Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

LET'S DO RANKING & SELECTION

Barry L Nelson

Department Industrial Engineering & Management Sciences
Northwestern University
Evanston, IL 60208, USA

ABSTRACT

Many tutorials and survey papers have been written on ranking & selection because it is such a useful tool for simulation optimization when the number of feasible solutions or "systems" is small enough that all of them can be simulated. Cheap, ubiquitous, parallel computing has greatly increased the "all of them can be simulated" limit. Naturally these tutorials and surveys have focused on the underlying theory of R&S and have provided pseudocode procedures. This tutorial, by contrast, emphasizes applications, programming and interpretation of R&S, using the `R` programming language for illustration. Readers (and the audience) can download the code and follow along with the examples, but no experience with `R` is needed.

1 INTRODUCTION

Due to the theoretical and practical success of ranking & selection (R&S) for simulation optimization, a number of tutorials and survey papers have documented useful procedures and the theory behind them. These include Kim and Nelson (2006), Frazier (2012), Chen et al. (2015), Hunter and Nelson (2017) and Hong et al. (2021). This tutorial is distinctly different in that the emphasis is on actually applying R&S procedures to realistic (although not real) problems; it was derived from an online Masterclass of videos, text, software and experiments that can be found at http://users.iems.northwestern.edu/~nelsonb/RSMasterclass.html. The software and examples for this tutorial may be downloaded from http://users.iems.northwestern.edu/~nelsonb/WSC2022Tutorial.html. All code is written in `R`, and is most easily executed from within RStudio (https://www.rstudio.com/), although Base R (https://www.r-project.org/) will suffice. To follow along the reader should download the three RScript files `Simulations.R`, `Procedures.R` and `ParallelProcedures.R`, open them in `R` or RStudio, and source them into the active window. However, the tutorial is both useful and understandable even if you do not use `R`.

This tutorial will only attack the "best mean" problem: selecting which of k systems has the largest (or near the largest) mean or expected value of performance with some statistical guarantee. We let $Y_j(x)$ be an output from the jth independent and identically distributed (i.i.d.) replication of system $x = 1, 2, \ldots, k$ with mean $\mu(x) = \mathrm{E}[Y_j(x)]$ and variance $\sigma^2(x) = \mathrm{Var}[Y_j(x)]$. Larger $\mu(x)$ is considered better. We let $x^\star$ be the index of the best system (there are no ties in the examples), and $\widehat{x}^\star$ the index of the system selected by the R&S procedure if it selects only one. Many R&S procedures assume that the outputs are normally distributed. Some procedures are *fixed precision*, which means they terminate when some prespecified statistical guarantee is achieved, while others are *fixed budget*, which means they expend a computational budget so as to attain as strong an inference as possible. All of the R&S procedures considered here treat the systems as *categorical*, so x is just an index and not some location in decision space.

The tutorial will employ five simulation models having different characteristics that are described in Section 2. In the sections that follow we walk the reader through several examples, including procedures that exploit parallel computing. Real-world applications may be found in WSC *Proceedings* since 1983.

2 THE MODELS

Five simulation models are employed in this tutorial to illustrate R&S. All of the R&S procedures are coded for maximization, so in some cases the simulation output is the negative of the natural response.

1. `TTF`: $Y(x)$ is the time to failure for $k = 4$ system designs that use redundancy to make the system resistant to failure. The output is highly variable and simulation execution is slow.
2. `Normal`: $Y(x) \sim \mathrm{N}(\mu(x), \sigma^2)$, where there are $k = 11$ different values of $\mu(x)$, {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}, so $x^\star = 11$, and $\sigma = 2$. This model satisfies the assumptions of any R&S procedure we try.
3. `Invt`: $Y(x)$ is $-$(cost of the inventory policy) for an (s,S) system. There are $k = 1600$ combinations of reorder point s and order-up-to level S that attempt to balance ordering, holding and lost sales costs. In this example there are many systems with similar performance.
4. `SAN`: $Y(x)$ is $-$(time to complete a stochastic activity network). The $k = 5$ systems allocate resources to one of the activities $A_1, A_2, \ldots, A_5$ to reduce the time to complete project. Specifically, the output is $Y(x) = -\max\{A_1(x)+A_4(x), A_1(x)+A_3(x)+A_5(x), A_2(x)+A_5(x)\}$, where the distributions of the activity times depend on x.
5. `MM1`: $Y(x)$ is $-$(long-run average cost of waiting + cost of service) for an M/M/1 queue. There are $k = 100$ different service rates that cost more for faster service. This is the second slowest simulation, with low variance of output, but many close competitors.

All of the R&S procedures in the following sections take as an argument `MySim` which is the name of the model to simulate, for example `Invt`. The simulation models themselves have arguments `(type, x, n, RandomSeed)`, where `x` defaults to system 1, `n` defaults to 1 replication, and `RandomSeed` defaults to the background random number seed in `R`. The argument `type` determines whether the function returns simulation results or the number of systems in this example. For instance, `Invt("sim", 3, 10)` generates 10 replications of system 3 for the `Invt` example using the background `R` seed, while `Invt("k")` returns the number of systems, which is 1600. See `Simulations.R` for the code.

3 PROGRAMMING R&S

Most R&S procedures for the best mean problem either track sample means, say $\bar{Y}(x) = \sum_{j=1}^{n} Y_j(x)/n$, or sums of pairwise differences, say $\sum_{j=1}^{n}(Y_j(x) - Y_j(x'))$, so the `R` functions `mean` and `sum` appear in the code. Sample variances $S^2(x)$ are often needed and computed using `var`. In the `gCEI` procedure of Section 5, the sample means and variances are updated frequently so a well-known update formula is used. Within the procedures we make use of the `R` function `c` to concatenate data, and `cbind` to bind vectors of data into columns of a matrix

Many R&S procedures depend on critical values that can be expressed as the solution to numerical root-finding problems. In the past great effort was expended to create tables of such values. However, these critical values are very often quantiles of some complicated random variable that can nevertheless be simulated easily. Thus, a more modern view is to treat finding these critical values as a quantile-estimation problem, one that can often be solved in real time as we do in the next section.

Our R&S procedures return more than just the selected system, for instance how many replications were obtained from each system and a point estimate of $\mu(x)$. To that end we use the `list` function of `R` to concatenate all returned information which can be referenced using the `result$component` syntax.

Finally, the R&S procedures are coded for clarity rather than maximum efficiency. In particular, in some functions certain loops could be avoided.

```
Rinott <- function(alpha, n0, delta, MySim){
  # implements Rinott's procedure
  # k = number of systems
  # 1-alpha = desired PCS
  # n0 = first-stage sample size
  # delta = indifference-zone parameter
  # note: uses 99% UCB for Rinott's h
  k <- MySim("k")
  h <- Rinotth(k, n0, 1-alpha, 0.99, 10000)$UCB
  Ybar <- NULL
  Vars <- NULL
  Ns <- NULL
# loop through the k systems
for (x in 1:k){
   Y <- MySim("sim", x, n0)
   S2 <- var(Y)
   N <- ceiling(h^2*S2/delta^2)
   if (N > n0){
     Y <- c(Y, MySim("sim", x, N-n0))
   }
   Ybar <- c(Ybar, mean(Y))
   Vars <- c(Vars, S2)
   N <- max(N, n0)
   Ns <- c(Ns, N)}
   list(Best = which.max(Ybar),
      Ybar = Ybar, Var = Vars, N = Ns)}
```

Figure 1: `Rinott` with the call to estimate its critical value highlighted.

4 A BASIC TWO-STAGE PROCEDURE

Among the many fixed-precision procedures available, perhaps the easiest to implement is Rinott (1978). Rinott's procedure provides two guarantees when the outputs are normally distributed and systems are simulated independently:

$$\text{Probability of Correct Selection (PCS): } \Pr\{\widehat{x}^\star = x^\star | \mu(x^\star) - \mu(x) \geq \delta, \forall x \neq x^\star\} \geq 1-\alpha$$
$$\text{Probability of Good Selection (PGS): } \Pr\{\mu(x^\star) - \mu(\widehat{x}^\star) < \delta\} \geq 1-\alpha$$

where α is the allowable error and δ is the allowable optimality gap. The PGS guarantee was established by Matejcik and Nelson (1995). PCS and PGS are standard objectives for fixed-precision procedures; see Eckman and Henderson (2018) for a broad discussion of the connections between PCS and PGS.

Rinott's procedure obtains a first-stage sample of `n0` replications from each system, calculates sample variances, and then computes how many additional replications are needed from each system before selecting the one with the largest sample mean; see Figure 1. The second-stage sample size is based on a power calculation for detecting differences of at least δ, a calculation that can be done for each system separately, without reference to the results from other systems. This makes `Rinott` easy to parallelize, but also inefficient because it learns nothing from the results of the first stage, other than the sample variances. Rinott's procedure is based on a very pessimistic scenario: $\mu(x^\star) - \mu(x) = \delta$, $\forall x \neq x^\star$. To try `Rinott` on the `TTF` problem, execute the statements below, but select your own seed.

```
set.seed(211256)
result <- Rinott(0.05, 50, 1000, TTF)
result
$Best
[1] 2
$Ybar
[1] 9410.758 9754.664 8356.336 9119.266
$Var
[1]  82758221  63909810  78679034 127332540
$N
[1]  809  625  769 1244
```

With the seed shown above we obtain $\widehat{x}^\star = 2$ when $1-\alpha = 0.95$ and $\delta = 1000$; the number of replications per system ranges from 625 to 1244. One way to state the conclusion is "with 95% confidence, the true mean time to failure of system 2 is no more than 1000 time units from that of the true best system (PGS), and if the best and second-best differ by 1000 or more time units then with 95% confidence system 2 is the unique best (PCS)." Looking at the sample means, $\delta = 1000$ time units is quite generous; it was

```
gcei <- function(n0, Nmax, MySim){
  # Algorithm that simulates system
  # with most negative gCEI or sum of gCEI
  # k = number of systems
  # n0 = reps for initial variance estimate
  # Nmax = max replications before termination
  k <- MySim("k")
  Ybar <- rep(0, k) # vector of sample means
  Sum2 <- rep(0, k) # vector of sums of squares
  N <- rep(0, k)    # vector of sample sizes
  Y <- rep(0, k)
  gCEI <- rep(0, k)
  xpath <- NULL
  Ypath <- NULL
  # get n0 reps from each system
  for (i in 1:k){
    for (j in 1:n0){
     Y[j] <- MySim("sim", i)
    }
    Ybar[i] <- mean(Y)
    Sum2[i] <- (n0-1)*var(Y)
    N[i] <- n0
  }
  x <- which.max(Ybar)
  # start sequential allocation
  while(sum(N) < Nmax){
    xstar <- which.max(Ybar) # current best
    xpath <- c(xpath, x)
    Ypath <- c(Ypath, Ybar[xstar])
  # Npath <- c(Npath, sum(N))
  # calculate scaled gCEIs
    S2 <- Sum2/(N - 1)/N
    for (i in 1:k){
      v <- sqrt(S2[xstar] + S2[i])
      gCEI[i] <- -(1/(2*v))*
          dnorm((Ybar[i] - Ybar[xstar])/v)
    }
    gCEI[xstar] <- 0
    x <- which.min((S2/N)*gCEI)
    if ((S2[xstar]/N[xstar])*sum(gCEI) <=
          (S2[x]/N[x])*gCEI[x]){x <- xstar}
 # simulate x and update statistics
    Yx <- MySim("sim", x)
    difference <- Yx - Ybar[x]
    Ybar[x] <- Ybar[x] + difference/(N[x]+1)
    Sum2[x] <- Sum2[x] + difference*(Yx - Ybar[x])
    N[x] <- N[x] + 1
  }
  list(xstar = xstar, xpath = xpath,
     Ypath = Ypath, N = N, Ybar = Ybar)}
```

Figure 2: `gCEI` procedure, with the code to compute the gradient of CEI and choose the next system to simulate highlighted.

chosen so that this illustration would not take too long to execute. The smaller δ is or the larger the sample variances are, the more replications are required to select the best system because the number of replications is proportional to $S^2(x)/\delta^2$.

We call Rinott's critical value `h` here. In the example it is the 0.95 quantile of a random variable we can simulate, so we use an upper confidence bound on `h` to be conservative. The code for `Rinotth` is included in `Procedures.R`.

5 OPTIMAL ALLOCATION OF A FIXED BUDGET

At the other end of the coordination spectrum from Rinott's procedure are Bayesian or Bayesian-inspired procedures that attempt to sequentially allocate *each individual replication* of a fixed budget of replications as effectively as possible, perhaps even optimally based on some objective. See Chen and Lee (2011), Frazier (2012) and Chen et al. (2015) for overviews.

Recently Chen and Ryzhov (2019) established an important connection between some Bayesian R&S procedures and the static rate-optimal allocation of Glynn and Juneja (2004), where "rate optimal" means that this allocation drives the (frequentist) probability of incorrect selection to 0 at the fastest possible rate. Figure 2 displays one of these procedures, `gCEI` (Avci et al. 2021).

After expending an initial `n0` replications on each system, `gCEI` allocates one replication at a time until a replication budget `Nmax` is exhausted. After each replication the posterior distribution of the system means is updated, the gradient of complete expected improvement (CEI) is computed, and the next replication is allocated to the system that the gradient indicates will decrease CEI the most. CEI is a Bayesian acquisition function due to Salemi et al. (2019) that computes the expected positive gain of each system over the current sample best system, with respect to the posterior distribution of both.

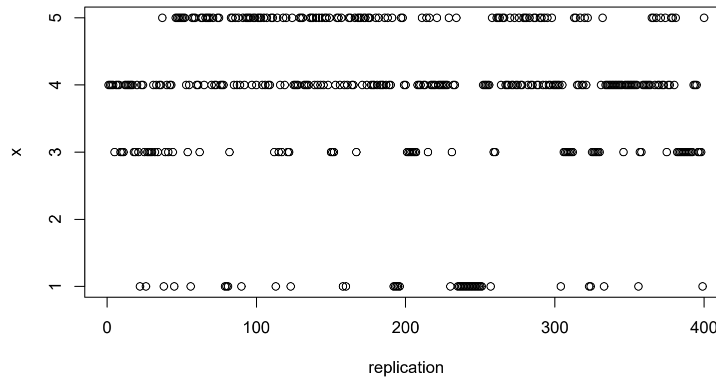

Figure 3: Plot of sequential replication allocation of `gCEI` for the `SAN` problem.

To try `gCEI` on the `SAN` problem with a budget of 500 replications, execute the statements below, but select your own seed.

```
set.seed(12211956)
result <- gcei(20, 500, SAN)
plot(result$xpath, xlab="replication", ylab="x")
result$xstar
[1] 4
result$N
[1]  63  20  91 190 136
```

Notice that system 4 is selected, implying it has the largest sample mean after expending a total of 500 replications. The plot (Figure 3) shows how `gCEI` allocates replications to systems after the initial 20 replicatons for each. System 4 received the most replications (190) which is often the case in the rate-optimal allocation. No confidence statement is associated with this result, but given a limited budget, `gCEI` and similar procedures are highly effective in minimizing selection error.

6 ELIMINATING PROCEDURES

Particularly when the number of systems is large, it often makes sense to try to eliminate clearly inferior systems from further simulation as soon as possible. There are two basic strategies:

Subset & select: Get a small number of replications from all systems, select a subset $\widehat{S} \subseteq \{1,2,\ldots,k\}$ that is as small as possible but still contains the best with high probability, then apply an efficient R&S procedure to the remainder. This usually requires splitting the allowable α error between subset and selection: $\Pr\{x^\star \in \widehat{S}\} \geq 1-\alpha/2$. Of course, subset selection is useful by itself as a screening tool.

Continuous screening: Iteratively replicate, eliminate, replicate, eliminate and so on until one system remains. Such procedures usually exploit pairwise comparisons and control overall error via (say) the Bonferroni inequality. They also need to account for taking "multiple looks" at the simulation output.

The subset selection procedure of Nelson et al. (2001) is shown in Figure 4. This procedure takes an equal number of replications from each system and returns a subset that is guaranteed to contain $x^\star$ with probability $\geq 1-\alpha$ when outputs are normally distributed and systems are simulated independently. To try `Subset` on the `MM1` problem with two different sample sizes execute the statements below, but select your own seed.

```
Subset <- function(alpha, n, MySim){
 # function to do subset selection
 # k = number of systems
 # n = number of replications (equal)
 # 1-alpha = confidence level
 # simulate:
 k <- MySim("k")
 Yall <- NULL
 for (x in 1:k){
  Yall <- cbind(Yall, MySim("sim", x, n))
 }
 # subset selection
 Ybar <- apply(Yall, 2, mean)
 S2 <- apply(Yall, 2, var)/n
 tval <- qt((1-alpha)^(1/(k-1)), df = n-1)
 Subset <- 1:k
 for (i in 1:k){
  for (j in 1:k){
   if (Ybar[i] < (Ybar[j]-
     tval*sqrt(S2[i] + S2[j]))){
    Subset[i] <- 0
    break}}}
 list(Subset = Subset[Subset != 0],
    Ybar = Ybar, S2 = S2)}
```

Figure 4: `Subset` with the elimination step highlighted.

```
set.seed(12211956)
result10 <- Subset(0.05, 10, MM1)
result10$Subset
[1] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
set.seed(12211956)
result100 <- Subset(0.05, 100, MM1)
result100$Subset
[1] 27 28 29 30 31 32 33
```

The correct statement for the 100 replication case is “with 95% confidence the system with the minimum expected cost is one of $\{27,28,29,30,31,32,33\}$.” Of course these indices x correspond to a particular service rate, which in this example is $1+x/5$ customers/time. Not suprisingly, the subset size tends to be smaller when the number of replications is larger; in the best case it contains one system that can be declared to be the best with $1-\alpha$ confidence. However, sequentially increasing the number of replications for `Subset` until one system remains does *not* control the overall error and should be avoided. Nelson et al. (2001) suggest passing the subset to Rinott’s procedure to make the final selection.

Fully sequential simulation and elimination requires a different type of statistical architecture, but algorithmically it is very simular to subset selection. To illustrate we present Paulson’s procedure (Paulson 1964), which assumes independent simulations, normally distributed output, and equal variances. However, `Procedures.R` also contains `KN` (Kim and Nelson 2001) which allows unequal variances and is statistically more efficient than `Paulson`.

`Paulson` requires as input an initial sample size from each system `n0` to estimate variances, and a smallest practically significant difference δ as in `Rinott`. To try `Paulson` on the `Normal` problem execute the statements below, but select your own seed.

```
set.seed(12211956)
result <- Paulson(0.05, 10, 0.1, Normal)
result
$Best
[1] 11
$n
[1] 2035
$Elim
[1]  422  465  607  517  554  817  903  968 1741 2035    0
```

Recall that in this problem we know that $x^\star = 11$, and that the true means are spaced 0.1 apart; thus, the best and second-best are separated by exactly δ. The `$Elim` result shows the number of iterations at which each inferior system was eliminated (0 if not eliminated), which follows how close they are to being

```
Paulson <- function(alpha, n0, delta, MySim){
  # function for Paulson with lambda = delta/2
  # k = number of systems
  # n0 = first-stage sample size
  # 1-alpha = desired PCS
  # delta = indifference-zone parameter
  k <- MySim("k")
  II <- 1:k
  Active <- rep(TRUE, k)
  Elim <- rep(0, k)
  # get n0 from each system and
  # compute pooled variance
  Yn0 <- matrix(0, nrow=k, ncol=n0)
  for (i in 1:k){
    for (j in 1:n0){
      Yn0[i,j] <- MySim("sim", i)}}
  S2 <- mean(apply(Yn0,1,var))
  # start sequential
  a <- eta(alpha, k, n0)*k*(n0-1)*S2/delta
  Ysum <- apply(Yn0, 1, sum)
  r <- n0
  # main elimination loop
  while(sum(Active)> 1){
   r <- r + 1
   ATemp <- Active
   for(i in II[Active]){
    Ysum[i] <- Ysum[i] + MySim("sim", i)}
   for(l in II[Active])
    if((Ysum[l] - max(Ysum[Active]))
        < min(0, -a+delta*r/2)){
      ATemp[l] <- FALSE
      Elim[l] <- r}
   Active <- ATemp}
  list(Best = II[Active], n = r, Elim=Elim)}
```

Figure 5: `Paulson` fully sequential procedure with the elimination step highlighted.

the best. `Paulson` does not have a proven PGS guarantee, only PCS; therefore, the correct statement is "with 95% confidence system 11 is the best, provided it is at least 0.1 units better than the second-best" (which it is in this case). A variation of `Paulson` by Zhong and Hong (2018) does guarantee PGS.

7 COMMON RANDOM NUMBERS

R&S procedures that employ pairwise comparisons can often be "sharpened" by using common random numbers (CRN) because

$$\mathrm{Var}(Y(x)-Y(x')) = \mathrm{Var}(Y(x)) + \mathrm{Var}(Y(x')) - 2\,\mathrm{Cov}(Y(x),Y(x')).$$

CRN tends to make $\mathrm{Cov}(Y(x),Y(x')) > 0$, reducing the variance of the difference. For instance, in the `Invt` simulation the demand does not depend on the system (choice of (s,S)). Therefore, CRN implies that each system sees *exactly the same sequence of demands*, and therefore reacts similarly to it (positive correlation). However, R&S procedures employing CRN usually require equal sample sizes across all systems to insure a favorable effect.

We illustrate the impact of CRN by altering `Subset` to exploit it; see Figure 6. The basis of `Subset` is all pairwise differences. `SubsetCRN` takes a random number seed as input, obtains all `n` replications of each system `x` as a batch, and starts the simulation of each batch with that same seed; see the blue highlight. Notice that in `R` it is faster to compute the variance-covariance matrix of the outputs than to compute variances of all pairwise differences; see the red highlight.

To try `SubsetCRN` on the `MM1` problem with sample size 10, execute the statements below, but select your own seed. Notice that the size of the subset drops from 16 systems to 3 relative to the previous application of `Subset` to independently simulated systems. We display the upper-left portion of the estimated correlation matrix among systems to show the strong positive correlation induced by CRN.

```
resultCRN <- SubsetCRN(0.05, 10, 12211956, MM1)
resultCRN$Subset
[1] 29 30 31
resultCRN$corr[1:3, 1:3]
          [,1]      [,2]      [,3]
[1,] 1.0000000 0.8503456 0.8524638
[2,] 0.8503456 1.0000000 0.9653994
[3,] 0.8524638 0.9653994 1.0000000
```

```
SubsetCRN <- function(alpha, n, seed, MySim){
 # function to do subset selection with CRN
 # k = number of systems
 # n = number of replications (equal)
 # 1-alpha = confidence level
 # seed = random number seed
 # simulate:
 k <- MySim("k")
 Yall <- NULL
 for (x in 1:k){
  Yall <- cbind(Yall, MySim("sim", x, n, seed))
 }
 # subset selection
 Ybar <- apply(Yall, 2, mean)
 S2 <- cov(Yall)/n
 tval <- qt(1-alpha/(k-1), df = n-1)
 Subset <- 1:k
 for (i in 1:k){
  for (j in 1:k){
   if (Ybar[i] < (Ybar[j]-tval*
     sqrt(S2[i,i] + S2[j,j] - 2*S2[i,j]))){
    Subset[i] <- 0
    break}}}
 list(Subset = Subset[Subset != 0],
    Ybar = Ybar, S2 = S2, corr=cor(Yall))}
```

Figure 6: `SubsetCRN` with simulation of each system using CRN highlighted in blue and use of the variance of the difference highlighted in red.

```
bootRS <- function(alpha, n0, delta, B, dn, MySim){
 # procedure to implement bootstrap R&S
 # k = number of systems
 # n0 = first-stage sample size
 # 1-alpha = desired PCS
 # delta = indifference-zone parameter
 # B = number of bootstrap samples to estimate PGS
 # dn = increment to increase n0
 k <- MySim("k")
 PGS <- 0
 Yall <- NULL
 for (x in 1:k){
  Yall <- cbind(Yall, MySim("sim", x, n0))
 }
 # increment n0 until bootstrap PGS >= 1 - alpha
 while(TRUE){
  bsum <- 0
  Ybar <- apply(Yall, 2, mean)
  xstar <- which.max(Ybar)
  for (i in 1:B){
   Ybarstar <-
    apply(apply(Yall, 2, sample, replace=TRUE),
          2, mean)
   diffs <- Ybarstar - Ybarstar[xstar] -
        (Ybar - Ybar[xstar])
   bsum <- bsum + prod(as.numeric(diffs <= delta))}
  PGS <- bsum/B
  print(c("N=", n0, "PGS =",PGS))
  if (PGS < 1 - alpha){
   Ytemp <- NULL
   for (x in 1:k){
    Ytemp <- cbind(Ytemp, MySim("sim", x, dn))}
   Yall <- rbind(Yall, Ytemp)
   n0 <- n0 + dn}
  else{break}}
 list(Best = xstar, PGS=PGS, N = n0)}
```

Figure 7: `bootRS` for expected value with bootstrapping step highlighted.

For subset selection, sharper comparisons means a smaller subset with the same number of replications. For procedures like `Paulson` or `KN` it means reaching a selection with fewer overall replications. `Procedures.R` contains a version of `KN` that exploits CRN.

8 BOOTSTRAP R&S

All of the R&S procedures presented to this point assumed the performance measure was the mean of normally distributed output data. The Holy Grail for R&S is a procedure that works for virtually any performance measure (mean, probability, quantile) and data type (normal, non-normal). Two insights make this possible: If we can construct estimators $\widehat{\theta}(x)$ of (generic) parameters $\theta(x)$ such that

$$\Pr\left\{\widehat{\theta}(x)-\widehat{\theta}(x^\star)-(\theta(x)-\theta(x^\star))\leq\delta,\ \forall x\neq x^\star\right\}\geq 1-\alpha \quad (1)$$

then $\mathrm{PGS}=\Pr\{\theta(x^\star)-\theta(\widehat{x}^\star)\leq\delta\}\geq 1-\alpha$ holds, where $\widehat{x}^\star=\mathrm{argmax}_x\widehat{\theta}(x)$. Further, given replications of output data from each system, we can estimate the probability in (1) using *bootstrapping*, and then increase the number of replications until it is $\geq 1-\alpha$. More specifically, bootstrapping estimates the probabililty that the current sample best system $\widehat{x}^\star$ satisfies (1); see Lee and Nelson (2016). The procedure `bootRS` in Figure 7 implements this approach for selecting the system with the largest expected value.

To try bootRS on the TTF problem with $\delta = 1000$ execute the statements below, but select your own seed. Notice that n0 is rather large at 50; bootRS needs a large initial number of replications to avoid early stopping. Here B is the number of bootstrap samples to estimate PGS (200), and dn is the increment of additional replications to add if the desired PGS has not been achieved (100). A very small increment is not advised as bootstrapping itself takes time, and also to avoid premature stopping. As it executes bootRS displays the estimated PGS to show progress.

```
set.seed(12211956)
resultBoot <- bootRS(0.05, 50, 1000, 200, 100, TTF)
[1] "N="    "50"    "PGS =" "0.5"
[1] "N="    "150"   "PGS =" "0.695"
[1] "N="    "250"   "PGS =" "0.765"
[1] "N="    "350"   "PGS =" "0.78"
[1] "N="    "450"   "PGS =" "0.855"
[1] "N="    "550"   "PGS =" "0.915"
[1] "N="    "650"   "PGS =" "0.955"
resultBoot$Best
[1] 2
```

Notice that the total sample size of 4×650 is comparable to our previous application of Rinott, but Rinott assumed normally distributed output while bootRS does not.

9 PARALLEL R&S

Parallel computing is an enormous asset for R&S. See Hunter and Nelson (2017) for an overview, and Ni et al. (2017), Luo et al. (2015), Pei et al. (2020), Pei et al. (2022) and Zhong and Hong (2022) for recent algorithms. Here we consider the simplest version of parallelization that can be done easily on a personal computer with multiple cores and threads. For this purpose we will use the doParallel library for R; the key commands are listed below.

```
# Packages needed for parallel computation
# library(doParallel)
# library(foreach) #<-- should be installed by doParallel
# library(parallel) #<-- should be installed by doParallel
# Useful commands
# detectCores() #<-- number of available workers
# cl <- makeCluster(4) #<-- make a cluster of 4 workers
# registerDoParallel(cl) #<-- register the cluster
# ptime <- system.time({ CODE HERE })[3] #<-- a wrapper for timing code
# getDoParWorkers() #<-- verify number of parallel workers doParallel will use
# stopCluster(cl) #<-- obvious
```

After loading the doParallel library, execute detectCores() to find out how large a parallel worker cluster you can recruit on your machine, and then make and register the cluster with something less than all of them.

Parallel R&S introduces new statistical and computational issues; we focus on the computational ones here and recommend Hunter and Nelson (2017) and Luo et al. (2015) for discussions of the statistical issues. Hunter and Nelson (2017) view a R&S procedure as a collection of “jobs” that must be executed by a worker. A job may consist of (a) a list of systems to simulate, how many replications to obtain and what random numbers to use for each; (b) a list of non-simulation calculations and the other jobs that must complete before these calculations can be undertaken; or (c) both simulations and calculations. Simulation and calculation jobs may be completed in parallel, but calculation jobs must wait for the other required jobs to complete, and it is this coupling that slows down parallel R&S procedures.

We will test a simple example, `SubsetParallel`, which parallelizes the simulations of each system in `SubsetCRN`. Recall that `SubsetCRN` obtains n replications from each system using CRN, then does pairwise comparisons. Thus, it can be viewed as k simulation jobs followed by one calculation job that depends on all of the simulation results being completed. Specifically, `SubsetCRN` employs the following loop to execute the k simulation jobs and to build an $n \times k$ matrix `Yall` in which each column corresponds to a system:

```
for (x in 1:k){
Yall <- cbind(Yall, MySim("sim", x, n, seed))
}
```

`SubsetParallel` replaces this serial loop with a parallelized loop that builds, say, 6 columns of replications at a time if the number of workers in the cluster is 6:

```
Yall <- foreach(x=1:k, .combine=cbind) %dopar% {MySim("sim", x, n, seed)}
```

To try `SubsetParallel` and compare it to `SubsetCRN` for the `Invt` example, execute the statements below (your cluster may be larger or smaller depending on `detectCores()`). The `[3]` element returned by `system.time` is the wall-clock time consumed in seconds; notice that the parallel version is about 50% faster on this 1600 system problem.

```
cl <- makeCluster(6)
registerDoParallel(cl)
ptimeSC <- system.time({resultSC <- SubsetCRN(0.05, 100, 12211956, Invt)})[3]
ptimeSP <- system.time({resultSP <- SubsetParallel(0.05, 100, 12211956, Invt)})[3]
ptimeSC
  elapsed
   8.17
ptimeSP
  elapsed
   3.97
```

Timings are not perfectly repeatable, even with the same code, due to dependence on what else is happening on your computer. However, if you check `resultSC$Subset` and `resultSP$Subset` you will see they are identical; `SubsetParallel` is simply `SubsetCRN` with parallel simulations.

Parallelizing R&S procedures can be tricky: unexpected behavior can occur, and the particular computer architecture and operating system matters. As a very high-level statement, the savings from doing simulations and computations in parallel has to overcome any computing overhead (e.g., message passing) and coordination bottlenecks (e.g., idling workers while waiting for computations to complete). Pei et al. (2020) discuss these issues in a message passing environment by comparing parallel subset selection, GSP (Ni et al. 2017) and bi-PASS (described below).

Bisection Parallel Adaptive Survivor Selection (bi-PASS) is a state-of-the-art parallel R&S algorithm created for problems with thousands to millions of systems. bi-PASS eliminates sytems by comparing them to an estimate of the mean of the best system that is learned collectively, avoiding pairwise comparisons, and it controls the expected false elimination rate (EFER) for systems as good as the best. bi-PASS can be executed as a fixed-budget procedure, as we do here, in which case it typically returns a subset of systems.

Two versions of bi-PASS are included in `Procedures.R`: `bipassSlow` which has no parallel simulations, and `bipassFast` which does (see Figure 8). The only problems for which bi-PASS makes any sense are `Invt` (1600 systems) and `MM1` (100 systems), although neither is large enough to get the full benefit. bi-PASS has solved million system problems in as little as 12 minutes. One would expect the most benefit on the large `Invt` problem, but in fact a greater speed up is obtained for `MM1`. Why? For `Invt` the simulations are very fast, and thus the computational overhead of parallelizing the simulations is more than what is saved. For `MM1`, on the other hand, the simulations are much slower so there is a benefit of executing batches of them in parallel.

```
bipassFast <- function(c, n0, dn, Nmax, MySim){
  # synchronized biPASS with pooled variance
  # k = number of systems
  # c = constant needed to guarantee EFER
  # n0 = first-stage sample size
  # dn = batch size per run
  # Nmax = maximum number of replications
  MySim <- MySim # make the simulation local
  k <- MySim("k")
  g <- function(t, calpha=c)
       {sqrt((calpha + log(t+1))*(t+1))}
  II <- 1:k
  Active <- rep(TRUE, k)  # systems not eliminated
  Elim <- rep(0, k)  # rep when elimination occurs
  # get n0 reps and compute pooled variance
  Yn0 <- foreach(i=1:k, .combine=rbind)
   %dopar% {MySim("sim", i, n0)}
  S2 <- mean(apply(Yn0,1,var))
  # start sequential elimination
  Ysum <- apply(Yn0, 1, sum)
  r <- n0
  N <- n0*k
  # main elimination loop
  while(sum(Active) > 1 && N < Nmax){
    r <- r + dn
    N <- N + dn*sum(Active)
    Ynew <- foreach(i=II[Active], .combine=rbind)
     %dopar% {MySim("sim", i, dn)}
    Ysum[Active] <- Ysum[Active] +
                      apply(Ynew, 1, sum)
    rmuhat <- sum(Ysum[Active])/sum(Active)
    for(l in II[Active]){
      if(Ysum[l] - rmuhat <= -g(r/S2)*S2){
        Active[l] <- FALSE
        Elim[l] <- r}}
  }
  list(Best = II[Active], n = r, Elim=Elim,
       Means = Ysum[Active]/r)}
```

Figure 8: bi-PASS with parallelized portions highlighted in blue. The estimated mean of the best system, times the current number of replications, is `rmuhat`.

To try bi-PASS on `MM1` execute the commands below. Notice that the critical value `c` for bi-PASS does not depend on the number of systems being compared, only the first-stage sample size `n0` because bi-PASS controls the EFER, not PCS or PGS. Parallized bi-PASS is roughly 50% faster here. Next try the same experiment on `Invt`.

```
cl <- makeCluster(6)
registerDoParallel(cl)
set.seed(12211956)
ptimeBS <- system.time({resultBS <- bipassSlow(5, 10, 20, 15000, MM1)})[3]
set.seed(12211956)
ptimeBF <- system.time({resultBF <- bipassFast(5, 10, 20, 15000, MM1)})[3]
ptimeBS
elapsed
  53.39
ptimeBF
elapsed
  25.36
```

ACKNOWLEDGEMENTS

This work was partially supported by National Science Foundation Grant No. DMS-1854562.

REFERENCES

Avci, H., B. L. Nelson, and A. Wächter. 2021. "Getting to "Rate-Optimal" in Ranking & Selection". In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chen, C.-H., S. E. Chick, L. H. Lee, and N. A. Pujowidianto. 2015. "Ranking and Selection: Efficient Simulation Budget Allocation". In *Handbook of Simulation Optimization*, edited by M. Fu, 45–80. New York: Springer.

Chen, C.-H., and L. H. Lee. 2011. *Stochastic Simulation Optimization: An Optimal Computing Budget Allocation*. Singapore: World Scientific.

Chen, Y., and I. O. Ryzhov. 2019. "Complete Expected Improvement Converges to an Optimal Budget Allocation". *Advances in Applied Probability* 51(1):209–235.

Eckman, D. J., and S. G. Henderson. 2018. “Guarantees on the Probability of Good Selection”. In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 351–365. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Frazier, P. 2012. “Tutorial: Optimization via Simulation with Bayesian Statistics and Dynamic Programming”. In *Proceedings of the 2012 Winter Simulation Conference*, edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, 79–94. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Glynn, P., and S. Juneja. 2004. “A Large Deviations Perspective on Ordinal Optimization”. In *Proceedings of the 2004 Winter Simulation Conference*, edited by R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 577–585. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hong, L. J., W. Fan, and J. Luo. 2021. “Review on Ranking and Selection: A New Perspective”. *Frontiers of Engineering Management* 8(3):321–343.

Hunter, S. R., and B. L. Nelson. 2017. “Parallel Ranking and Selection”. In *Advances in Modeling and Simulation*, edited by A. Tolk, J. Fowler, G. Shao, and E. Yücesan, 249–275. New York: Springer.

Kim, S.-H., and B. L. Nelson. 2001. “A Fully Sequential Procedure for Indifference-zone Selection in Simulation”. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 11(3):251–273.

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

Lee, S., and B. L. Nelson. 2016. “General-purpose Ranking and Selection for Computer Simulation”. *IIE Transactions* 48(6):555–564.

Luo, J., L. J. Hong, B. L. Nelson, and Y. Wu. 2015. “Fully Sequential Procedures for Large-scale Ranking-and-Selection Problems in Parallel Computing Environments”. *Operations Research* 63(5):1177–1194.

Matejcik, F. J., and B. L. Nelson. 1995. “Two-stage Multiple Comparisons with the Best for Computer Simulation”. *Operations Research* 43(4):633–640.

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

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

Paulson, E. 1964. “A Sequential Procedure for Selecting the Population with the Largest Mean from k Normal Populations”. *The Annals of Mathematical Statistics* 35:174–180.

Pei, L., B. L. Nelson, and S. R. Hunter. 2020. “Evaluation of bi-PASS for Parallel Simulation Optimization”. In *Proceedings of the 2020 Winter Simulation Conference*, edited by K. G. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and T. R., 2960–2971. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pei, L., B. L. Nelson, and S. R. Hunter. 2022. “Parallel Adaptive Survivor Selection”. *Operations Research*. Forthcoming.

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

Salemi, P., E. Song, B. L. Nelson, and J. Staum. 2019. “Gaussian Markov Random Fields for Discrete Optimization via Simulation: Framework and Algorithms”. *Operations Research* 67(1):250–266.

Zhong, Y., and L. J. Hong. 2018. “Fully Sequential Ranking and Selection Procedures with PAC Guarantee”. In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 1898–1908. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Zhong, Y., and L. J. Hong. 2022. “Knockout-tournament Procedures for Large-scale Ranking and Selection in Parallel Computing Environments”. *Operations Research* 70:432–453.

AUTHOR BIOGRAPHY

BARRY L. NELSON is the Walter P. Murphy Professor in the Department of Industrial Engineering and Management Sciences at Northwestern University. He is a Fellow of INFORMS and IISE. His research centers on the design and analysis of computer simulation experiments on models of stochastic systems. His e-mail address is nelsonb@northwestern.edu.