf{N}$, the procedure compares the partial sum difference between alternatives i and k with the continuation region to make elimination decisions. Once $t > \mathbf{N}$, the procedure directly samples the alternatives to the maximal number of observations $N_{\max}$ and compares the sample means to make elimination decisions. Thus, we can write down the false elimination probability between alternatives i and k as follows,

$$\begin{aligned}
&\mathbb{P}\left(\text{alt. } i \text{ eliminates alt. } k\right)\\
&\le \mathbb{P}\left(\left\{\max_{n_0\le t\le \mathbf{N}}\left[Z_{ik}\left(t\right)+\lambda t\right]\ge h^2\left(S_i^2+S_k^2\right)\right\}\bigcup\left\{Z_{ik}\left(N_{\max}\right)\ge 0\right\}\right)\\
&\le \mathbb{P}\left(\left\{\max_{n_0\le t< N_{\max}}\left[Z_{ik}\left(t\right)+\lambda t\right]\ge h^2\left(S_i^2+S_k^2\right)\right\}\bigcup\left\{Z_{ik}\left(N_{\max}\right)\ge 0\right\}\right)\\
&\le \mathbb{P}\left(\max_{n_0\le t\le N_{\max}}\left[Z_{ik}\left(t\right)+\lambda t\right]\ge h^2\left(S_i^2+S_k^2\right)\right)\\
&= \mathbb{P}\left(\max_{n_0\le t\le N_{\max}}\sum_{\ell=1}^{t}\left(X_{i,\ell}-X_{k,\ell}+\lambda\right)\ge h^2\left(S_i^2+S_k^2\right)\right), \qquad (16)
\end{aligned}$$

where the second inequality holds because $\mathbf{N} < N_{\max}$, and the third inequality holds because by the definition of $N_{\max}$, $\bar{X}_i(N_{\max}) \ge \bar{X}_k(N_{\max})$ implies $N_{\max}[\bar{X}_i(N_{\max}) - \bar{X}_k(N_{\max})] \ge h^2(S_i^2+S_k^2) - \lambda N_{\max}$. Notice that for $\ell = 1, 2, \ldots, t$, $X_{i,\ell} - X_{k,\ell} + \lambda$ are i.i.d. normal random variables with mean $\mu_i - \mu_k + \lambda < 0$ and variance $\sigma_i^2 + \sigma_k^2$, and for $t \ge n_0$, $\sum_{\ell=1}^{t}(X_{i,\ell} - X_{k,\ell} + \lambda)$ is independent of S_i^2 and S_k^2. Then we can rewrite Equation (16) as,

$$\begin{aligned}
(16) &= \mathbb{E}\left[\mathbb{P}\left(\max_{n_0\le t\le N_{\max}}\sum_{\ell=1}^{t}\left(X_{i,\ell}-X_{k,\ell}+\lambda\right)\ge h^2\left(S_i^2+S_k^2\right)\middle| S_i^2, S_k^2\right)\right]\\
&\le \mathbb{E}\left[\mathbb{P}\left(\sum_{\ell=1}^{t}\left(X_{i,\ell}-X_{k,\ell}+\lambda\right)\ge h^2\left(S_i^2+S_k^2\right)\text{ for some } t<\infty\middle| S_i^2, S_k^2\right)\right]\\
&\le \mathbb{E}\left[\exp\left(\frac{2\left(\mu_i-\mu_k+\lambda\right)h^2\left(S_i^2+S_k^2\right)}{\sigma_i^2+\sigma_k^2}\right)\right] \quad \text{(by Lemma 1)}\\
&\le \mathbb{E}\left[\exp\left(\frac{2\left(\lambda-\delta\right)h^2\left(S_i^2+S_k^2\right)}{\sigma_i^2+\sigma_k^2}\right)\right]\\
&= \mathbb{E}\left[\exp\left(2\left(\lambda-\delta\right)h^2\frac{\sigma_i^2}{\sigma_i^2+\sigma_k^2}\frac{S_i^2}{\sigma_i^2}\right)\right]\mathbb{E}\left[\exp\left(2\left(\lambda-\delta\right)h^2\frac{\sigma_k^2}{\sigma_i^2+\sigma_k^2}\frac{S_k^2}{\sigma_k^2}\right)\right]. \qquad (17)
\end{aligned}$$

Notice that Lemma 1 is applied to get the third equation, the second-to-last equation holds because $\mu_i \leq \mu_k - \delta$, and the last equality holds because the observations generated by alternatives i and k are independent. Since both random variables $(n_0-1)S_i^2/\sigma_i^2$ and $(n_0-1)S_k^2/\sigma_k^2$ follow χ^2 distribution with n_0-1 degrees of freedom. Using the moment generating function of the χ^2 random variable, and also the definition of h^2, Equation (17) can be rephrased to,

$$\begin{aligned}(17) &= \left(1+4\frac{(\delta-\lambda)\,h^2}{n_0-1}\frac{\sigma_i^2}{\sigma_i^2+\sigma_k^2}\right)^{-\frac{n_0-1}{2}} \times \left(1+4\frac{(\delta-\lambda)\,h^2}{n_0-1}\frac{\sigma_k^2}{\sigma_i^2+\sigma_k^2}\right)^{-\frac{n_0-1}{2}} \\ &\leq \left(1+4\frac{(\delta-\lambda)\,h^2}{n_0-1}\right)^{-\frac{n_0-1}{2}} = \frac{\alpha}{k-1}. \end{aligned} \tag{18}$$

Plug in the result in (18) to (15), it yields,

$$\mathbb{P}\left(\text{select alt. } k\right) \geq 1 - \sum_{i=1}^{k-1} \mathbb{P}\left(\text{alt. } i \text{ eliminates alt. } k\right) \geq 1-\alpha.$$

It concludes the proof. □

6.3. PPP with PAC Guarantee

We are aware that as pointed out by Ni et al. (2017), when the number of alternatives gets large, the IZ formulation may become questionable because it is difficult to ensure that the mean of the best alternative is at least δ larger than that of the second best. Indeed, there may exist multiple alternatives whose means lie within δ to the mean of the best. Then, the IZ assumption no longer holds. To address the issue, Ni et al. (2017) use a selection guarantee which requires the procedures to select an alternative which lies within δ to the best with the predefined probability even when the IZ assumption is violated, i.e.,

$$\mathbb{P}\left(\mu_k - \mu_I \leq \delta\right) \geq 1-\alpha,$$

where I is the index of the alternative that the procedure terminates with. It is called the *probably approximately correct* (PAC) guarantee. Notice that the PAC guarantee makes no assumptions on the configuration of the means and is stronger than the PCS guarantee. For many fully sequential procedures which satisfy the PCS guarantee, it remains an open question as to whether they satisfy the PAC guarantee. However, as shown by Kao and Lai (1980), with a slight modification, i.e., changing the screening process in (1) to,

$$\mathcal{I} = \left\{ i : i \in \mathcal{I}_{\text{old}} \text{ and } Z_{ij}\left(t\right) \geq -h^2 S_{ij}^2 - \left(\delta-\lambda\right)t, \forall j \in \mathcal{I}_{\text{old}}, j \neq i \right\},$$

Paulson's procedure can satisfy the PAC guarantee. Because our parallel procedure is built upon Paulson's procedure, in Theorem 3 below, we show that the same modification is applicable to our parallel procedure as well, and the proof is provided in the Appendix.

THEOREM 3. *If the means of the alternatives are in ascending order, i.e.,* $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_k$*, and the screening process (14) in PPP is changed to*

$$\mathcal{I}^s = \left\{ i : i \in \mathcal{I}^s_{\text{old}} \text{ and } W_i^+(t) \geq W_{i^*_-}^-(t) - (\delta - \lambda)\, t \right\}, \tag{19}$$

then PPP can guarantee that,

$$\mathbb{P}\left(\mu_k - \mu_I \leq \delta\right) \geq 1 - \alpha,$$

where I *is the index of the alternative that is finally selected.*

Notice that in order to satisfy the PAC guarantee, the screening process (19) essentially bumps out the continuation region of PPP by $\delta \cdot t$. After the modification, the procedure's continuation region is uniformly larger than that of PPP and the procedure requires more observations to eliminate alternatives. There is an efficiency loss on the total sample size. In the rest of this paper, we call the procedure using the screening process (19) as PAC-PPP. We close this section with a remark that even though there is no rigorous proof showing that existing fully sequential procedures, which satisfy the PCS guarantee, also theoretically satisfy the PAC guarantee, in practice, when the IZ assumption is violated, they do deliver the PAC guarantee, e.g., Luo et al. (2015).

7. Numerical Experiments

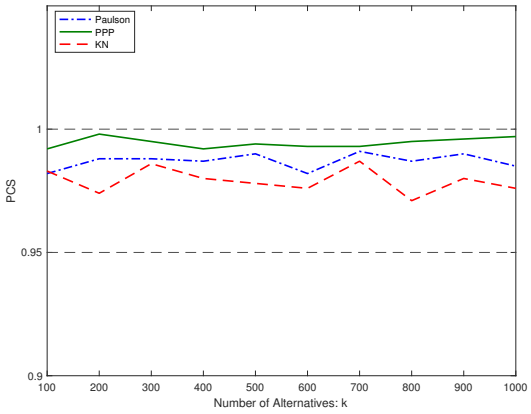
In this section, we test the performance of our procedures and some existing R&S procedures. The goals of the numerical studies are as follows. First, we show that compared to traditional fully sequential procedures, our procedures indeed can significantly reduce the comparison time while keeping the total sample size roughly unchanged. Second, with a real simulation problem, we demonstrate that the batching method is preferred over the one-by-one task assignment scheme when the network latency is high. We also show that our parallel procedure is very competitive with some existing parallel R&S procedures, including the Vector-filling KN (VKN) procedure of Luo et al. (2015), the Good Selection Procedure (GSP) of Ni et al. (2017), and the Parallel Survivor Selection (PSS) procedure of Pei et al. (2018).

For the numerical experiments listed in this section, when the procedures are implemented in a single-processor computing environment, the programming language is `Java`. When parallel computing is used, the underlying parallel computing platform is Message Passing Interface (MPI) and the default programming language is `C++`. MPI is considered as one of the most efficient parallel computing platforms. It allows direct communications between any two processors, and the users to customize and optimize the implementation of a parallel procedure from low-level. In this paper, the implementation of MPI is MPICH-3.3.2 which can be downloaded from `https://www.mpich.org/`. For all the procedures implemented in this section, the desired PICS α and IZ parameter δ are set to be 0.05 and 0.1 respectively. To estimate the variance of each alternative, by default, the first-stage sample size n_0 is set to be 50. All the codes used in this section can be retrieved from `https://github.com/biazhong/PPP`.

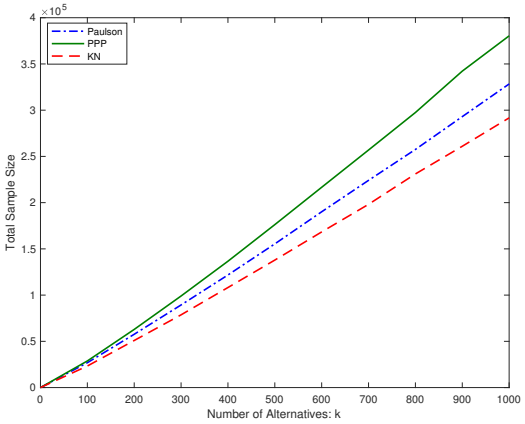
7.1. Normal Observation Test Problems

In this subsection, we use a simple example where all alternatives draw observations from normal distributions to test the performance of PPP and two classical fully sequential procedures, Paulson's procedure and the KN procedure, in a single-processor computing environment. Particularly, we conduct the experiment on a local PC with Intel Core i7-3770 CPU (4 processors)[2], 8 GB memory, and Linux Ubuntu 16.04 LTS operating system. In a single-processor computing environment, the only difference between PPP and Paulson's procedure is the screening process used at each round, i.e., PPP uses the screening process (14) and Paulson's procedure uses the screening process (1). For the test problems, we consider the slippage configuration of the means, i.e., $\mu_1 = \mu_2 = \cdots = \mu_{k-1} = \mu_k - \delta = 0$. This mean configuration is often treated as the most difficult setting in R&S literature. All alternatives share a common but unknown variance 0.25, i.e., $\sigma_1^2 = \sigma_2^2 = \cdots = \sigma_k^2 = 0.25$. For PPP and Paulson's procedure, the user-specified parameter λ is set to be $\delta/2$, and for the KN procedure, the user-specified parameter c is set to be 1. We test the performance of all three procedures with the number of alternatives $k = a \times 10^2$, where $a = 1, 2, \ldots, 10$. In Figure 4, based on 1,000 independent macro-replications we report total sample sizes, and wall-clock times of the three procedures as they solve these problems.

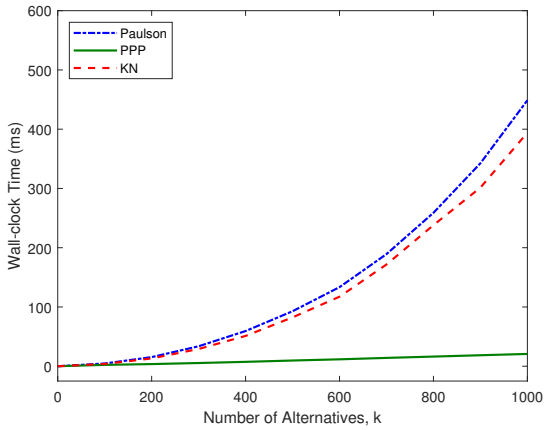
[2] Only 1 processor is used in this experiment.



(a) PCS



(b) Total Sample Size



(c) Wall-clock Time

Figure 4 Comparisons of PPP, Paulson's procedure, and the KN procedure.

According to the results, we have three findings. First, all three procedures over-deliver PCS in this experiment. This is because, while developing these procedures, some conservative inequalities are used, e.g., Bonferroni's inequality. As a result, these procedures often require more observations than necessary. Second, in terms of the total sample size, the KN procedure is the most efficient procedure while PPP is the least efficient procedure among these three procedures. Even though PPP requires more observations to identify the best than the other two procedures, in general, the difference is not very large and is tolerable. Third, because generating a normal random observation takes almost no time and there is no communication cost in a single processor, in this experiment, the wall-clock times reflect the actual comparison times of these three procedures. From Figure 4(c), we can conclude that the comparison times of Paulson's procedure and the KN procedure grow in quadratic-like rates, and they grow much faster than that of PPP does.

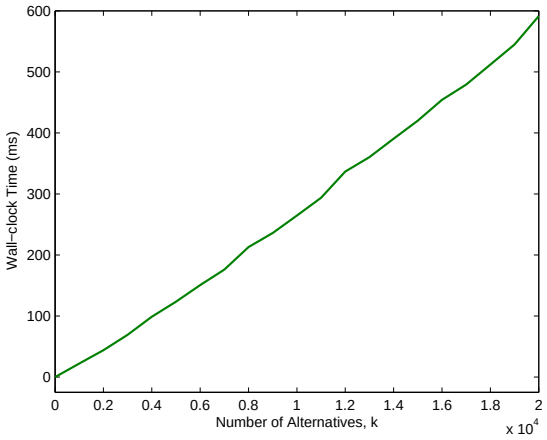


Figure 5 The average wall-clock time of PPP.

To further investigate the exact growth rate of the comparison time of PPP, we separately let PPP solve larger scale problems where $k = a \times 10^3$ and $a = 1, 2, \ldots, 20$. We plot the wall-clock times as PPP solves these problems in Figure 5.

From Figure 5, we observe that being different from those of Paulson's procedure and the KN procedure, the comparison time of PPP appears to grow linearly in k. This is consistent with our theoretical analysis that PPP can reduce the computational complexity of comparisons and thus the comparison time significantly when the number of alternatives is large. It is also worthwhile noting that, from the results in Figure 4(c) and Figure 5, we observe that, for this particular example, the wall-clock time that PPP takes to solve a problem with 20,000 alternatives is about the same as the wall-clock times that Paulson's procedure and the KN procedure require to solve the problem with 1,000 alternatives. It suggests that PPP indeed has the potential to solve large-scale problems.

7.2. Three-stage Buffer Allocation Problems

In this subsection, we test the performance of our procedures and some existing parallel procedures with the three-stage buffer allocation problem of Buzacott and Shanthikumar (1993). This problem is considered as a practical simulation testing problem, which has already been included in the simulation optimization library of Henderson and Pasupathy (2014) (`http://simopt.org`), and also been studied in the existing parallel R&S papers, i.e., Luo et al. (2015) and Ni et al. (2017).

The problem considers a flowline with three stations 1, 2, and 3. There is an infinite number of jobs waiting in front of station 1. Each job will be sequentially processed by the

Table 1 Summary of three-stage buffer allocation problems.

Configuration (R, B)	Number of alternatives k	Highest mean μ_k	Number of alternatives within $[\mu_k - \delta, \mu_k]$
(20,20)	3,249	5.78	21
(50,50)	57,624	15.70	43
(128,128)	1,016,127	41.66	97

three stations (from station 1 to station 3). The service times of processing jobs at station i, where $i = 1, 2, 3$, are independently and exponentially distributed with service rate x_i. In front of stations 2 and 3, there are buffers of capacities x_4 and x_5, respectively. If the buffer in front of station i, where $i = 2, 3$, is full, the upstream station $i - 1$ is blocked. At this time, the finished job in station $i - 1$ cannot be released, and the station cannot process new jobs. The service times and the buffer capacities are assumed to be positive integers and constrained by $x_1 + x_2 + x_3 = R$ and $x_4 + x_5 = B$, where R and B are two pre-specified positive integers. By setting different values to R and B, there are different numbers of feasible solutions (alternatives) to the problem. Our goal is to find an optimal allocation of service rates and buffer capacities such that the steady-state throughput of the flowline is maximized. While applying R&S procedures to solve this problem, we observe the random throughputs of a feasible solution by conducting simulation experiments. For each simulation experiment, after a warm-up period of releasing 2,000 jobs, we observe the throughput of subsequent 50 jobs. Notice that in this subsection, the observations generated by different alternatives are not normally distributed.

In the following experiments, we consider three different combinations of the values for R and B. They correspond to small-, medium-, and large-scale R&S problems. We summarize the detailed information about the problems in Table 1. Notice that because this flowline system can be alternatively viewed as a continuous-time Markov chain and for this example, the service times are exponentially distributed, we can analytically calculate the expected throughput of each feasible solution by solving the balance equations for the underlying Markov chain from Buzacott and Shanthikumar (1993). Thus, we are able to provide some detailed information on these problems in Table 1.[3]

[3] Recall that we set $\delta = 0.1$ throughout this section.

7.2.1. A Comparison of Two Task Assignment Schemes. Before comparing our procedures with other parallel R&S procedures, we first take the problem with 57,624 alternatives as an example to examine the wall-clock times for the two task assignment schemes discussed in Section 4, i.e., one-by-one and batching of alternatives, required to generate one round of observations. We conduct the experiment on two local PCs, which are the same as the one in the previous experiment and connected via the Internet. We use four processors to run the simulation experiments and consider two different settings of the processors: (1) all four processors are located in the same PC; (2) the master processor and the other three processors are located separately in two PCs. These two different settings represent two levels of communication cost between the master and the workers. The one-way network latencies between the master and a worker, i.e., t_c, in the first setting and the second setting are about 0.013ms and 2.2ms, respectively. The communication cost in the second setting is much higher than that of the first setting. In this experiment, the average time needed to generate one observation in a worker, i.e., θ, is about 1ms. In order to test how the variations in simulation times may affect the performance of the two task assignment schemes, after a worker simulates an observation, we pause it for some time κ. We consider three different settings for the pause time κ: (1) $\kappa = 1$ms; (2) κ is sampled from a uniform distribution with mean 1ms and width 1ms, i.e., $\kappa \sim \text{Unif}(0.5\text{ms}, 1.5\text{ms})$; (3) $\kappa \sim \text{Unif}(0\text{ms}, 2\text{ms})$. These three settings represent different levels of variability in simulation times. In Table 2, based on 100 independent macro-replications, we report the average wall-clock times the two task assignment schemes require to generate one round of observations and their 95% confidence intervals under different settings.

Table 2 A comparison of wall-clock times (sec) of generating one round of observations for the problem with 57,624 alternatives between one-by-one and batching using 4 processors.

Pause time, κ	Processors in one PC		Processors in two PCs	
	One-by-one	Batching	One-by-one	Batching
$\kappa = 1$ms	33.32 ± 0.01	33.30 ± 0.01	111.17 ± 0.23	33.64 ± 0.01
$\kappa \sim \text{Unif}(0.5\text{ms}, 1.5\text{ms})$	33.36 ± 0.01	33.33 ± 0.01	110.86 ± 0.41	33.87 ± 0.01
$\kappa \sim \text{Unif}(0\text{ms}, 2\text{ms})$	33.34 ± 0.01	33.49 ± 0.01	111.66 ± 0.40	34.04 ± 0.01

We highlight the main findings from this experiment as follows. First, when the processors are in the same PC, the communication time between the master and workers is

relatively small compared to the simulation time. Therefore, the wall-clock times that the two task assignment schemes require to generate one round of observations are roughly the same. Second, the performance of the one-by-one task assignment scheme depends heavily on the network latency. When the processors are spread across two PCs, it uses significantly more wall-clock time to generate observations than it does for the case where the processors are in the same PC. Meanwhile, for the batching method, we only observe a slight increase on the wall-clock time as we allocate the master and workers in different PCs. This verifies our analysis in Section 4.3. Third, comparing the results in different columns, it seems that, in this example, the variations in simulation times do not affect the performance of the two task assignment schemes significantly. For the one-by-one task assignment scheme, because it can always finely balance the workloads of different processors, the impact of variability in simulation times is limited. For the batching method, even though the time needed to simulate a single observation may vary significantly, due to the large batch size, the average times needed to simulate an observation in different workers are roughly the same. In this situation, the variability in simulation times would not affect the performance of the method too much either.

7.2.2. Solving the Problem with 3,249 Alternatives. We conduct experiments on a local server to test the performance of our procedures and some existing parallel R&S procedures. Specifically, the local server has 48 processors, 128 GB of memory, and Red Hat Enterprise Linux (RHEL) 7.4 operating system, and we compare PPP and PAC-PPP with the VKN procedure of Luo et al. (2015), GSP of Ni et al. (2017), and the PSS procedure of Pei et al. (2018):

- *The VKN procedure* is a variant of the KN procedure in parallel computing environments and considered to be the earliest parallel R&S procedure. The procedure inherits all selection features as well as the statistical validity from the KN procedure. It can be thought as a direct implementation of the KN procedure in parallel computing environments. Because the procedure requires one to store every individual simulation output and the first-stage sample variance of the partial sum difference between any two alternatives, the procedure needs a large amount of memory in practice. Therefore, it is difficult to deploy the procedure at scale and use it to solve very large-scale problems. For example, to solve the problem with 57,624 alternatives, storing the first-stage sample variances as the double data type alone would cost more than 20 GB memory.

- *GSP* is the first parallel R&S procedure that can be used to solve large-scale problems with over a million alternatives. It significantly improves the performance of a fully sequential procedure in parallel computing environments by using the strategies of "divide-and-conquer" to reduce the computational complexity of comparisons at each round and "batching the observations from the same alternative" to reduce the communication time. Also, different from the VKN procedure, GSP can theoretically satisfy the stronger PAC guarantee while solving large-scale R&S problems. At the beginning of the selection process, GSP requires the user to specify the batch size β of observations that each alternative can take at a time and the maximum number of rounds $\bar{r}$ that the screening process is conducted. While implementing GSP in this paper, we set $(\beta, \bar{r}) = (100, 10)$ and $(\beta, \bar{r}) = (200, 5)$ as recommended by the authors.
- *The PSS procedure* is a recently developed parallel procedure. Compared to existing parallel R&S procedures, it has a different scope. It provides an expected false elimination rate (EFER) guarantee upon stopping and can be used with various stopping conditions. In this paper, we consider the fixed-budget setting for the procedure, i.e., while conducting the selection, the total computational budget (total sample size) is fixed. Under the fixed budget setting, the procedure aims to deliver a subset of alternatives which contains the best alternative when the computational budget is consumed. The procedure requires one to input the mean performance of the best alternative and the computational budget at the beginning of the selection process. In order to make the procedure comparable with our procedures, while solving a problem, we first use PAC-PPP to solve the problem. Then, according to the results of PAC-PPP, we set the total sample size of the PSS procedure to be roughly the same as that of PAC-PPP. For the procedure, we additionally set the number of observations that each alternative can take at a time to be 10 as recommended by the authors, and the EFER to be 0.05.

We first use all the procedures listed above to solve the problem with 3,249 alternatives. In this experiment, we do not pause a worker after it simulates an observation. Based on 100 independent macro-replications, we examine the wall-clock time, the total simulation time, and the total comparison time of each procedure and their 95% confidence intervals. Note that the total simulation time and the total comparison time of a procedure are the simulation time and the comparison time summed across all processors. Because the IZ

Table 3 A comparison of PPP ($\lambda = \delta/2$) and PAC-PPP ($\lambda = \delta/2$) with other parallel R&S procedures using the configuration: $k = 3,249$ alternatives on 48 processors.

Procedure	PAC	Total sample size ($\times 10^5$)	Wall-clock time ($\times 10$ sec)	Total simulation time ($\times 10^2$ sec)	Total comparison time ($\times 10^{-3}$ sec)	Utilization
PPP	1.00	4.04 ± 0.03	1.74 ± 0.02	7.73 ± 0.06	4.27 ± 0.06	94.36%
PAC-PPP	1.00	4.18 ± 0.04	1.80 ± 0.02	8.00 ± 0.05	5.14 ± 0.08	94.82%
VKN	1.00	2.12 ± 0.08	2.63 ± 0.01	4.09 ± 0.02	$16,750 \pm 40$	33.11%
GSP ($\beta = 100$)	1.00	4.68 ± 0.06	2.28 ± 0.01	9.26 ± 0.07	14.68 ± 0.16	86.45%
GSP ($\beta = 200$)	1.00	6.53 ± 0.08	2.95 ± 0.01	12.37 ± 0.09	13.59 ± 0.15	89.10%
PSS	0.99	4.20 ± 0.00	2.80 ± 0.02	8.13 ± 0.02	4.12 ± 0.02	61.82%

assumption is violated in this problem, instead of reporting the PCS, we focus on estimating the PAC probabilities of different procedures. Furthermore, because the PSS procedure returns a subset of alternatives upon stopping, while reporting the PAC probability of the procedure, we report the proportion of the alternatives whose mean are within δ to the best alternative. Also, in this example, we set the total sample size of the PSS procedure to be 4.2×10^5 which is roughly the same as that of PAC-PPP. Lastly, as a measure of a procedure's efficiency in parallel computing environments, we calculate the *utilization*, where

$$\text{Utilization} = \frac{\text{Total simulation time}}{\text{Wall-clock time} \times \text{Number of workers}},$$

for each procedure. We report the results in Table 3.

From the results in Table 3, we may draw following conclusions. First, in this experiment, all procedures numerically demonstrate the ability to select an alternative within δ of the best, although only PAC-PPP and GSP can theoretically satisfy the PAC guarantee. Second, the VKN procedure uses the smallest amount of observations to conduct the selection. However, the procedure's utilization is low due to the long comparison time. It suggests that in parallel computing environments, the total sample size is no longer the only criterion to measure the performance of a procedure. Third, besides the VKN procedure, the utilization of the PSS procedure is not satisfactory as well. The main reason is that for the PSS procedure, once the number of surviving alternatives falls below the number of workers in the parallel computing environments, there are more and more workers staying idle. In the meantime, when only a few alternatives are left in this example, these alternatives tend to be very good alternatives. It becomes increasingly difficult to

eliminate alternatives. Therefore, many workers would stay idle for a relatively long time, and it leads to a low utilization. Fourth, in terms of the wall-clock time and the total sample size, we may conclude that PPP and PAC-PPP perform relatively well compared to the other procedures.

Because among all the procedures, only PAC-PPP and GSP can theoretically satisfy the PAC guarantee which is more suitable for large-scale problems, in the remaining experiments, we focus on comparing these two procedures. Then, we test the wall-clock times that PAC-PPP and GSP require to solve the small-scale problem as different numbers of processors in the local server are used. Based on 100 independent macro-replications, we report the results in Figure 6.

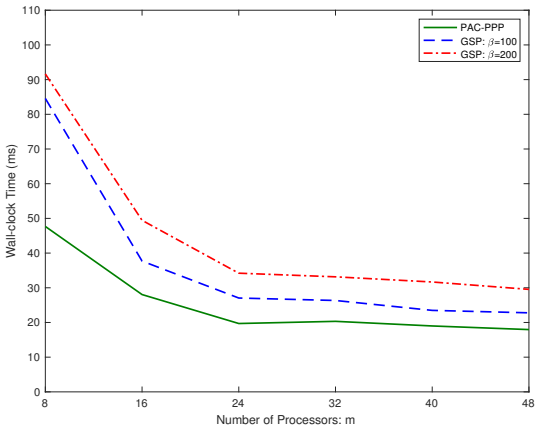


Figure 6 **A comparison between PAC-PPP ($\lambda = \delta/2$) and GSP using the configuration: k = 3,249 on different number of processors.**

From Figure 6, we find that, in general, PAC-PPP requires less wall-clock time to solve the problem than GSP does under all settings. However, for both procedures, after the number of processors exceeds 24, as more processors are added to the parallel computing environment, we hardly observe any decrease on the wall-clock time as expected. A possible reason is that, in this experiment, the total computing resource, e.g., the memory, of the local server is limited. As more processors are running, the average amount of computing resource assigned to each processor decreases. As a result, more time is needed for a processor to simulate an observation. Even though there are more workers, the total simulation time that all workers require to simulate a fixed number of observations becomes larger. Therefore, after the number of processors exceeds a threshold, by simply adding more

Table 4 Testing the performance of PAC-PPP ($\lambda = \delta/2$) with random simulation times using the configuration: $k = 3,249$ alternatives on 48 processors.

Pause time, κ	Total sample size ($\times 10^5$)	Wall-clock time ($\times 10$ sec)	Total simulation time ($\times 10^3$ sec)	Total comparison time ($\times 10^{-3}$ sec)	Utilization
$\kappa = 1$ms	4.20 ± 0.05	2.69 ± 0.03	1.18 ± 0.01	5.21 ± 0.13	93.27%
$\kappa \sim \text{Unif}(0.5\text{ms}, 1.5\text{ms})$	4.21 ± 0.04	2.74 ± 0.03	1.20 ± 0.01	5.14 ± 0.13	93.14%
$\kappa \sim \text{Unif}(0\text{ms}, 2\text{ms})$	4.20 ± 0.05	2.78 ± 0.03	1.20 ± 0.01	5.17 ± 0.12	91.75%

processors, it may not help the procedures reduce the wall-clock times. In fact, to verify this assumption, we add another identical server to the parallel computing environment and run the instance of 48 processors again. In the new setting, 24 processors are used for each server. Compared to the original setting (48 processors are in the same server), we observe that in the new setting, the wall-clock times for PAC-PPP, GSP ($\beta = 100$), and GSP ($\beta = 200$) indeed decrease from 17.95, 22.78, and 29.54 seconds to 12.79, 15.89, and 19.35 seconds respectively.

Next, we conduct an additional experiment for PAC-PPP to test its robustness to random simulation times. Specifically, we use PAC-PPP to solve the problem under the three settings for the pause time as we considered in Section 7.2.1. We summarize the results and their 95% confidence intervals in Table 4.[4] Because we only run 50 macro-replications for each instance, we do not include the estimated PAC probability in the table.

From Table 4, we can observe that as the variability in simulation times increases, the procedure's utilization decreases and the procedure spends more time on finding the best alternative. In general, the difference in wall-clock times among the three instances is not much and we think it is tolerable.

7.2.3. Solving the Problem with 57,624 Alternatives. With the same local server as that in the previous experiment, in this experiment, we try to use PAC-PPP and GSP to solve the problem with 57,624 alternatives. Because the two procedures can always select an alternative within δ of the best alternative, in the rest of this section, we no longer report the estimated PAC probability. In this experiment, we do not pause a worker after it simulates an observation, and for each instance, we run 10 independent macro-replications. We summarize the results and their 95% confidence intervals in Table 5.

[4] In Table 4, the pause time is included in the simulation time.

Table 5 A Comparison between PAC-PPP ($\lambda = \delta/2$) and GSP using the configuration: $k = 57,624$ alternatives on 48 processors.

k	Procedure	Total sample size ($\times 10^6$)	Wall-clock time ($\times 10^2$ sec)	Total simulation time ($\times 10^3$ sec)	Total comparison time (sec)	Utilization
57,624	PAC-PPP	10.76 ± 0.15	4.47 ± 0.06	19.71 ± 0.29	0.5021 ± 0.016	93.77%
	GSP ($\beta = 100$)	10.83 ± 0.10	4.65 ± 0.11	19.83 ± 0.21	3.7341 ± 0.005	90.67%
	GSP ($\beta = 200$)	13.18 ± 0.11	5.68 ± 0.10	24.43 ± 0.23	3.6585 ± 0.005	91.44%

From Table 5, we can observe that the performance of PAC-PPP is significantly better than that of GSP ($\beta = 200$) and slightly better than that of GSP ($\beta = 100$). One may argue that combining the results in Table 3 and Table 5, for GSP, it seems that as we set a smaller value for β, the procedure can use less wall-clock time and fewer observations to solve the problem. However, it may not always be the case. As one keeps decreasing β, more frequent comparisons are made. This can lead to a rapid increase on the comparison time, and the procedure behaves more and more like the VKN procedure. In practice, without any prior knowledge on the configuration of the means, it may be very hard to determine the optimal value for β. It is interesting to point out that even though, in this experiment, both procedures can still maintain the comparison time at a low level, compared to the results in Table 3, we may conclude that the comparison time of PAC-PPP grows slower than that of GSP does as the the problem size increases.

7.2.4. Solving the Problem with 1,016,127 Alternatives. Finally, we run PAC-PPP and GSP to solve a large-scale problem. We conduct the experiment on two local servers, which are the same as the one in previous experiments. Due to the long wall-clock time of solving the problem, we only run 5 macro-replications for each procedure. We list the results and their 95% confidence intervals in Table 6.

From Table 6, similar to what we have seen in the previous experiments, the performance of PAC-PPP is better than that of GSP ($\beta = 100$) and GSP ($\beta = 200$). However, one may observe that, compared to the results in Table 3 and Table 5, there is a decrease in the utilization of PAC-PPP in Table 6. The main reason is that, for PAC-PPP, inferior alternatives are sequentially eliminated and the batch size of alternatives keeps decreasing. The batching method used in PAC-PPP behaves more and more like the one-by-one task assignment scheme. Especially, in this experiment, the master and some workers are located separately in two different servers. The communication cost between the master and these

Table 6 Comparisons between PAC-PPP ($\lambda = \delta/2$) and GSP using the configuration: $k = 1,016,127$ alternatives on 96 processors.

k	Procedure	Total sample size ($\times 10^8$)	Wall-clock time ($\times 10^3$ sec)	Total simulation time ($\times 10^5$ sec)	Total comparison time ($\times 10$ sec)	Utilization
1,016,127	PAC-PPP	4.03 ± 0.08	8.92 ± 0.18	7.70 ± 0.15	3.02 ± 0.09	89.92%
	GSP ($\beta = 100$)	4.28 ± 0.20	9.38 ± 0.37	8.28 ± 0.41	576.95 ± 1.35	92.97%
	GSP ($\beta = 200$)	4.97 ± 0.15	1.09 ± 0.26	9.71 ± 0.25	566.85 ± 0.92	93.46%

workers is relatively high. It further lowers the utilization of the processors. To address this issue, when the number of surviving alternatives falls below a threshhold, one may instruct a worker to not only simulate observations for a batch of alternatives but also simulate a batch of observations for each individual alternative. It may keep a worker simulating a relatively large amount of observations at a time and increase the utilization. However, it is at the expense of some efficiency loss on the total sample size.

8. Conclusions

In this paper, we make three modifications on the classical fully sequential procedure, Paulson's procedure, to speed up its selection process in parallel computing environments. We show that if no CRNs are used, we can significantly reduce the computational complexity of comparisons to the same order as that of the simulations. Then, the comparison time becomes negligible compared to the simulation time and is no longer the bottleneck of the procedure. Also, we demonstrate that by batching of different alternatives, we can reduce the frequency of the communications and thus the communication time among the processors. It helps the procedure further improve its efficiency. Finally, to boost the selection process in the final-stage, once the number of surviving alternatives is less than the number of workers, we suggest to sample all surviving alternatives to the maximal number of observations they can take. We use several numerical experiments to show that the modified procedure is more efficient compared with existing parallel procedures in the literature.

Acknowledgments

The authors would like to thank the Area Editor, Prof. Bruno Tuffin, the Associate Editor, and two reviewers for their insightful and detailed comments that have significantly improved this paper. This research was supported in part by the Natural Science Foundation of China [Grants 71722006, 72031006 and 72072117].

References

Bechhofer RE (1954) A single-sample multiple decision procedure for ranking means of normal populations with known variances. *The Annals of Mathematical Statistics* 25(1):16–39.

Buzacott JA, Shanthikumar JG (1993) *Stochastic Models of Manufacturing Systems* (Prentice Hall).

Chen CH, Chick SE, Lee LH, Pujowidianto NA (2015) Ranking and selection: Efficent simulation budget allocation. Fu M, ed., *Handbook of Simulation Optimization*, 45–80 (Springer).

Chick SE (2006) Subjective probability and bayesian methodology. Henderson SG, Nelson BL, eds., *Elsevier Handbooks in Operations Research and Management Science: Simulation*, 225–257 (Elsevier).

Henderson SG, Pasupathy R (2014) Simulation optimization library. `http://www.simopt.org`, [Online; accessed 25-August-2019].

Hong LJ (2006) Fully sequential indifference-zone selection procedures with variance-dependent sampling. *Naval Research Logistics* 53(5):464–476.

Hong LJ, Luo J, Zhong Y (2016) Speeding up pairwise comparisons for large scale ranking and selection. Roeder TMK, Frazier PI, Szechtman R, Zhou E, Huschka T, Chick SE, eds., *Proceedings of the 2016 Winter Simulation Conference*, 749–757 (Piscataway, New Jersey: IEEE).

Hong LJ, Nelson BL, Xu J (2015) Discrete optimization via simulation. Fu M, ed., *Handbook of Simulation Optimization*, 9–44 (Springer).

Jennison C, Johnstone IM, Turnbull BW (1980) Asymptotically optimal procedures for sequential adaptive selection of the best of several normal means. Technical report, Department of Operations Research and Industrial Engineering, Cornell University.

Kao SC, Lai TL (1980) Sequential selection procedures based on confidence sequences for normal populations. *Communications in Statistics-Theory and Methods* 9(16):1657–1676.

Kim SH, Nelson BL (2001) A fully sequential procedure for indifference-zone selection in simulation. *ACM Transactions on Modeling and Computer Simulation* 11(3):251–273.

Kim SH, Nelson BL (2006) Selecting the best system. Henderson SG, Nelson BL, eds., *Elsevier Handbooks in Operations Research and Management Science: Simulation*, 501–534 (Elsevier).

Luo J, Hong LJ (2011) Large-scale ranking and selection using cloud computing. Jain S, Creasey RR, Himmelspach J, White KP, Fu M, eds., *Proceedings of the 2011 Winter Simulation Conference*, 4051–4061 (Piscataway, New Jersey: IEEE).

Luo J, Hong LJ, Nelson BL, Wu Y (2015) Fully sequential procedures for large-scale ranking-and-selection problems in parallel computing environments. *Operations Research* 63(5):1177–1194.

Ma S, Henderson SG (2017) An efficient fully sequential selection procedure guaranteeing probably approximately correct selection. Chan WKV, D'Ambrogio A, Zacharewicz G, Mustafee M, eds., *Proceedings of the 2017 Winter Simulation Conference*, 2225–2236 (Piscataway, New Jersey: IEEE).

Nelson BL, Swann J, Goldsman D, Song W (2001) Simple procedures for selecting the best simulated system when the number of alternatives is large. *Operations Research* 49(6):950–963.

Ni EC, Ciocan DF, Henderson SG, Hunter SR (2017) Efficient ranking and selection in parallel computing environments. *Operations Research* 65(3):821–836.

Paulson E (1964) A sequential procedure for selecting the population with the largest mean from k normal populations. *The Annals of Mathematical Statistics* 35(1):174–180.

Pei L, Nelson BL, Hunter S (2018) A new framework for parallel ranking & selection using an adaptive standard. Rabe M, Juan AA, Mustafee N, Skoogh A, Jain S, Johansson B, eds., *Proceedings of the 2018 Winter Simulation Conference*, 2201–2212 (Piscataway, New Jersey: IEEE).

Perng S, et al. (1969) A comparison of the asymptotic expected sample sizes of two sequential procedures for ranking problem. *The Annals of Mathematical Statistics* 40(6):2198–2202.

Rinott Y (1978) On two-stage selection procedures and related probability-inequalities. *Communications in Statistics - Theory and Methods* A7:799–811.