Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

PARALLEL EMPIRICAL STOCHASTIC BRANCH AND BOUND FOR LARGE-SCALE DISCRETE OPTIMIZATION VIA SIMULATION

Scott Rosen
Peter Salemi
Brian Wickham
Ashley Williams
Christine Harvey
Erin Catlett

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102, USA

Sajjad Taghiyeh
Jie Xu

Department of Systems Engineering and Operations Research
George Mason University
4400 University Drive
Fairfax, VA 22030, USA

ABSTRACT

Real-life simulation optimization applications often deal with large-scale simulation models that are time-consuming to execute. Parallel computing environments, such as high performance computing clusters and cloud computing services, provide the computing power needed to scale to such applications. In this paper, we show how the Empirical Stochastic Branch and Bound algorithm, an effective globally convergent random search algorithm for discrete optimization via simulation, can be adapted to a high-performance computing environment to effectively utilize the power of parallelism. We propose a master-worker structure driven by MITRE's Goal-Directed Grid-Enabled Simulation Experimentation Environment. Numerical experiments with the popular Ackley benchmark test function and a real-world simulation called *runway*Simulator demonstrate the number of cores needed to achieve a good scaled efficiency of parallel empirical stochastic branch and bound for increasing levels of simulation run times.

1 INTRODUCTION

In this paper, we introduce an approach to implement the simulation optimization algorithm called Empirical Stochastic Branch and Bound (ESBB) (Xu and Nelson 2013) in a parallel environment. ESBB is a state of the art algorithm for discrete optimization via simulation problems which combines the stochastic branch and bound method with empirical bound estimates. The details of this method will be discussed in Section 2. Large-scale simulations are often very time consuming to run, which can increase the runtime of the ESBB algorithm. The specific partitioning structure of ESBB, in which the majority of the steps in the algorithm can be completed independently, lends itself nicely to the use of multi-core clusters. By implementing the ESBB algorithm in a parallel environment, we can reach an acceptable speedup, up to 100 times faster than in a sequential environment, as we show in the Numerical Experiments and Results section.

The parallel version of some simulation optimization problems have been investigated before. In Bak et al. (2011), the authors use a bottom-up approach for solving the "Rectangular Guillotine Strip Cutting Problem" in which the parallelization is based on creating subsequent intermediate meta-rectangles that can be processed independently of each other by each of the available processors. A parallel version of the branch and bound algorithm for "Multidimensional Scaling with City-block Distances" has been proposed and investigated in Zilinkas (2012). In their procedure, each processor runs the same algorithm generating and branching partial solutions up to some level. They also use a load balancing strategy in which they assign every n^{th} job (n being the number of processors) to each processor. A parallel branch and bound

algorithm without communication was implemented in Laursen (1994), and it was concluded that branch and bound without communication is not inherently inefficient.

Luo et al. (2015) investigated both implementation and statistical issues which may arise while making large-scale ranking and selection procedures work in a parallel environment. They proposed two algorithms to tackle these issues. The first algorithm, called a "Vector Filling Procedure," is a synchronous master-worker approach which creates a vector to record the observations in the same order of the input sequence, requiring a large amount of memory to store simulation runs. This introduces an asynchronous master-worker approach that is asymptotically valid and computationally more efficient. In their approach, screening was limited to the master processor limiting their ability to efficiently scale with additional processors. Ni, Hunter, and Henderson (2013) distributed the screening procedure along the worker processors and used an initial stage of sampling to reduce the communication requirements to balance the workload. With these modifications, they were able to apply the ranking and selection procedure on approximately 1 million systems which was traditionally applied to only 20 systems in the sequential version. These studies show that using an efficient parallel algorithm can achieve considerable speedup compared to sequential versions of algorithms.

Several parallel strategies are used in literature to enhance the performance of sequential algorithms (Ni, Hunter, and Henderson 2013; Kishimoto, Fukunaga, and Botea 2013; Buluk and Madduri 2011; Xia and Prasana 2011; Caromel et al. 2007). The majority of them use a master-worker structure to make the algorithms parallel. Ni, Hunter, and Henderson (2013) introduces a general partitioning framework in a parallel environment which starts with a root processor and generates seed nodes using a sequential algorithm. It then assigns those seeds among the available processors. These seed nodes become the local root for a local sequential search algorithm. The authors also compare the centralized approach, in which everything should be communicated to the master processor, to decentralized approaches that distribute work among workers. A load balancing technique is used in Buluk and Madduri (2011), which dynamically adjusts the number of threads according to the estimated scalability. In order to attain scalability, a multi-layer master-worker structure is introduced in Xia and Prasana (2011), which is composed of four kinds of entities (master, sub-master, worker, and leader), where the task of the master is distributed among sub-masters in order to overcome the communication overhead and achieve a better speedup.

In this paper, a parallel version of the ESBB algorithm is implemented using a master-worker and scheduling structure implemented on MITRE's Elastic Goal-Directed Simulation Framework (Page et. al. 2012) that consists of minimal communication overhead. We demonstrate that a modest master-slave structure can be highly effective in improving efficiency and speedup of the ESBB due to the structure of the ESBB algorithm. The rest of paper is organized as follows. In Section 2, we formally describe the simulation optimization problem and the sequential ESBB algorithm. In Section 3, we present the details of both the synchronous and asynchronous approaches to make the ESBB algorithm parallel. We then describe numerical experiments used to compare the performance of sequential ESBB to synchronous parallel ESBB in Section 4. Section 5 compares the performance of the algorithm in both sequential and parallel environments. Conclusions are then drawn in Section 6.

2 EMPIRICAL STOCHASTIC BRANCH AND BOUND (ESBB)

The ESBB algorithm was proposed by Xu and Nelson (2013) as an algorithm for simulation optimization. In their approach, the partitioning structure of stochastic branch and bound is combined with an empirical performance evaluation of each sub-region. ESBB also uses the nested partitions (Shi and Olafsson 2000) bounding estimation using the objective function values of the solutions simulated so far. ESBB iteratively partitions the solution space into sub-regions using the partitioning scheme and estimates the lower and upper bound on the objective function for the sub-regions. Finally, it selects as the record set the sub-region with the largest upper bound (or lower bound depending on the optimization problem). The global convergence of ESBB is proved in Xu and Nelson (2013) for a finite number of feasible solutions. The details of the ESBB algorithm are as follows.

The goal of ESSB is to find x that solves Equation (1).

$$max_{x \in X} \ \mu(x) \tag{1}$$

Define the region X' as the intersection of the n-dimensional integer lattice Z^n and the n-dimensional hypercube H^n defined by the constraints

$$l_i \leq x_i \leq u_i, i = 1,2, \dots, n, \ \ where \ \ l_i, x_i, u_i \in Z \tag{2}$$

Also define D as the sub-region in H^n for which the following inequalities hold:

$$D = x \in R^n : g_j(x) \leq 0, j = 1,2, \dots, p. \tag{3}$$

The solution space X is defined as $X = X' \cap D$. It is assumed that D is convex and only a finite number of solutions exist. In this paper, the stochastic problem with $\mu(x) = E[Y(x)]$ is considered and the value of $\mu(x)$ is estimated via simulation using the simulation output, which can be interpreted as observations of the random variable $Y(x)$. We also assume that the stochastic noise is independent and identically distributed. The term *sample* is used whenever we randomly select a solution in the solution space and the term *simulate* is used when an observation of $Y(x)$ is obtained.

2.1 ESBB algorithm

First we define the notation used in the ESBB algorithm. We use the same notations as what already exists in Xu and Nelson (2013).

Let k denote the iteration counter. At each iteration, the sampled solutions are recorded in the set S^k. Also, the set θ^k contains all of the solutions that have been sampled through iteration k. The current partition structure is denoted by P_k and the most promising sub-region at iteration k, called the record set, is denoted by R^k. The ESBB algorithm consists of the following steps, described in Xu and Nelson (2013):

- **Initialization:** Set $k = 0$, $\theta^0 = \emptyset$, $P_0 = X$ and $R^0 = X$.
- **Step 1.** *Partitioning*: If R^k is a singleton, then set $P'_k = P_k$ and go to step 2. Else, partition the record set according to the partitioning scheme and store it in the set $P_k{}''(R^k)$. The new full partition is defined as $P'(k) = (P_k \backslash R^k) \cup P''(R^k)$. Elements of P'_k will also be denoted by X^P.
- **Step 2-1.** *Solution Sampling*: For each sub-region in the partitioned record set, randomly sample v_R solutions. If $k > 0$, for sub-regions which are not in the record set, sample $\theta(X^P)$ solutions that depends on the formation of the sampled solution space in the previous iteration (θ^{k-1}). Record all of the sampled solutions in the set S^k.
- **Step 2-2.** *Bound estimation*: For each sampled solution in S^{k-1}, sample Δn_F observations if it has not been sampled before and sample Δn_A additional observations if it exists in θ^{k-1}. Calculate the upper-bound and lower-bound estimates $U(X^P)$ and $L(X^P)$ of each sub-region according to the sampled solutions.
- **Step 2-3.** *Sample allocation*: Compute the number of solutions that should be sampled in the next iteration from each sub-region $\theta(X^P)$ based on information in the current set of sampled solutions (θ^k).
- **Step 3.** *Updating partitions and the record set*: Find the record set for the next iteration $R^{k+1} = arg\ max_{x^p \in p'_k} \eta^{k+1}(X^P)$ and go to Step 1.

If we define $\overline{Y}(x)$ as the sample average of all solutions in each sub-region, the upper-bound and lower-bound for each sub-region is calculated as follows:

$$U(X^P) = L(X^P) = max_{x \in X^P \cap \theta^k} \overline{Y}(x). \quad (4)$$

The algorithm usually terminates when the simulation budget is reached. At termination, the solution with the maximum cumulative average will be selected as the best solution. In Xu and Nelson (2013), the global convergence of this algorithm for problems with a finite number of feasible solutions is proven and several approaches to calculate the potential of each sub-region $\theta(X^P)$ are proposed. For simplicity, since our goal is decreasing the algorithm runtime, we assign the same sampling scheme for all non-record set partitions, meaning we do not calculate the value of θ at each iteration.

3 PARALLEL EMPIRICAL STOCHASTIC BRANCH AND BOUND

In this section, we propose a scalable synchronous parallel framework in which, without loss of convergence, we can divide the workload among multiple cores with a small amount of communication overhead and ad hoc job assignments geared at keeping all processors continuously busy. This framework is a natural extension of MITRE's Elastic Goal-Directed Simulation Framework (MEG) driven by the Ouroboros web service, which is described below. The framework is compatible with any simulation capable of interfacing with an input file and an output file.

3.1 Goal-Directed Grid-Enabled Simulation Experimentation Environment

MITRE initiated the development of the MITRE Elastic Goal-Directed Simulation Framework (MEG) (Page et.al. 2012) after motivation from the success of the High Level Architecture (HLA) as an enabler for legacy simulations and by recent advances in grid-enabled simulation. The objective of MEG is to provide a middleware framework that achieves parallel and distributed simulation through grid-based replication management with access to substantial computing power.

The MEG web services architecture, shown in Figure 1, is primarily developed in JAVA. However, MEG's web services architecture is an extensible framework and is not tied to JAVA for development of new services. Users can develop add-on services in a language of their choice as long as they implement the service with the predefined Web Service Definition Languages (WSDLs). The standard API definition for each of the services allows us to rapidly deploy new services and/or reconfigure existing services for use within MEG.

The experiment service is responsible for controlling the configuration, monitoring, and persistence of data during a MEG execution. During a run, the experiment service initializes and configures the distributed MEG services for a MEG execution, starts an execution, and monitors all jobs that are launched for an execution.

The Ouroboros web service is responsible for executing the optimization algorithm that creates jobs and evaluates the job results as they return to determine which jobs to launch next. Developers have the ability to add new optimization algorithms by implementing a standalone service with the predefined WSDLs. The plug-in interface described above was employed for this work to integrate the PESBB algorithm into the described extensible MEG web services architecture. The ESBB algorithm is written in C++ and a combination of gSOAP and libcurl were used to interface with the JAVA based MEG web services and transform the ESBB algorithm into the PESBB described in the following section.

The MEG scheduler services [Job Manager Service/Resource Manager Delegate Service (RMDS)] were designed to accept jobs from and return job results to the Ouroboros service. The job execution occurs in a runtime environment such as a multicore laptop/workstation, a grid enabled cluster managed by a local resource manager scheduling software [such as Condor, Sun Grid Engine (SGE), or Portable Batch System (PBS)], and/or in the cloud. Users have the ability to add new execution environments if desired by implementing a scheduling service for the defined WSDLs.

3.2 Parallel ESBB Implementation Framework in MEG

In this section, we propose a parallel framework for implementing ESBB (PESBB) in MEG requiring a minimal amount of communication overhead and balancing. Our framework consists of only two entity types, a master node and worker nodes, and maintains efficiency through the Ouroboros web service as described above. The nature of the ESBB algorithm allows for a simple communication structure to be highly effective as shown in the case study in the following section. Details behind the tasks for each entity and how the algorithm steps are implemented are provided below:

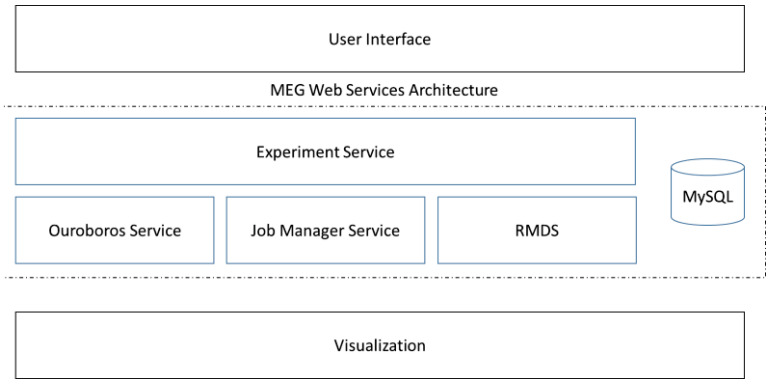

Figure 1: The MEG Web Services Architecture.

3.3 Master Processor

The master processor is in charge of managing the execution of the entire algorithm and partitioning the record set R^k into subsets R_1^k, R_2^k, ..., R_m^k (where m is the number of sub-master processors). Then it assigns sampling jobs in each subset R_1^k, R_2^k, ..., R_m^k to the worker node. The master processor also determines the record set for the next iteration and at the end of each iteration. The master processor is also in charge of the termination of the PESBB algorithm and reporting the final solution.

3.4 Worker processors

At the worker level, each worker receives a partitioned subset determined by the master. Then, it samples and simulates solutions within the subset and sends back the simulation results to the master. The number of workers is fixed and the partitioning scheme is not assigned to match the worker number directly. The number of partition sets should at least equal the number of workers. If it is greater than the number of workers, multiple parallel job launchings are performed to obtain all of the sampling for the partitions at that iteration of the algorithm.

3.5 PESBB Algorithm Implementation on Master and Worker Nodes

We start with one master node and declare the other cores to be worker nodes. Let Q_1, Q_2, ..., Q_m denote the portion of solution space stored by the master. Other notations are the same as what was mentioned in Section 2. The synchronous PESBB algorithm is as follows:

- **Initialization:** Set $k = 0$, $\theta^0 = \emptyset$, $P_0 = X$, $R^0 = X$, and $Q_1 = Q_2 = \cdots = Q_m = \emptyset$.
- **Step 1.** The master processor divides R^k into m sub-regions R_1^k, R_2^k, ..., R_m^k and further partitions R_i^k into w sub-regions R_{i1}^k, R_{i2}^k,..., R_{iw}^k. The master processor then assigns each partition to each available worker node until all workers are utilized. Update Q_1, Q_2, ..., Q_m according to the assigned portion of the solution space.

- **Step 2.** Workers start sampling and simulating solutions on the given parts of the solution space and report back the sampled solutions and simulation results to the master.
- **Step 3.** The non-record set sampling is then assigned to the workers as they become available and these simulation results are reported back to the master after they are performed.
- **Step 4.** The master receives the results from each worker node and updates θ^{k+1} after last corresponding worker finishes the job. Then the master calculates the upper-bound (lower-bounds) of the assigned solution space and records it.
- **Step 5.** The master processor then compares the received upper-bounds (lower-bounds) and records the partition corresponding to the best upper bound. This will be R^{k+1} and will be removed from the sub-master. Let $k = k + 1$ and go to Step 1.

4 NUMERICAL EXPERIMENTS AND RESULTS

Experimentation was performed on the PESSB algorithm to measure the potential improvements in solution time through implementation in a parallel computing environment. The number of cores in the parallel computing environment were varied throughout the experimentation to assess efficiency and performance improvements as a function of the number of cores. The first phase of experimentation was performed on the Ackley test function. This allows control for the expected run time of the model as well as the variation in run times across replications. Here, the effect of the model run times with respect to the clock time until convergence can be measured and an efficiency curve can be derived.

4.1 Ackley Testing Function

The Ackley test function has the following form (Tang et al. 2009) :

$$F_{Ackley}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e, \tag{5}$$

where d is the dimension of the solution space and x is a vector of dimension d. To put this into the form of a simulation optimization problem, Gaussian noise with a mean of zero and standard deviation of 0.3 is added to the function output. Also, we add sleep time in order to make the function evaluation time-consuming, since many simulations have long run times per each replication. We also test the impact of random variations in function evaluation times on the scalability of the algorithm. To do so, we make the sleep time follow a Gaussian distribution with mean τ and standard deviation $c\tau$. When we set $c = 0$, the function has deterministic evaluation times.

Table 1 provides a plausible experimental design to test the effect of sleep time on the convergence time and efficiency of the PESSB algorithm in a parallel computing environment. Sleep time is varied between 0.0001 minutes to 1 minute. In addition, the number of CPUs working in parallel to execute the PESBB algorithm are also varied between 1 and 128. The variance in sleep times are initially set to zero. Next, we will augment the experimental design to investigate the effect of sleep time variance.

Table 1: Experimental design with varying sleep-time $t \sim N(\tau, (c\tau)^2)$ added to the Ackley test function.

	$\boldsymbol{\tau = 0.0001}$	$\boldsymbol{\tau = 0.01}$	$\boldsymbol{\tau = 0.1}$	$\boldsymbol{\tau = 1}$
CPUs = 1	*Run 1*	*Run 5*	*Run 9*	*Run 13*
CPUs = 32	*Run 2*	*Run 6*	*Run 10*	*Run 14*
CPUs = 64	*Run 3*	*Run 7*	*Run 11*	*Run 15*
CPUs = 128	*Run 4*	*Run 8*	*Run 12*	*Run 16*

Results are presented through two metrics: duration of clock time until convergence and efficiency. Efficiency is a relative measure of the convergence time scaled by the number of CPUs. More specifically, it is the proportion of reduced convergence time with respect to the convergence time of one CPU divided by the number of CPUs utilized to achieve that reduction. These metrics are plotted in Figures 2-3. The two

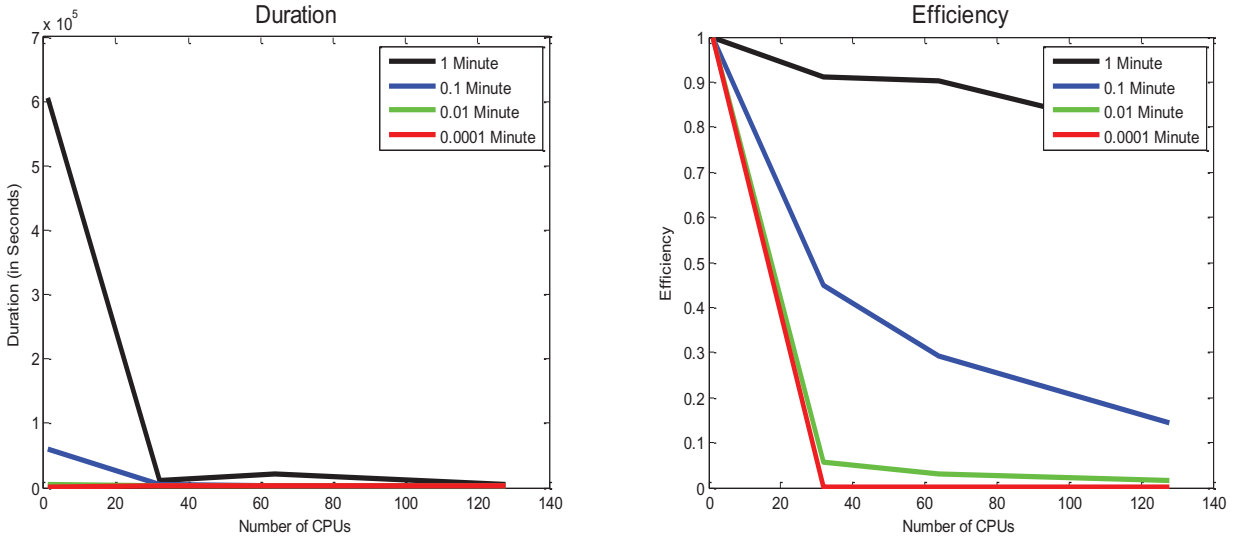

Figure 2: Duration and efficiency as a function of the number of CPUs.

plots in Figure 2 show duration (in seconds) and efficiency (speedup/# of CPU's, where speedup is the number of times faster than with one CPU) as the number of CPUs used for parallelization increases.

In terms of speedup, Figure 3 shows that the benefits of PESBB are greatest for longer durations. As the sleep time decreases, the benefit of using parallelization also decreases, since the simulation is much faster to run and thus parallelization is not necessary. This is due to the fact that, for smaller sleep times, the cost of parallelization may actually inhibit the run time of the algorithm due to time needed for scheduling CPUs throughout the algorithm run; this behavior is well known and expected. However, as we can see from the duration plot in Figure 2, the run time of the algorithm for longer sleep times is reduced to the run time of the algorithm for very short sleep times when the number of CPUs is increased beyond 32 CPUs. The efficiency curve in Figure 2 shows us that the benefits, in terms of efficiency are greatest for longer sleep times, and decrease as the sleep time decreases. If the feasible region was sufficiently large, even for smaller sleep times, we can still benefit from parallelization.

The PESBB algorithm was also tested on the Ackley test function while applying a standard deviation of 20 seconds along with a mean sleep time of one minute. The results for these experiments are given in Table 2. We observed in this additional experiment that there was little deviation in terms of speedup, efficiency, and duration. The performance in terms of these metrics was slightly less, but variability in the run time of the simulation did not appear to have a significant effect on these metrics.

Table 2: Comparison of duration, efficiency, and speedup between a sleep time standard deviation of 0 seconds and 20 seconds under a mean sleep time of 1 minute.

	Duration (Seconds)	**Efficiency**	**Speedup**
$\boldsymbol{\sigma = 0}$ **seconds**	20276.20	0.9326	29.8429
$\boldsymbol{\sigma = 20}$ **seconds**	20743.80	0.9116	29.1702

4.2 Large Scale Simulation: *runway*Simulator

The next phase of the experimentation involves executing the PESBB implementation on a real-world large-scale simulation. For this study, a simulation called *runway*Simulator is used. *runway*Simulator (Kuzminski 2013) is a medium resolution simulation model that enables the analysis of airport capacity for

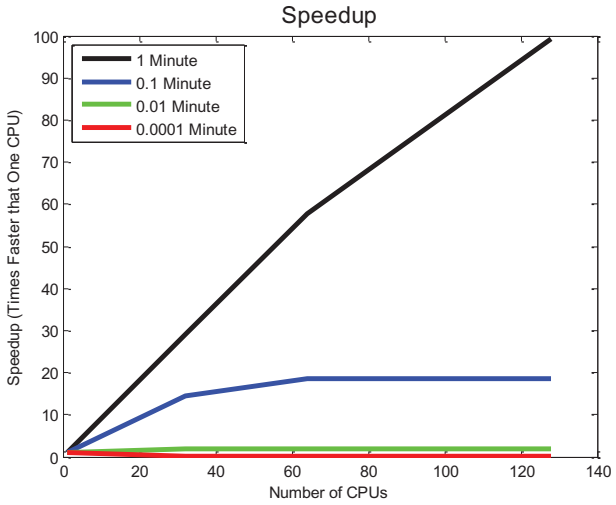

Figure 3: Speedup as a function of the number of CPUs.

operational concepts beyond the reach of standard analytical models like the Federal Aviation Administration's (FAA's) Airfield Capacity Model. MITRE has used *runway*Simulator as its primary tool for airport capacity estimation of baseline capacities at major airports, to parameterize simulations for system-wide analyses, to assess the effects of operational changes (e.g., wake class re-categorization, arrival-departure windows), and to study alternative runway improvement and closure projects at various airports.

The simulation optimization problem involving *runway*Simulator pertains to the simulated airport having two or more runways, r_i, each of which can service arriving or departing flights based on the type of aircraft, a_j, and the target proportion, p_{ij}, of that aircraft type on that runway (the "eligibility policy"). Airport capacity, C, strongly depends on the runway eligibility policies for each aircraft type. The objective is to obtain the set of eligibility policies, $P = p_{ij}\ \forall i, j$ that maximizes airport capacity in each of three "operating regimes": arrival priority, departure priority, and balanced. In the simulation, each eligibility policy, when expressed as a percentage of the aircraft type's eligibility to each runway, can only be set in increments of 5%. That is, $p_{ij} \in 0 \leq N \times 5 \leq 100$, where N is the set of natural numbers.

The PESBB experimentation on *runway*Simulator was performed in high performance computing environments consisting of varying numbers of CPUs. The algorithm was implemented in a 32 node parallel environment, a 64 node environment, and a 128 node environment. The algorithm was configured to run for 4000 iterations for each of these scenarios and did not have any other termination criteria. An individual replication of runway simulator consumes about 12 minutes of clock time. For a single CPU environment, it was projected that the convergence would take roughly 485 hours or approximately 20 days of time.

The duration, efficiency, and speedup performance measures for the four scenario implementations are provided in the plots in Figure 4. For duration, we see a tremendous improvement from 1 CPU to 32 CPUs. However, as the number of CPUs is increased above 32, the rate of improvement significantly decreases. Both of these trends are similar to what is observed above for the Ackley function. Efficiency, however, remains consistent when increasing from 1 to 32 CPUs. However, it decreases to 0.95 when increasing the number of CPUs to 64, and then decreases at a higher rate to 0.82 when increasing the number of CPUs to

100. The efficiency curve for the Ackley function consisted of a very different shape and included a drop-off when going from 1 to 32 CPUs. In terms of speedup, there is a linear increase from 1 to 64 CPUs, but the rate of increase decreases when the number of CPUs is increased. This trend for speedup is also very consistent to what was observed for the Ackley test function.

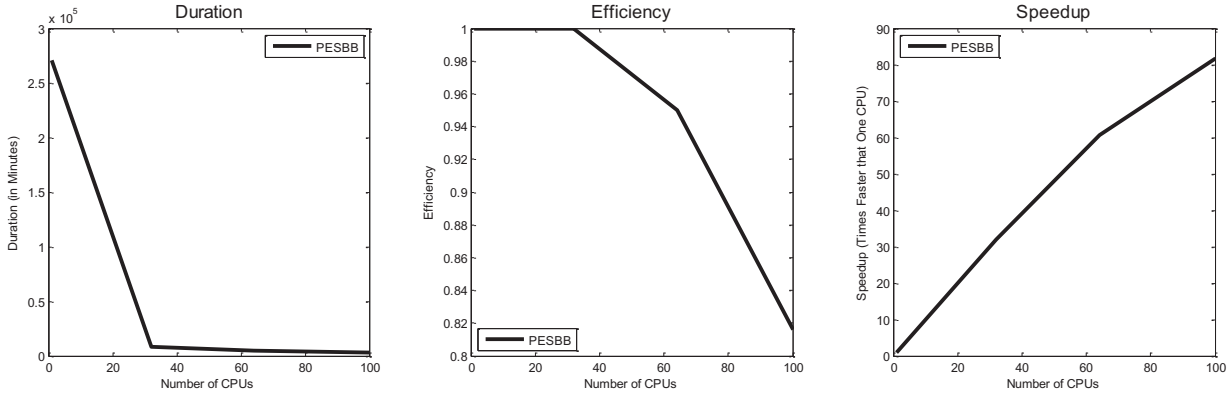

Figure 4: Duration, efficiency, and speedup of PESBB applied to *runway*Simulator across an increasing number of CPUs.

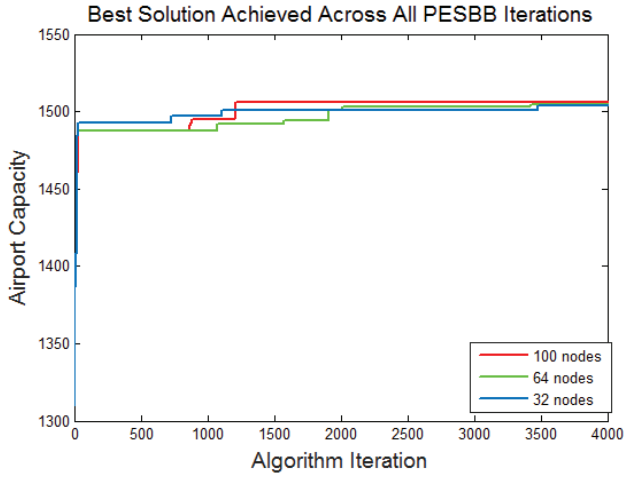

Figure 5: Best solution over time for each implementation of *runway*Simulator.

In addition to studying the effect on solution time through implementation across increasingly parallel computing environments, the final solution of PESBB was examined as well to evaluate its impact on improving capacity at a local airport. First the optimal solutions achieved across each of our implementations of *runway*Simulator in varying CPU environments are compared. Figure 5 depicts the best solution achieved across all PESBB iterations for the 32, 64, and 128 case. Moreover, the final optimal solutions were very similar, but this was to be expected as each implementation was performed for a consistent set number of iterations. The 100 node scenario generated the best solution, and no configuration had an advantage at achieving a superior solution since a set number of iterations were used for each. The range of difference though between all three optimal solutions is 0.5% of the best optimal solution achieved. This points to the robustness of the algorithm and its ability to produce repeatable results even with a stochastic model What can also be observed from Figure 6 is that by iteration 2000 or earlier, all three implementations of PESBB reached their maximum value or very close to their maximum value.

Runway capacity curves visualizing the final solution found by PESSB are provided in Figure 6 to demonstrate the impact that PESBB and simulation optimization can have in generating runway configurations. The capacity curves represent the maximum arrivals and departures for the airport that can be achieved with a runway configuration solution. The median capacity curve (green cone) is provided along with the maximum found by PESBB (orange cone) to demonstrate the level of improvement that can be achieved through simulation optimization. The median solution can be interpreted in this study as a standard airport configuration or benchmark configuration generated without the use of simulation analysis.

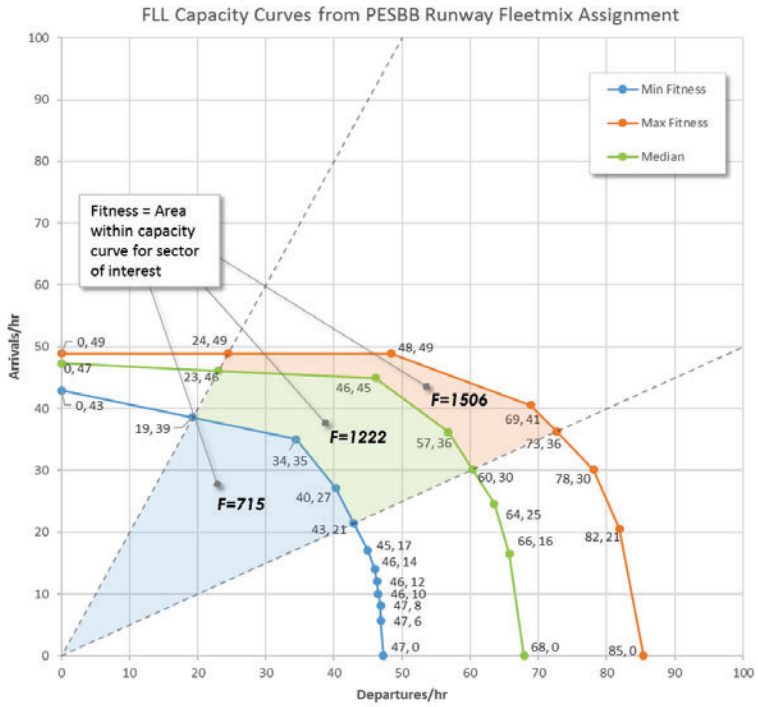

Figure 6: Airport capacity curve containing maximum value found with PESBB.

Here it can be seen that there is over a 23% improvement in airport capacity in going from the median runway configuration to the best configuration located by PESBB. The capacity area value of 1506 found by PESBB is shown inside the orange cone in Figure 6. This was the final solution found during the 100 node implementation of PESBB, which can also be observed on the red line in Figure 5.

5 CONCLUSION AND FUTURE RESEARCH

In this paper, we introduced a parallel implementation of the Empirical Stochastic Branch and Bound (ESBB) algorithm for discrete optimization via simulation, called Parallel Empirical Stochastic Branch and Bound (PESBB). To analyze the performance of the PESBB algorithm, we tested it on the Ackley test function and a real-world, large-scale simulation called *runway*Simulator. These experiments showed that the PESBB algorithm can greatly decrease the total clock time of the ESBB algorithm, particularly when a replication of the simulation is time consuming. Trends concerning improved duration and speed-up with increasing CPUs where observed on both *runway*Simulator and the Ackley testing function. Further testing is required to determine which characteristics within discrete simulation optimization problems the PESBB algorithm is best suited.

ACKNOWLEDGEMENTS

Approved by MITRE for Public Release; Distribution Unlimited. Case Number 16-2402

REFERENCES

Bak, S., J. Baewicz, G. Pawlak, M. Paza, E. K. Burke, G. Kendall. 2011. "A Parallel Branch-and-Bound Approach to the Rectangular Guillotine Strip Cutting Problem." *INFORMS Journal on Computing* 23(1): 15-25.

Buluc, A., K. Madduri. 2011. "Parallel Breadth-First Search on Distributed Memory Systems." *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage, and Analysis* 65.

Caromel, D., A. D. Costanzo, L. Baduel, S. Matsuoka. 2007. "GridBnB: A Parallel Branch-and-Bound Framework for Grids." *Proceedings of the 14th International Conference on High Performance Computing,* 566-579.

Ilinskas, J. 2012. "Parallel Branch-and-Bound for Multidimensional Scaling With City-Block Distances." *Journal of Global Optimization* 54(2): 261-274.

Kishimoto, A., A. Fukunaga, A. Botea. 2013. "Evaluation of a Simple, Scalable, Parallel Best-First Search Strategy." *Artificial Intelligence* 195: 222-248.

Kuzminski, P. 2013. "An Improved runwaySimulator Simulation for Runway System Capacity Estimation." *Proceedings of the 2013 Integrated Communications Navigation and Surveillance (ICNS) Conference,* 2155-4943.

Laursen, P. S. 1994. "Can Parallel Branch-and-Bound Without Communication Be Effective." *SIAM Journal on Optimization* 4(2): 288-296.

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

Ni, E. C., S. R. Hunter, S. G. Henderson. 2013. "Ranking-and-Selection in a High Performance Computing Environment." In *Proceedings of the 2013 Winter Simulation Conference,* edited by R. Pasupathy, S. H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 833-845. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Page, E., L. Litwin, M. McMahon, B. Wickham, M. Shadid, E. Chang. 2012. "Goal-Directed Grid-Enabled Computing for Legacy Simulations." *12th Institute of Electrical and Electronics Engineers/Association for Computing Machinery International Symposium on Cluster, Cloud and Computing*, 873-879.

Shi, L., S. Olafsson. 2000. "Nested Partitions Method for Global Optimization." *Operations Research* 48(3): 390-407.

Tang, K., X. Yáo, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, Z. Yang. 2007. "Benchmark Functions for the CEC' 2008 Special Session and Competition on Large-Scale Global Optimization." *Nature Inspired Computation and Applications Laboratory, USTC, China*, 153-177.

Xia, Y., V. K. Prasanna. 2011. "Topologically Adaptive Parallel Breadth-First Search on Multi-Core Processors." *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* 65.

Xu, W. L., B. L. Nelson. 2013. "Empirical Stochastic Branch-and-Bound for Optimization via Simulation." *IIE Transactions* 45(7): 685-698.

AUTHOR BIOGRAPHIES

SCOTT L. ROSEN is the Chief Engineer for the MITRE Corporation's Operations Research Department. He has spent the last ten years at MITRE applying Operations Research to some of the United States' most critical and challenging Systems Engineering problems. His research interests include Simulation

Optimization, Simulation Metamodeling, Decision Analysis, and Quantitative Systems Engineering. He received his B.S. from Lehigh University in Industrial and Systems Engineering in 1998 and a M.S. and Ph.D. in Industrial Engineering and Operations Research from The Pennsylvania State University in 2000 and 2003, respectively. His email address is srosen@mitre.org.

PETER SALEMI is a senior data scientist in the Operations Research department at the MITRE Corporation. He holds a Ph.D. in Industrial Engineering and Operations Research from Northwestern University and a Masters in Operations Research from the University of California, Berkeley. His research interests include simulation optimization and simulation metamodeling. His email address is psalemi@mitre.org.

BRIAN WICKHAM is a lead simulation modeling engineer at the MITRE corporation. He holds a MS in computer science from Johns Hopkins university and a BS in computer science and mathematics from Rose-Hulman Institute of Technology. He is the lead developer of the MITRE Elastic Goal-Directed Simulation Framework [MEG]. His research interest includes distributed simulation, high performance computing, and data visualization. His email address is bwickham@mitre.org.

ASHLEY WILLIAMS is a Lead Modeling & Simulation Engineer at the MITRE Corporation's Center for Advanced Aviation System Development. He holds degrees in Architectural Engineering from University of Texas at Austin and Transportation Engineering from University of California at Berkeley. Before joining MITRE, he was a manager of international strategic planning at Continental Airlines, Inc., a senior airport planner as Leigh Fisher Associates, and a senior consultant at Booz Allen Hamilton. Over this time, he has designed and implemented many simulation models from simple airport operations models to systems models of the entire national airspace. He is more broadly interested in understanding complex adaptive systems through the use of agent-based modeling and simulation, machine learning, human cognition and multi-objective optimization. His email address is ashley@mitre.org.

CHRISTINE HARVEY is a Modeling and Simulation Engineer at the MITRE Corporation where she specializes in data analysis and high performance computing in simulation. Her research interests include HPC in simulation, Agent Based Modeling, and Big Data Analysis particularity in the field of Healthcare. She completed her Masters in Computational Science from Stockton University in 2013 and is currently working on her PhD in Computational Science and Informatics at George Mason University. Her email address is ceharvey@mitre.org.

ERIN CATLETT is a Senior Modeling & Simulation Engineer at the MITRE Corporation's Center for Advanced Aviation System Development. She holds a BS degree in Mathematics from the College of William and Mary and a MS in Computer Science from the College of William and Mary. Her email address is ecatlett@mitre.org.

SAJJAD TAGHIYEH is a doctoral student in the Department of Systems Engineering and Operations Research at George Mason University. He received his B.S. degree in Industrial Engineering from Sharif University of Technology, Iran. His research focuses on simulation-based optimization. His email address is staghiy2@masonlive.gmu.edu.

JIE XU is an Assistant Professor in the Department of Systems Engineering and Operations Research at George Mason University. He received his PhD degree in Industrial Engineering & Management Sciences from Northwestern University. His research interests include Monte Carlo simulation, simulation-based optimization, computational intelligence, and applications in risk management and aviation. His email address is jxu13@gmu.edu.