Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

Asynchronous Knowledge Gradient Policy for Ranking and Selection

Bogumił Kamiński

Warsaw School of Economics
Al. Niepodległości 162
Warsaw, POLAND

Przemysław Szufel

Warsaw School of Economics
Al. Niepodległości 162
Warsaw, POLAND

ABSTRACT

The simulation of alternative evaluations in the ranking and selection problems often requires extensive amounts of computing power, so it is natural to use clusters with several workers for this task. We propose to extend the standard Knowledge Gradient policy to allow parallel and asynchronous dispatch of computation tasks among workers and denote it as the Asynchronous Knowledge Gradient. Simulation experiments indicate that performance loss due to parallelization of computations is below 25%. This implies that the proposed policy can yield significant benefits in terms of the time needed to obtain a desired approximation of the solution. We describe a master-slave architecture allowing for asynchronous dispatching of jobs among workers that handles problems with worker failures that are encountered in cluster environments. As a test bed of the procedure we developed an emulator of a heterogeneous computing cluster that allows testing of the parallel performance of stochastic optimization algorithms.

1 INTRODUCTION

The goal of the paper is to propose an asynchronously parallelized version of the Knowledge Gradient policy (Frazier 2009, Powell and Ryzhov 2012) for solving computationally demanding ranking and selection (R&S) problems. The approach assumes that simulations are run on clusters consisting of several heterogeneous computing workers, that the simulation itself may have a high variance of computational time for different runs and that the procedure should be robust with respect to worker failure. Therefore, the proposed algorithm is parallelized and allows for asynchronous dispatching of simulation execution requests. We call the policy *Asynchronous Knowledge Gradient* (AKG). In this algorithm the initiation of new computation tasks is conditional on (i) previously collected simulation results and (ii) information about types of computation tasks that were started but have not finished. Taking into account this second information differentiates the AKG policy from the standard sequential Knowledge Gradient (KG).

The mainstream of research on algorithms used for stochastic optimization concentrates either on sequential (single worker) execution or synchronized parallelization, where all available workers in each optimization step are started at the same time and all computations are required to complete before the next step is made. Let us briefly review the most important results in R&S domain. Optimal computing budget allocation (OCBA) proposed by He, Chick, and Chen (2007) is a sequential procedure that performs computation allocation based on asymptotically optimal solution for the problem of maximization of approximated probability of correct selection. In this algorithm in a single step a fixed number of simulations is performed, see section 4.1.1 of Chen and Lee (2011). Chick and Inoue (2001) derive a two-step $\mathscr{LL}(\mathscr{B})$ and a sequential $\mathscr{LL}(\mathscr{S})$ procedure for the Bayesian R&S. Frazier and Powell (2010) in equation (3) formulate the one-step optimal Bayesian allocation R&S problem and later propose the sequential KG($*$) policy that takes into account that the value of measurements is not concave in general. All the above methods share one common feature — when computation allocation decision is made no simulations are in progress. This means that all workers tasks have to be synchronized. Even if we started

multiple parallel computations we have to wait till all of them finished before next step of the algorithm is made.

In this paper we take a different approach and consider *asynchronous parallelization*. Under this scheme, when a worker finishes its job it is instantly assigned a new computational task without waiting for other workers to complete their simulations. A similar method has been considered by Agarwal and Duchi (2012) for continuous problems where they model delays in information flow within a computing grid. Luo and Hong (2011) and Ni, Hunter, and Henderson (2013) consider R&S in a master-slave setting — an approach that is applied here — but they do not consider the Bayesian approach in their procedures.

An asynchronous parallelization policy is reasonable to use when at least one of the following three conditions are met: (i) there is a high variance of execution time of individual simulation, (ii) computing power availability is heterogeneous across nodes in a cluster or (iii) there is the need to handle worker failures. In each of these cases, a researcher has to take into account the fact that all workers are not guaranteed to finish their computation tasks in approximately the same time. In such situations, an asynchronous policy minimizes the time workers are waiting idle for the next task to be assigned. Let us now explain why these assumptions are often encountered in simulation practice.

High variance simulation execution time results from the fact that in many situations it is not possible to estimate *ex-ante* how many simulation steps will be computed for a particular design point and the simulation complexity can differ significantly across design points. This problem is encountered particularly often in discrete event simulations (DES) with stochastic stopping criteria or multi-agent simulations (MAS) where different design points can lead to very divergent simulation dynamics. Fu (1994) notes that in parallel computation scenarios different computational times should be included in algorithm design. He further points out that massively parallel execution of whole simulation instances is still more efficient than another approach to achieve computation speed-up — i.e. splitting a single DES simulation into parallel processes. Currently several tools and algorithms for parallel running of a single DES or MAS simulation on multiple processes have been developed — e.g. for the DES simulation see Misra (1986) and for the MAS simulation see Cordasco et al. (2012).

The *heterogeneous computing power* assumption arises from the fact that in many cases the most cost-effective approach to run a large scale simulation is by utilizing cloud computing. In such environments we can have workers with different speeds (e.g. new and old machines) and different loads (workers can perform alternative tasks in parallel to simulation). An example could be the service offered by Amazon — Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 offers the computing powered measured by EC2 Compute Unit (ECU). The ECU used to have the following definition: "*one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor*" (the original statement is no longer available on Amazon web site). Hence, the exact computing power assigned to 1 ECU is defined as a range rather than an exact value, which makes the process of estimation of the computational time more difficult.

Worker failure handling means that we allow for the possibility that a single simulation might fail and return an error or a worker can break down or become unavailable. This can happen in a large scale simulation optimization — e.g. when a particular node is shut down or simply one of the simulations crashes. Our algorithm can recover in such cases and continue the simulation-optimization process. Hence, our approach enables a fault tolerance both on a task-level and on a workflow-level (Yu and Buyya 2005).

In the development of the procedure we follow a standard assumption made for KG-type algorithms (Frazier 2009) and take that in each iteration of the algorithm we perform one-step ahead allocation optimization. This means that the desired algorithm should be designed in such a way that it can be interrupted in any moment. The need for *optimization interruptibility* arises for instance from the fact that computing power can be bought from cloud services within a particular budget. When the budget and time runs out it should be possible to pick the best alternative for the SO problem and present it to a decision maker. This feature makes our approach particularly suited for simulation problems where the computing power budget is measured in hours and the exact computational power requirements can not be estimated

ex-ante. Hence, the interruptibility can be useful when using cloud computing with a limited budget. This situation justifies the one-step optimization performed under the standard KG policy as being reasonable in practice (which, as is known, is not globally optimal in general).

The paper is structured as follows. In Section 2 we develop the AKG policy. Subsequently, in Section 3 we propose a master-slave architecture that can be used to implement it practically. Finally, in Section 4 we perform verification experiments of the AKG policy versus the sequential KG policy. Additionally, in this section we develop a universal emulator of computing clusters that can be used to compare different stochastic optimization algorithms in asynchronous and parallel environments.

2 ASYNCHRONOUS KNOWLEDGE GRADIENT ALGORITHM

Let $\{1,2,\ldots,N\}$ be the set of alternatives from which we want to select the best. We will follow the standard Knowledge Gradient (Frazier 2009) approach and assumptions. Each alternative has some unknown mean value that we want to maximize. These mean values cannot be observed but we can run a simulation and obtain an unbiased sample of it, normally distributed with a variance σ_ε^2 that is known to the researcher.

We take a Bayesian approach and assume that we have a random variable $Y=(Y_1,Y_2,\ldots,Y_N)$ representing our beliefs about distribution of values of alternatives. We take it that the prior distribution of Y is multivariate normal with means $(\mu_{(0),1},\mu_{(0),2},\ldots,\mu_{(0),N})$ and a covariance matrix that is diagonal with variances $\sigma^2_{(0),i}$.

Now let us assume that in step k we have run the simulation for alternative i and have observed a sample that has value y. Then, following the Bayes' rule we can update our beliefs concerning distribution Y_i as follows (Powell and Ryzhov 2012):

$$\sigma^2_{(k),i}=\left(\frac{1}{\sigma^2_{(k-1),i}}+\frac{1}{\sigma^2_\varepsilon}\right)^{-1},$$

$$\mu_{(k),i}=\left(\frac{\mu_i^{(k-1)}}{\sigma^2_{(k-1),i}}+\frac{y}{\sigma^2_\varepsilon}\right)\sigma^2_{(k),i}.$$

Notice that by the independence assumption, $\sigma^2_{(k),j}$ and $\mu_{(k),j}$ for $j\neq i$ are not changed.

In an asynchronous environment we have a set W denoting a pool of workers and we want to assign a new task to *worker* $\in W$ immediately after it becomes available for computations. After observing the k-th sample the worker that provided it is waiting for the next job. However, when deciding on what task to assign to it we have to take into account that other workers can be performing simulations that were started earlier but have not yet finished (this is a key assumption for the asynchronous algorithm). This is a desired approach because we do not want to wait for a decision until all workers finish their computations as this would imply a loss of available computing power or an unnecessary delay. Because we have $|W|$ workers then while deciding on a new job for a worker that has just finished computations, we have maximally $|W|-1$ simulations running. Denote the number of scheduled but not observed measurements of alternative i as s_i (this is the number of workers currently running a simulation for alternative i). We thus have $\sum_{i=1}^N s_i\leq|W|-1$. Denote $\mathbf{s}=(s_1,s_2,\ldots,s_N)$.

Let us now assess our beliefs regarding how s_i scheduled measurements of alternative i will influence $\mu_{(k),i}$ after they are performed. Using the standard Bayesian inference (Powell and Ryzhov 2012) we see that the distribution of these beliefs is normal with mean $\mu_{(k),i}$ and variance given by formula:

$$\sigma^2_{(k),i|s_i}=\sigma^2_{(k),i}-\left(\frac{1}{\sigma^2_{(k),i}}+\frac{s_i}{\sigma^2_\varepsilon}\right)^{-1}.$$

Informally we can say that the variance $\sigma^2_{(k),i|s_i}$ measures by how much $\mu_{(k),i}$ can change after s_i measurements. It is instructive to consider two extreme cases. If $s_i=0$ then $\sigma^2_{(k),i|0}=0$ which simply

means that without making any measurements we will not change our beliefs about $\mu_{(k),i}$. Next notice that as s_i increases $\sigma^2_{(k),i|s_i}$ also increases. The more measurements we make the larger changes of $\mu_{(k),i}$ can happen. Finally $\lim_{s_i\to+\infty}\sigma^2_{(k),i|s_i}=\sigma^2_{(k),i}$. This is also in line with the intuition. By making infinitely many measurements we will learn the true value of the mean and after k measurements the uncertainty of our beliefs is captured by their variance that is equal to $\sigma^2_{(k),i}$.

In the following computations by $\Phi(x)=(2\pi)^{-1}\int_{-\infty}^{x}e^{-t^2/2}dt$ we denote the cumulative distribution function of a standard normal random variable. Then the cumulative distribution function of posterior beliefs $F_{(k),i|s_i}(x)$ about $\mu_{(k),i}$ is $\Phi((x-\mu_{(k),i})/\sigma^2_{(k),i|s_i})$ for $s_i\geq 1$ and is a function with $0-1$ jump at $\mu_{(k),i}$ for $s_i=0$. Denote a random variable with this distribution as $M_{(k),i|s_i}$.

Now let us consider a random variable $V_{(k)|\mathbf{s}}$ that represents the expected value of a selected alternative after all $\mathbf{s}$ currently executed simulations finish. Notice that:

$$V_{(k)|\mathbf{s}}=\max_{i\in\{1,2,\ldots,N\}}M_{(k),i|s_i}.$$

The formula reads that after s_i measurements we expect that our beliefs about the mean of alternative i are described by the normal random variable $M_{(k),i|s_i}$ (keeping in mind that for $s_i=0$ this random variable is degenerated to a point $\mu_{(k),i}$). We know that having performed all $\mathbf{s}$ measurements we will select the alternative with the highest evaluation of the mean. Therefore $V_{(k)|\mathbf{s}}$ is the maximum of $M_{(k),i|s_i}$. Remember that we have assumed that our beliefs about evaluation of alternatives are independent (measurement of one alternative does not influence our beliefs about other alternatives) therefore random variables $M_{(k),i|s_i}$ are independent.

Now we can define the AKG policy at time k that takes into account the number of scheduled computations s_i as:

$$AKG(k,\mathbf{s})=\arg\max_{i\in\{1,2,\ldots,N\}}E\left(V_{(k)|\mathbf{s}+\mathbf{e}_i}\right),$$

where $\mathbf{e}_i$ is the i-th standard basis vector in N-dimensional space (consisting of all 0 and 1 in i-th position).

Notice $\mathbf{s}+\mathbf{e}_i$ in the maximization formula means that we are assuming that after starting the new computation s_i will increase by 1 because the waiting worker will be scheduled to simulate alternative i.

The AKG policy defined above can be interpreted as follows. In the step k of the algorithm we follow a standard KG assumption that we are planning to assign only one more observation. However, we know that before assigning it we are going to collect s_i measurements of the alternative i. Therefore we want to select an alternative i for which we get the most benefit from collecting s_i+1 observations instead of s_i observations. Notice that this optimization task can be equivalently formulated as follows:

$$E\left(V_{(k)|\mathbf{x}}\right)\to\max\quad\text{subject to}\quad\forall 1\leq i\leq N: x_i\geq s_i\wedge\sum_{i=1}^{N}x_i=1+\sum_{i=1}^{N}s_i,$$

where we denote $\mathbf{x}=(x_1,x_2,\ldots,x_N)$. This formulation shows that in a single step of the procedure the AKG allocation is an optimal solution to R&S problem with computing budget $1+\sum_{i=1}^{N}s_i$, as defined for example in equation (3) by Frazier and Powell (2010), subject to additional constraints $x_i\geq s_i$ that some of the computations are already started. This additional assumption significantly reduces the optimization process complexity and makes the AKG policy relatively simple to compute.

The key difficulty is the evaluation of $V_{(k)|\mathbf{s}}$. Because all $M_{(k),i|s_i}$ are independent we can compute its cumulative distribution function as $C_{(k)|\mathbf{s}}(x)=\prod_{i=1}^{n}F_{(k),i|s_i}(x)$. Using this CDF we can evaluate $E\left(V_{(k)|\mathbf{s}+\mathbf{e}_i}\right)$ by a numerical approximation of univariate improper integral $\int_{-\infty}^{+\infty}xdC_{(k)|\mathbf{s}}(x)$. In our code (available for download at `http://bogumilkaminski.pl/pub/AKGv1.1.zip`) in order to evaluate the integral we use the formula given by equation (13) in Ross (2003) and next transform it to the $[0,1]$ interval with the variable substitution $t=1/(x+1)$:

$$\int_{-\infty}^{+\infty} x dC_{(k)|\mathbf{s}}(x) \;=\; \int_{0}^{+\infty} 1 - C_{(k)|\mathbf{s}}(x) - C_{(k)|\mathbf{s}}(-x)dx \;=\; \int_{0}^{1} \frac{1 - C_{(k)|\mathbf{s}}\left(t^{-1}-1\right) - C_{(k)|\mathbf{s}}\left(1-t^{-1}\right)}{t^2} dt.$$

Next, we numerically approximate the integral using adaptive Gauss-Kronrod quadrature with 61 nodes and 10^{-12} deviation threshold used for interval bisection decision (see the method `calculateEV_KG` in the class `simtools.CalculateEv` in the source code).

Using the approach proposed here we do not need to evaluate the derivative of $C_{(k)|\mathbf{s}}$, which would increase computational complexity of the algorithm and reduce its numerical accuracy. In the implementation we also have to remember to distinguish cases $s_i > 0$ and $s_i = 0$, because in the latter $F_{(k),i|0}(x)$ is degenerate.

For practical application we note that the process of identification of $AKG(k,\mathbf{s})$ scales quadratically with the number of alternatives N (we take the maximum of N options and in each option we have the product of N cumulative distribution functions). However, we have found it to be acceptably fast relative to the expected execution times of simulation steps. This is the only time consuming task performed by the master machine as described in Section 3. In calculations on a laptop computer with the master process running on a single core for $N = 5$ and $|W| = 5$ we have found that computing $AKG(k,\mathbf{s})$ takes around 20 milliseconds.

If the optimization process is finished after k measurements we select the alternative v, for which our current belief about mean $\mu_{(k),v}$ is maximal — exactly as in the standard KG policy.

Apart from the formulas given above, that identify optimal decisions for worker scheduling, the AKG policy requires appropriate computation architecture to run it, as we describe in the next section.

3 MASTER-SLAVE ARCHITECTURE FOR ASYNCHRONOUS KNOWLEDGE GRADIENT

In this section we outline how an AKG algorithm can be implemented in a master-slave architecture. We assume that we have one master machine and a set W of worker machines. The master controls the entire simulation process (the execution of the AKG algorithm) while workers are assigned simulation execution tasks. A worker starts the simulation for a configuration given by the alternative $i \in \{1,2,\ldots,N\}$ when it is invoked by the master. After the worker finishes the simulation it notifies the master that it is ready to return simulation output and start a new task.

The full procedure has been presented as Algorithm 1. Let us now highlight its major features. Every *worker* $\in W$ apart from executing simulations possesses the following three attributes:

1. *status*:
   (a) `free` — newly created *worker*;
   (b) `working` — simulation is running;
   (c) `completed` — simulation completed and can take up a new job;
   (d) `error` — simulation finished with an error, but *worker* is ready to start a new simulation job;
   (e) `down` — node will not be available — for example due to failure or unavailability (e.g. worker assigned to other tasks by a cluster administrator);
2. *job*:
   a number of currently simulated alternative; if *status* is not equal to `working` it is undefined;
3. *output*:
   a value returned after simulation execution; if *status* is not equal to `completed` it is undefined.

A complete state transition diagram for worker attributes can be found in Figure 1. The attribute *job* is additionally stored on the master machine in order to be able to retrieve it in case the worker *status* becomes `down`, whereupon it might even be impossible for the master to communicate with it. When a worker goes `down` it is removed from the set W. Similarly new workers can be added to set W at any instance in the main loop in Algorithm 1 execution. In the case of a new worker being added it must have set its *status* set to `free`.

Similarly to the standard KG procedure, initial beliefs $\mu_{(0),i}$ and $\sigma_{(0),i}$ must be supplied by the researcher.

At the start of Algorithm 1 there are $|W|$ free workers. In order to keep the algorithm simple we assume that they are assigned to tasks sequentially. However, it is possible to find an optimal initial allocation $\mathbf{s}$ that maximizes knowledge gain under the constraint that $\sum_{i=1}^{N} s_i = |W|$ and assign tasks to workers simultaneously — an approach equivalent to solving problem given by the equation (3) from Frazier and Powell (2010). Notice, however, that for large $|W|$ and N this could be a computationally expensive task.

Algorithm 1 Asynchronous Knowledge Gradient procedure for the master node

```
1: procedure AKG_MASTER
2:     Initialize μ_(0),i and σ_(0),i
3:     s_i ← 0 for all i ∈ {1,2,...,N}
4:     k ← 0
5:     worker.status ← free for all worker ∈ W
6:     repeat
7:         wait until (exists worker ∈ W: worker.status not equal to working)
8:                 or (loop was externally interrupted)
9:         i ← worker.job
10:        if worker.status equals completed then          ▷ collect results from a completed job
11:            k ← k + 1
12:            y ← worker.output
13:            Update μ_(k),i and σ_(k),i.
14:        end if
15:        if worker.status is not free then               ▷ decrease scheduled computation counter
16:            s_i ← s_i − 1
17:        end if
18:        if worker.status equals down then               ▷ remove crashed workers from the pool
19:            W ← W − {worker}
20:            if for all worker ∈ W worker.status equals down then  ▷ terminate if no workers are left
21:                break
22:            end if
23:        else
24:            Calculate next point to simulate AKG(k,s)
25:            s_AKG(k,s) ← s_AKG(k,s) + 1;
26:            worker.job ← AKG(k,s)
27:            start simulation computation on worker
28:        end if
29:    until Stopping condition is true
30:    return arg max_{i∈{1,2,...,N}} μ_(k),i              ▷ return currently best alternative
31: end procedure
```

The master spends most of its time in the **wait** step of the algorithm. Workers finish their jobs asynchronously so it is possible that several workers finish computations at almost the same time. If the master encounters a situation that it has several available workers, the most efficient case is when it processes workers in the following *status* sequence: `down`, `completed`, `error`, `free`.

Lastly let us note, as mentioned in the introduction, that we assume that the algorithm can have an arbitrary stopping conditions or even can be externally interrupted at any time. This reflects the fact that in practice the stopping conditions can have different forms, for example: computing budget (expressed in money or in processor time), execution time (the total time allowed to run all simulations) or criteria based on predicted values of considered alternatives $\mu_{(k),i}$.

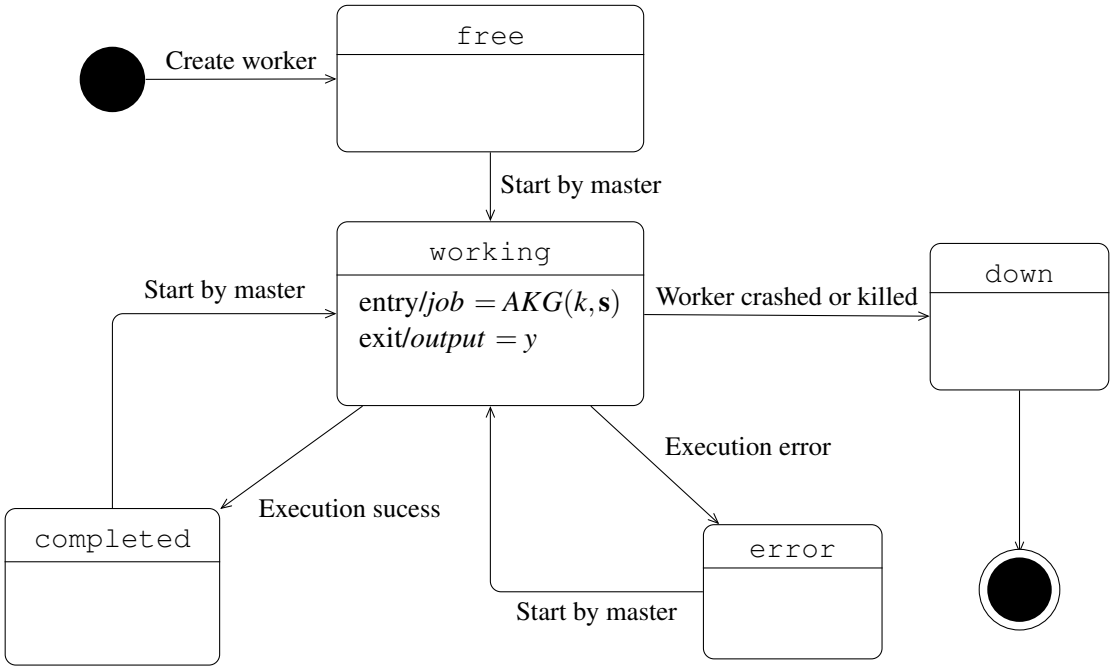

Figure 1: State transition diagram for worker status.

Here it should be noted that arbitrary stopping of AKG algorithm might introduce a slight bias in obtained evaluations of $\mu_{(k),i}$. The reason for such a situation is that simulation completion times might be correlated with observed values of y_i. For a broader discussion of this problem see Ni, Hunter, and Henderson (2013). This means that at an arbitrary algorithm stopping moment it is more probable that simulations that are "under computation" have large completion times. Therefore it is recommended that, if possible, after stopping of the main loop of the algorithm all started simulations are finished and their outputs are taken into account. If this policy is applied the obtained evaluations $\mu_{(k),i}$ will be unbiased. However, it should be noted that this problem is becoming less important when the number of collected samples is increasing. Additionally, for the same reasons as explained above, if we have the strong correlation of simulation completion times with y_i, the Bayesian updating proposed in this paper is only an approximation because the distribution of y_i does not have to be exactly normal with zero bias and variance σ_ε^2. This might lead to a bias in the knowledge gradient evaluation. In this paper we assume that this problem has a negligible influence of the outcome of the procedure for larger samples because then the proportion of "fast" and "slow" computations for each alternative i will be approximately correct.

4 SIMULATION EXPERIMENTS

The goal of this section is to present the numerical results of AKG method tests. We benchmark the AKG algorithm vs the standard sequential KG. The test scenarios for algorithm performance are based on KG tests proposed by Frazier and Powell (2007).

The layout of this section is as follows. Firstly, we propose a testbed for asynchronous SO algorithms — *virtual computing cluster emulator for R&S* (VCCE4RS). Secondly, we describe the details of the experiment design. Finally, we discuss the results of simulation experiments and compare the ability of AKG to find an optimal solution vs KG performance.

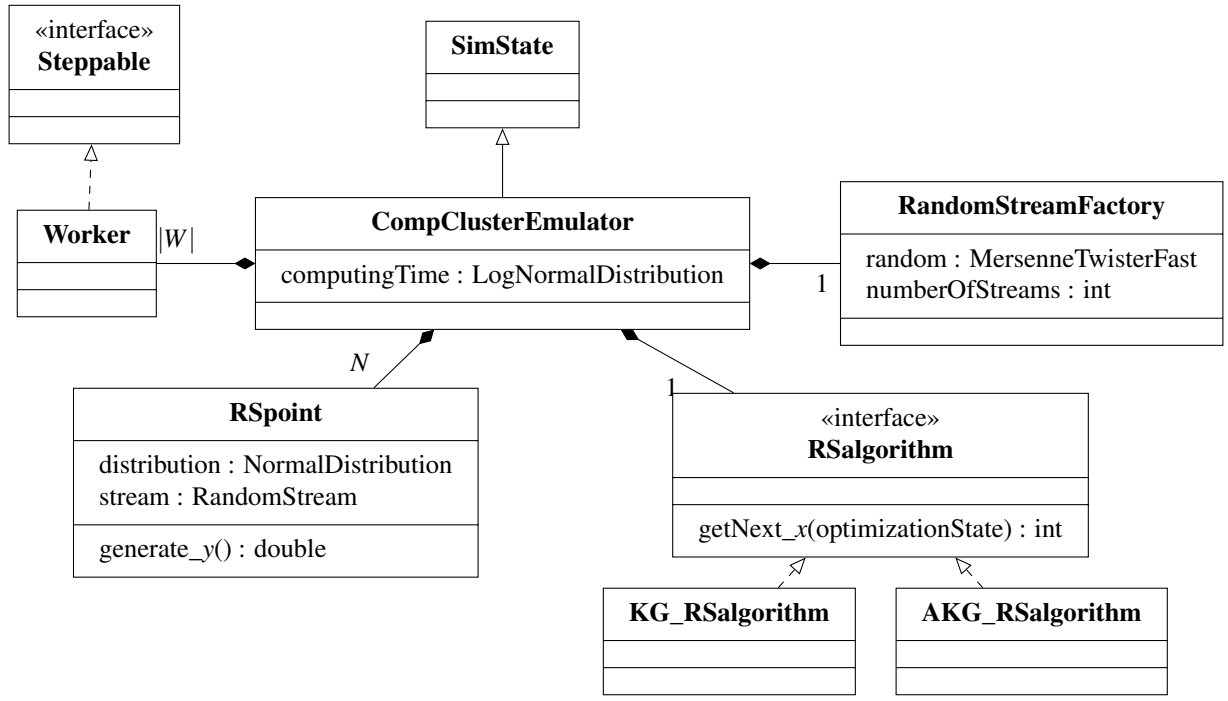

Figure 2: The class diagram for virtual computing cluster emulator for R&S (VCCE4RS).

4.1 Virtual computing cluster emulator for R&S (VCCE4RS)

To test the efficiency of our AKG algorithm we have created a simulation model that emulates a computing cluster. The virtual computing cluster emulator for R&S simulation-optimization (VCCE4RS) is a simulation framework for testing SO algorithms programmed in Java and we use the MASON library version 17 (`http://cs.gmu.edu/ eclab/projects/mason/`), see Luke et al. (2005).

For probability density functions and integration we use Apache Commons — Math version 3 (`http://commons.apache.org/proper/commons-math/`).

The Eclipse (`http://www.eclipse.org/`) project containing source code and binaries of VCCE4RS is available for download at `http://bogumilkaminski.pl/pub/AKGv1.1.zip`.

The VCCE4RS can simulate a computing cluster with an arbitrary amount of workers $|W|$ for any given number of points $|N|$. We assumed a star topology where the simulation is controlled by a central node (master) and the calculations are performed by subsidiary nodes — workers.

The class diagram of VCCE4RS is presented on Figure 2. The main class of our simulation environment is the class `CompClusterEmulator`. The object of the `CompClusterEmulator` class contains the following fields:

- `R&S points` — represent various alternatives presented by random variables $(Y_1, Y_2, \ldots, Y_N)$;
- `Workers` — represent computing nodes in a cluster;
- `R&S algorithm` that can be plugged-in to test of various SO algorithms on the virtual cluster.

The points represent alternatives $\{1, 2, \ldots, N\}$. The virtual simulation process is represented by generating a point from the normal distribution.

In our computing cluster model, the workers are represented as agents (in MASON terminology they implement the Steppable interface). The virtual computing times are represented by delays of workers.

Table 1: Scenarios for simulation experiments

algorithm	N	workers — $\|W\|$	steps — k	repetitions
KG	$\{5,10\}$	1	85	114122
AKG	$\{5,10\}$	$\{1,2,3,4,5\}$	85	114122
KG	20	1	85	107560
AKG	20	$\{1,2,3,4,5\}$	85	107560

In our framework, R&S algorithms can be plugged-in through an interface. Hence, we can easily compare different SO algorithms. We have implemented only KG and AKG algorithms within the proposed framework but other algorithms can easily be added by implementing the `RSalgorithm` interface.

In order to ensure the comparability of experiments for different SO algorithms, the VCCE4RS uses a common random numbers technique (Law 2007). More precisely, we use the leap-frog approach for handling CRN (Pawlikowski, Schoo, and McNickle 2006). We utilize a single Mersenne Twister (Matsumoto and Nishimura 1998) random number generator and split it's random numbers into $N+|W|$ streams with 1 random number stream for each `R&S point` and $|W|$ streams utilized for generating virtual computing for the workers. The main loop described in Algorithm 1 is run within the simulation state class `CompClusterEmulator`.

The proposed VCCE4RS can be utilized to test and benchmark various asynchronous or synchronous simulation-optimization algorithms in R&S problems. Notice that in the master-slave setting the design assumptions behind our simulator are robust to typical problems encountered in cluster computing such as network latency. This is due to the fact that all the synchronization of slave nodes is ensured by a centralized simulation dispatch on the master node. If we had considered decentralized computation architectures, like for example *peer-to-peer*, the design of the algorithms would have taken such issues into account.

4.2 Experiment design

The goal of the simulation experiments is to calculate the speed of convergence of the KG and AKG algorithms. For the model parametrization initialization we follow the approach proposed by Frazier and Powell (2007).

The experiment scenarios are represented in Table 1. We consider three values for N: 5, 10 and 20 points. For each design point $i \in \{1,\ldots,N\}$, we generate the expected values $\tilde{\mu}_i$ ('unknown' to the KG/AKG algorithm) from the standard normal distribution i.e. $\tilde{\mu}_i \sim N(0,1)$. For each point we assume that measurement standard deviations are equal to 1 i.e. $\sigma_\varepsilon^2 = 1$. We run the AKG simulation with $|W| = \{1,2,3,4,5\}$ workers. It should also be noted that we run AKG with one worker in order to validate our implementation — AKG with one worker should have performance identical to KG. Altogether 12 different simulation scenarios have been considered with more than 100,000 repetitions for each scenario.

Testing the AKG algorithm requires simulation of computing times. We have assumed that those times are log normally distributed, i.e. $computingTime \sim LN(0,1)$.

The goal of our simulation experiments is to measure how the parallelization of the KG method decreases the speed at which the algorithm converges. Hence, when testing the AKG on VCCE4RS, we do not consider simulation execution errors, worker crashes or scenarios where a new worker is being added to the computing grid. In our simulations the workers can have only three states presented in Figure 1: `free`, `working`, `completed`.

4.3 Simulation results

The simulation results for the AKG algorithm are presented in Figures 3 and 4 (KG simulation results were the same as results reported by the AKG with 1 worker – see the discussion of experiment design in Section 4.2). As a measure of algorithm efficiency we use an expected loss of reward $E(\max_i \tilde{\mu}_i - \tilde{\mu}_{\arg\max_i \mu_{(k),i}})$ after obtaining k simulation outcomes (i.e. performing k simulations). The expected loss is the difference

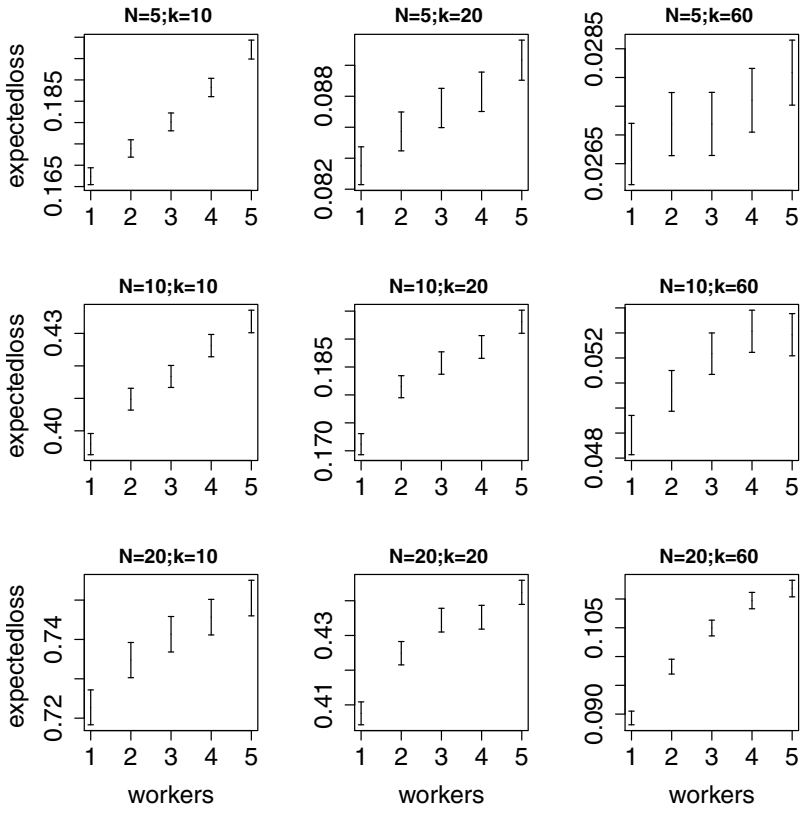

Figure 3: The expected loss for different number of workers $|W|$, number of successfully executed simulations k and alternatives N. For each point we present the mean and 95% its confidence interval.

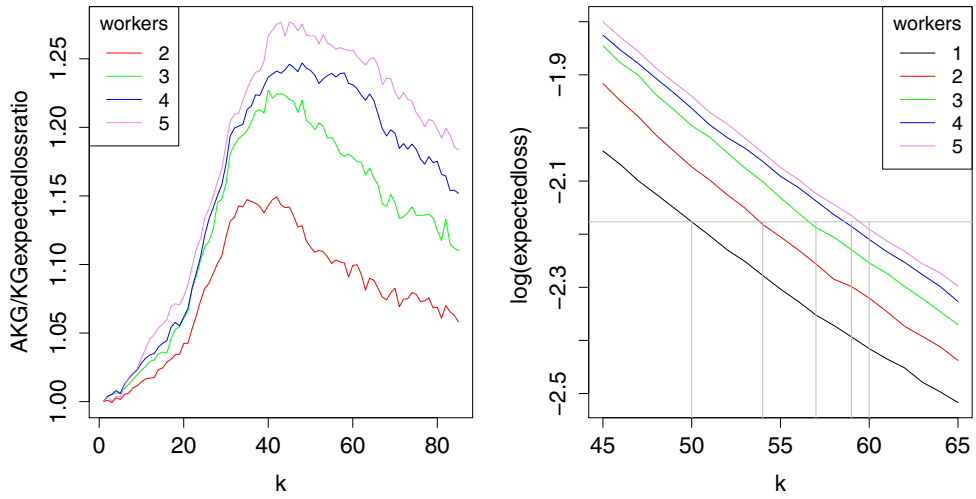

Figure 4: Analysis of the expected loss for different number of workers $|W|$ and increasing number of successfully executed simulations k for $N = 20$ alternatives.

between the objective value for a truly optimal solution and the expected objective value of the solution selected by the algorithm. In Figure 3 it can be seen that with increasing number of workers the expected loss increases. However, the performance differences are very small (we had to run around 100000 simulation replications to be able to detect statistically significant differences).

In order to assess the level of the performance loss in Figure 4 on the left we plot the ratio of the expected loss generated by AKG relative to the expected loss for KG (one worker) for $N = 20$. We can see that the AKG performance degrades when the number of worker increases, but it is not more than $\sim$25% for 5 workers. Considering the fact that with 5 workers we can work 5 times faster we conclude that the benefit of parallelization is substantial. For $N \in \{5, 10\}$ the loss ratio is lower and has a similar shape. The right subplot compares the number of calculated outcomes and the log of expected loss for varying number of workers. We can see that in order to achieve the expected loss gained after 50 evaluations (horizontal grey line) in the standard KG algorithm we need not more than 60 evaluations for AKG with 5 workers. Moreover adding each worker produces lower increase in number of evaluations needed, as we need 50 evaluations for 1 worker, 54 for 2 workers, 57 for 3 workers, 59 for 4 workers and 60 for 5 workers (vertical grey lines).

The simulation results show that use of parallelization (i.e. moving from KG to AKG) can lead to *significant computation time savings* when multiple workers are available.

5 CONCLUDING REMARKS

In this paper we have proposed an asynchronous version of the knowledge gradient algorithm for ranking and selection problems. In this procedure when a new computation task is initiated information about types of computation tasks that are started but have not finished yet is taken into account.

We have validated our approach with a simulation and determined that the AKG method scales very well — adding new worker nodes to a computing cluster almost linearly decreases the time required to find an optimal solution. However, what remains for further work is to develop a formal theory that establishes bounds on the efficiency loss of the AKG policy vs. the sequential KG policy.

In this text we have restricted our consideration to the master-slave architecture where a centralized master server controls many worker nodes. This design choice was made because it is a typical scenario for cluster computing and rental of computational power. Another reason is that the master-slave protocol is simple and easy to implement. However, other approaches can be also considered, such as the *peer-to-peer* communication in computing grids as discussed for stochastic optimization by Agarwal and Duchi (2012).

The proposed asynchronous master-slave approach can be also easily applied to other SO algorithms. In particular a natural venue for extension of the proposed asynchronous optimization approach is to apply it in combination with OCBA-type or multi-step ahead policies. Moreover, the developed Virtual Computing Cluster Emulator for R&S simulation-optimization (VCCE4RS) can be utilized to test other algorithms.

Finally an important area for further consideration is analyzing of the consequences of the numerical instability of KG and AKG algorithms in general. We have discovered that in many scenarios the choice of next point $E\left(V_{(k)|\mathbf{s}+\mathbf{e}_i}\right)$ leads to tie or near-tie situations and the next point $AKG(k,\mathbf{s})$ can be chosen due to numerical error rather than a maximal expected value.

ACKNOWLEDGMENTS

The authors would like to thank the two anonymous referees for insightful comments helped to significantly improve the text.

REFERENCES

Agarwal, A., and J. C. Duchi. 2012. "Distributed Delayed Stochastic Optimization.". In *IEEE 51st Annual Conference on Decision and Control (CDC)*, 5451–5452. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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

Chick, S., and K. Inoue. 2001. "New Two-Stage and Sequential Procedures for Selecting the Best Simulated System". *Operations Research* 49 (5): 732–743.

Cordasco, G., R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, and C. Spagnuolo. 2012. "A framework for distributing agent-based simulations". In *Euro-Par 2011: Parallel Processing Workshops*, 460–470. Springer.

Frazier, P. 2009. Knowledge-gradient methods for statistical learning. Ph. D. thesis, Princeton University.

Frazier, P., and W. Powell. 2007. "The knowledge gradient policy for offline learning with independent normal rewards". In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, 143–150. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Frazier, P. I., and W. B. Powell. 2010. "Paradoxes in Learning and the Marginal Value of Information". *Decision Analysis* 7 (4): 378–403.

Fu, M. C. 1994. "Optimization via simulation: A review". *Annals of Operations Research* 53 (1): 199–247.

He, D., S. E. Chick, and C.-H. Chen. 2007. "Opportunity cost and OCBA selection procedures in ordinal optimization for a fixed number of alternative systems". *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 37 (5): 951–961.

Law, A. M. 2007. *Simulation modeling and analysis*. McGraw-Hill Education Boston, MA.

Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. 2005. "Mason: A multiagent simulation environment". *Simulation* 81 (7): 517–527.

Luo, J., and L. Hong. 2011. "Large-scale ranking and selection using cloud computing". In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 4051–4061. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Matsumoto, M., and T. Nishimura. 1998. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8 (1): 3–30.

Misra, J. 1986. "Distributed discrete-event simulation". *ACM Computing Surveys (CSUR)* 18 (1): 39–65.

Ni, E., S. Hunter, and S. 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. Kuhl, 833–845. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pawlikowski, K., M. Schoo, and D. McNickle. 2006. "Modern generators of multiple streams of pseudo-random numbers". In *Proceedings of the International Mediterranean Modelling Multiconference (ESM06)*, 553–559. Barcelona.

Powell, W. B., and I. O. Ryzhov. 2012. *Optimal learning*. John Wiley & Sons.

Ross, A. M. 2003. Useful bounds on the expected maximum of correlated normal random variables. Working paper 03W–004, Industrial and Systems Engineering, Lehigh University.

Yu, J., and R. Buyya. 2005. "A taxonomy of workflow management systems for grid computing". *Journal of Grid Computing* 3 (3-4): 171–200.

AUTHOR BIOGRAPHIES

BOGUMIŁ KAMIŃSKI is Head of Decision Support and Analysis Division at Warsaw School of Economics. He received his M.S. and Ph.D. in Quantitative Methods in Economics from Warsaw School of Economics. His email address is bkamins@sgh.waw.pl.

PRZEMYSŁAW SZUFEL is Assistant Professor in Decision Support and Analysis Division at Warsaw School of Economics. He received his M.S. and Ph.D. in Quantitative Methods in Economics from Warsaw School of Economics. His email address is pszufe@sgh.waw.pl.