Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

A MOVING MESH APPROACH FOR SIMULATION BUDGET ALLOCATION ON CONTINUOUS DOMAINS

Mark W. Brantley
Chun-Hung Chen

Dept. of Systems Engineering and Operations Research
4400 University Drive, MS 4A6
George Mason University
Fairfax, VA 22030, U.S.A

ABSTRACT

We introduce a moving mesh algorithm for simulation optimization across a continuous domain. During each iteration, the mesh movement is determined by allocating simulation runs to partitions of the domain that are critical in the process of identifying good solutions. The partition boundaries are then adjusted to equally distribute the allocation runs between each partition. To test the moving mesh algorithm, we implemented it using the OCBA method to allocate simulation runs to each partition. But, the simplicity of the procedure should provide flexibility for it to be used with other simulation optimization techniques. Results are presented for several numerical experiments and suggest that the technique has potential given further development.

1 INTRODUCTION

Simulation optimization is a method to find a design consisting of combination of input decision variable values of a simulation system that optimizes a particular output performance measure of the system. When presented with a stochastic, continuous domain with an infinite number of values for each input decision variable and a finite simulation budget, we must efficiently allocate our simulation runs in order to investigate the combinations of input decision variable values (Law and Kelton 2000). Most simulation optimization methods use points in the domain to represent designs (Swisher et al. 2000). These methods typically require indifference zones to not only represent solutions within a certain distance of the best solution but to also ensure that simulation runs are not wasted by comparing two designs that are essentially the same. Shi and Olafsson present a randomized method for global optimization called Nested Partition (NP) that seeks to efficiently concentrate the computational effort in parts of the domain that may be most likely to contain the global optimum (Shi and Olafsson, 2000). NP aggregates the information from designs to allocate additional runs and then partitions the domain to search the most promising region.

This paper investigates a different approach of using partitions of the domain that is motivated by mesh moving techniques for finite difference and finite element schemes. Similar to nested partition, these numerical techniques can use what is typically called local mesh refinement to divide the mesh in certain regions of the domain to reduce the error or to adapt to nonuniformity (Arney and Flaherty 1986). An alternate approach for adapting the mesh is to keep a fixed number of partitions on the domain but to move the mesh to have a fine grid where needed and a course grid elsewhere (Adjerid and Flaherty 1986). The method we introduce mirrors this alternate approach. Instead of partitioning the domain by refining the mesh like nested partitioning, we will move the mesh to concentrate the search in the most promising region. As we move the mesh, we reduce the size of the partitions in the most promising regions and increase the size of the partitions elsewhere.

By changing our problem from finding the most promising point in our domain to searching for the most promising region, we have transformed our continuous stochastic optimization problem to a discrete stochastic optimization problem. We now seek to identify the best partition *b* out of *k* competing partitions. While discrete stochastic optimization is still an active field of research, recent advances provide techniques that greatly reduce the number of simulation runs required to obtain a good or the best solution. Given the simplicity of the moving mesh algorithm, we expect that it can be coupled with many of the discrete stochastic optimization techniques. However, we focused our efforts on using a highly efficient technique developed by Chen et al. (2000) called the Optimal Computing Budget Allocation (OCBA) method. Their numerical comparisons have shown that OCBA can achieve a speedup factor of approximately 4 for a small number of competing designs and can be as much as 20 times faster than traditional approaches for a much larger number of designs.

2 THE OCBA ALGORITHM

Since the implementation of our moving mesh technique in this paper utilizes OCBA to award runs to partitions during each iteration, we present a summary of the technique. OCBA allocates simulation runs by considering the following optimization problem:

$$\max_{N_1,\ldots,N_k} \quad P\{CS\}$$
$$s.t.\, N_1 + N_2 + \cdots + N_k = T\,. \tag{1}$$

Under a Bayesian model, OCBA approximates the probability of correctly selecting the best design, P{CS}, using the Bonferroni inequality and offers an asymptotic solution to this approximation. In particular, OCBA allocates simulation runs according to:

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i / \delta_{b,i}}{\sigma_j / \delta_{b,j}}\right)^2,$$
$$\mathrm{i,j} \in \{1,2,\cdots,\mathrm{k}\}, \text{ and } \mathrm{i} \neq \mathrm{j} \neq \mathrm{b}, \tag{2}$$

$$N_b = \sigma_b \sqrt{\sum_{i=1, i\neq b}^{k} \frac{N_i^2}{\sigma_i^2}}\,. \tag{3}$$

Chen et al. (2000) denote by

N_i :	the number of simulation runs for design *i,*
X_{ij} :	the *j*-th independent and identically distributed sample of the performance measure from design *i*,
$\bar{J}_i$:	the sample average of the simulation output for runs in design *i,* $\bar{J}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} X_{ij}$,
σ_i^2 :	the variance of the simulation output for runs in design *i*, approximated by the sample variance of the simulation output, i.e., $\sigma_i^2 \approx S_i^2 = \frac{1}{N_i - 1}\sum_{j=1}^{N_i}(X_{ij} - \bar{X}_i)^2$,
b:	the design having the smallest sample mean performance measure, i.e., $\bar{J}_b \le \min_i \bar{J}_i$, $\delta_{b,i} \equiv \bar{J}_b - \bar{J}_i$.

For our discussion, we will adopt this same convention.

3 THE MOVING MESH ALGORITHM

3.1 Moving Mesh Algorithm Steps

Limiting our discussion to a one-dimensional domain without loss of generality, we have *k* continuous partitions (intervals) on a domain of length *L* and we denote

Φ_i :	the *i*-th partition where by convention we order the partitions such that $X_{i,j} \le X_{(i+1),m}\ \forall\ j,m$,
$X_{i,j}$:	the *j*-th independent and identically distributed sample of the performance measure from the region of the domain currently assigned to partition *i*,
$\Omega_{i,j}$:	the *j*-th boundary for partition *i*. For the one-dimensional case each Φ_i will have two boundaries $\Omega_{i,1}$ and $\Omega_{i,2}$ with coordinates $x_{i,1}$ and $x_{i,2}$ constructed such that $x_{i,2} = x_{(i+1),1}\ \forall\ i < k$.

Although the implementation of the moving mesh algorithm is dependent upon the allocation method that we use, the basic algorithm is very simple:

1. Determine *k*, the number of partitions, and then construct the mesh uniformly across the domain such that $x_{i,2} - x_{i,1} = x_{j,2} - x_{j,1}\ \forall\ i,j$, and the intervals span the entire domain such that $x_{k,2} - x_{1,1} = L$.
2. Randomly generate n_0 initial runs on each partition. For this paper, we used a uniform distribution to select the location of the run on the partition. The number of initial runs for each Φ_i will be dependent upon the allocation method that we use.
3. Allocate more runs to each Φ_i . In order to do this, we aggregate the information for all of the runs in each Φ_i to calculate the sample statistics required by the allocation method that we are using. When coupled with OCBA, we estimate σ_i^2 and calculate $\delta_{b,i}$ for each partition as presented in Section 2. We then allocate additional runs, ΔN_i , to each Φ_i according to Equations (2) and (3).
4. Keeping *k*, the number of partitions, fixed, move $\Omega_{i,j}$ so that each Φ_i has the same number of runs or $N_i = N_j\ \forall\ i,j$. By convention, we establish the boundary by merely equally dividing the distance between the last point in one interval and the first point in the next interval.

5. Repeat steps 3 and 4 until we exhaust the computing budget.
6. After exhausting the computing budget, determine a point to represent the partition having the smallest sample mean performance measure. For this paper, we used the midpoint of the partition but we could use other conventions such as selecting the point in the best partition that has the best performance measure.

A one-dimensional example of the first five steps using OCBA can be seen in Figures 1 - 3 below. The underlying function used is $f(x) = (x-5.5)^2 + 4U(0,1)$ where $x \in (0,10)$ and the optimal solution is located at $x = 5.5$.

1. As shown in Figure 1, the domain is divided into 5 equal intervals.
2. We set $n_0 = 20$ for each Φ_i and randomly distribute (uniform distribution) the initial runs across each separate interval $\left(x_{i,1}, x_{i,2}\right)$.

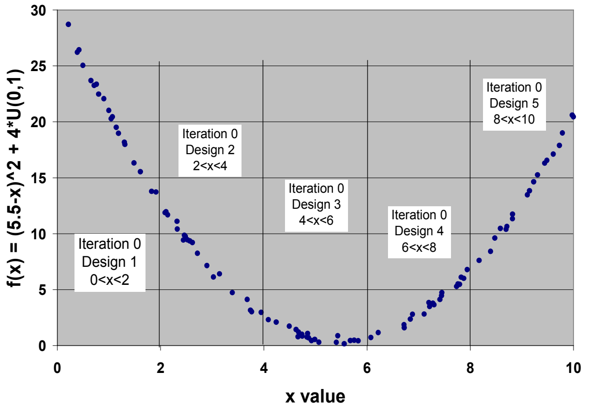

Figure 1: Moving Mesh Example, Steps 1 and 2

3. We calculated the mean and standard deviation of the runs in each interval and used OCBA to allocate a total of 50 more runs across the entire domain. In this case, $\Delta N_1 = 3$, $\Delta N_2 = 12$, $\Delta N_3 = 7$, $\Delta N_4 = 22$, and $\Delta N_5 = 6$. These runs are shown in Figure 2 with the new runs portrayed by triangles.

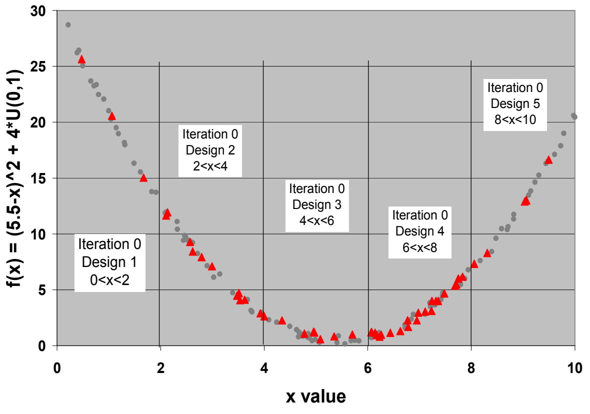

Figure 2: Moving Mesh Example, Step 3

4. Keeping the number of intervals fixed, we then adjusted $\Omega_{i,j}$. Since we have allocated a total of 150 runs among the five designs, we move $\Omega_{i,j}$ so that $N_i = 30\ \forall\ i$. Using this construct, we now have the new intervals shown in Figure 3.

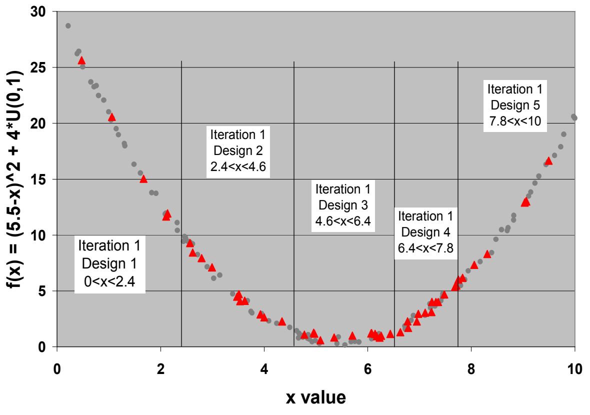

Figure 3: Moving Mesh Example, Step 4

5. From here, we would repeat Step 3 which calculates the new mean and standard deviation for each interval and then uses OCBA to allocate new runs to each interval until we exhaust our computing budget.

3.2 Algorithm Convergence

OCBA, and other methods, concentrate the simulation effort on designs that are promising and do not allocate to designs that are not promising. Mathematically, we do not change the OCBA method described by Chen et al. (2000). Instead of designs consisting of points, we merely compete designs consisting of a group of runs distributed across a partition against each other in order to maximize the probability of identifying the best partition. We obtain our convergence by dynamically redefining the designs. The mesh will get smaller for the partitions that receive more runs allocated in an iteration. Intuitively, we expect these boundaries to converge on the optimal design partition (or point). In fact, as the mesh size for a partition gets smaller, the mean and standard deviation of the runs of the partition begin to resemble those from a point since these measures are less affected by the distribution of the runs across the partition and more heavily influenced by the variance of the underlying function.

However, as Chen et al. (2000) mention, when using OCBA, the number of runs allocated to a particular design increases as the mean the design decreases or the standard deviation of the design increases. It is this property that enables our moving mesh method to maintain a global perspective. By iteratively widening the interval boundaries of a less desirable partition, we expect that the mean of this design will decrease and the standard deviation of the design will increase until it becomes competitive for additional simulation runs.

Figure 4 shows the convergence map for an experiment using the moving mesh method with 10 partitions coupled with OCBA for the example function in Section 3 of this paper.

.

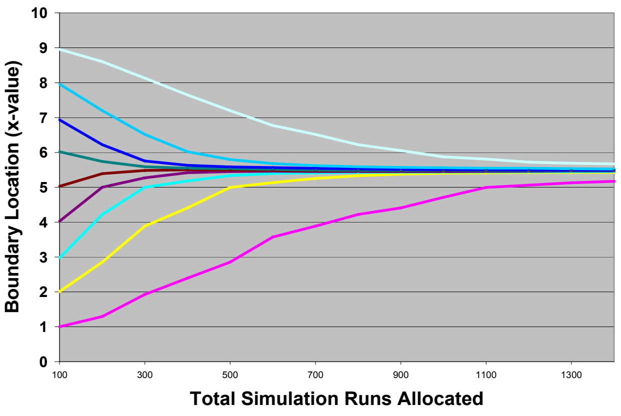

Figure 4: Convergence Example (10 Partitions)

4 NUMERICAL TESTING FRAMEWORK

In this section, we describe how we tested our new moving mesh approach and compared it with a series of numerical experiments it against two allocation procedure: Equal Allocation-Uniform Mesh (EA-UM) and OCBA-Uniform Mesh (OCBA-UM). The next section will provide the results of these experiments.

4.1 Equal Allocation – Uniform Mesh (EA-UM)

This is a brute force method for allocating the number of runs, N_i, to each design. Given a simulation budget T and k designs, we space the k designs uniformly across the domain and allocate the runs equally such that $N_i = T/k$ for each i. The efficiency of this method is dependent upon two inversely proportional parameters: the number of runs allocated to each design and the size of the mesh (number of designs). A small mesh enables the method to potentially have a design close to the optimal solution but a limited computing budget for each design may prevent the method from differentiating the best possible design from others. A large mesh provides enough runs to differentiate the designs under consideration but the best possible solution may be relatively removed from the optimal solution. Through experimentation, we found that this method performed best using about 20 designs during our tests.

4.2 OCBA Allocation – Uniform Mesh (OCBA-UM)

Given a simulation budget T and k designs, we space the k designs uniformly across the domain for this method. However, instead of equally allocating the runs between the designs we use OCBA. Like EA-UM method, the efficiency of this method is dependent upon two inversely proportional parameters: the number of runs initially allocated to each design, n_0, and the size of the mesh (number of designs). Based upon the discussion by Chen et al. (2000) and a little experimentation, we used $n_0 = 5$ for all of our testing. A small mesh enables the method to potentially have a design close to the optimal solution but leaves a limited computing budget for OCBA after providing each design with its initial allocation. However, like EA-UM, a large mesh provides enough runs to differentiate the designs under consideration but the best possible solution may be relatively removed from the optimal solution. Through experimentation, we found that this method enabled us to use a finer mesh than EA-UM and performed best using about 40 - 50 designs during our tests. This method also requires us to specify an additional parameter, Δ, for the total number of runs allocated during each iteration. We used $\Delta = 10$ for all of our testing.

4.3 Moving Mesh – OCBA (MMO)

As previously discussed, the convergence of this method is dependent upon the number of partitions we used. In addition, it is also dependent upon parameters for the allocation method it uses. Since we used OCBA, we had to decide values for n_0 and Δ. Through experimentation, we found that the method performed best during our tests using 8 – 10 partitions, $n_0 = 10$, and $\Delta = 10k$, where k is the number of partitions. Allocating too few runs per iteration does not provide enough new points to cause significant movement in the mesh boundaries. Allocating too many obviously wastes runs that could be concentrated in the most promising partitions.

4.4 Test Procedures

Given the differing parameters for each of these methods, we constructed our experiments to provide a fair comparison. The EA-UM and OCBA-UM have fixed mesh sizes while the MMO method obviously has a dynamic mesh. In order to fairly compare these methods, each of our experiments incorporates a randomly selected optimal solution and our comparison metric is the distance from our best solution to the randomly generated optimal solution. In addition, the methods have varying fixed costs associated with them. To mitigate this difference, we calculate the error for each method during each iteration of the method until the total simulation budget is exhausted. For each experiment, we limited the simulation budgets to 2,000 runs since it was a sufficient number to differentiate between the different methods. We repeat this whole procedure 10,000 times and then calculate the average error obtained for each method during these 10,000 independent applications of each method. This average error obtained from each different procedure serves as our measurement of its effectiveness.

5 NUMERICAL TESTING RESULTS

To initially test our method, we conducted the following experiments on a one-dimensional domain. Each of the numerical experiments was constructed to see if the moving mesh method had convergence problems relative to the two other methods we tested. The first experiment is a baseline experiment where the underlying function is a quadratic. The next experiment has an underlying function with two optimal solutions and that is relatively flat when compared to its variance. The third experiment has two quadratic functions constructed on each half of the domain with one of the critical points having a lower functional value than the other.

5.1 Experiment 1: Convex Function

This experiment is our baseline and uses the following underlying function:

$$f(x) = (x-\xi)^2 + 4U(0,1) \text{ where } \xi \sim U(0,10)$$

and where $x \in (0,10)$.

The optimal solution is ξ so the error for each iteration is measured as $|x_b - \xi|$. Note that for the portions of domain in the interval $(\xi-1, \xi+1)$, the variance of the $4U(0,1)$ term clearly dominates a change in the underlying function $f(x) = (x-\xi)^2$. Figure 5 contains the simulation results for the three methods. We can see that MMO obtains rapid convergence in the first few iterations and then slowly improves after that. Compared to the other two methods, MMO performs the best for when the simulation budget is less than 400 runs, performs about the same as OCBA-UM when the budget is between 400 and 1,000 runs, and performs worse than OCBA-UM after 1,000 runs.

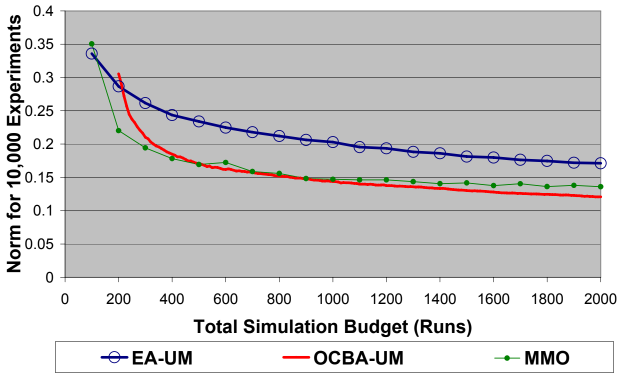

Figure 5: Results for Experiment 1 (Convex Function)

5.2 Experiment 2: Two Optimal Solutions

This experiment is constructed to see if the two optimal solutions cause the MMO method to diverge and uses the following underlying function:

$$f(x) = \cos(x - \frac{3\xi}{10}) + 4U(0,1) \text{ where } \xi \sim U(0,10)$$

and where $x \in (0,10)$.

There are two optimal solutions on the interval (0,10) at $x_a = \pi - 3\xi/10$ and $x_b = 3\pi - 3\xi/10$ so the error for each iteration is measured as the minimum of $|x_b - x_1|$ and $|x_b - x_2|$. For the underlying function $f(x) = \cos(x - 3\xi/10)$, we obtain values in the interval [-1,1] so the variance of the $4U(0,1)$ term again dominates a change in the underlying function. The results of this experiment are very similar to those from the first experiment and are shown in Figure 6. MMO makes most of its convergence in the first 7 iterations of the method (700 total runs allocated). At this point it begins to converge slowly at best and OCBA-UM begins to provide better results.

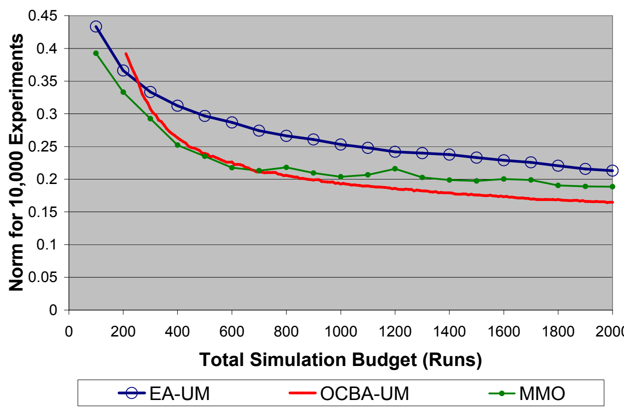

Figure 6: Results for Experiment 2 (Two Optimal Solutions)

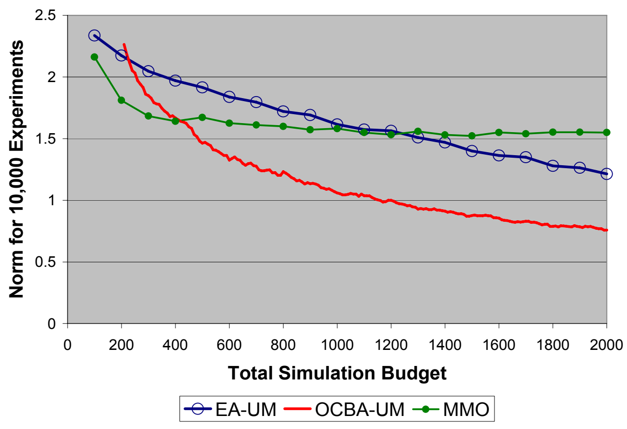

Figure 7: Results for Experiment 3 (Competing Near Optimal Alternate Solution)

5.3 Experiment 3: Competing Near Optimal Alternate Solution

This experiment is an extension of Experiment 2. Instead of determining if MMO can find one of two optimal solutions, we test to see if it can differentiate between an optimal solution and another near optimal solution. We define the underlying difference between the optimal solution and the near optimal solution as the constant λ. For $\xi \sim U(1,4)$ and $x \in (0,10)$, this experiment uses the following underlying function:

$$f(x) = (x-\xi)^2 + 4U(0,1) \text{ when } x \le 5$$

and

$$f(x) = (x-10+\xi)^2 + 4U(0,1) + \lambda \text{ when } x \le 5 .$$

The optimal solution is ξ so the error for each iteration is measured as $|x_b - \xi|$. Just as in Experiment 1, for the portions of domain in the interval $(\xi-1, \xi+1)$, the variance of the $4U(0,1)$ term clearly dominates a change in the underlying function $f(x) = (x-\xi)^2$. The results for this experiment with $\lambda = 0.1$ are shown in Figure 7 and are not encouraging. While MMO again converges rapidly, OCBA-UM performs better than this method after only 400 runs and EA-UM performs better after about 1300 runs.

However, these results are very sensitive to the value we use for the constant λ. If we use $\lambda = 0.05$ instead of $\lambda = 0.1$ as shown above, MMO is better than OCBA-UM until we exceed 800 runs and remains better than EA-UM throughout the 2,000 runs. If we use $\lambda = 0.02$, MMO is clearly superior to OCBA-UM until we allocate about 1800 runs. When compared to EA-UM for when $\lambda = 0.02$, it only takes MMO 300 runs to obtain better results than those obtained by EA-UM in 2,000 runs.

In order to see if modifications to MMO might improve its performance, we repeated Experiment 3 with $\lambda = 0.1$. However, for the MMO method we trimmed simulation runs from the upper portion of the domain for each iteration after we had awarded 200 runs. The results for trimming 30 points each iteration (MMO-T30) and 75 points each iteration (MMO-T75) are compared against our original results for this experiment in Figure 8. This naïve trimming approach clearly improves the performance. However, it introduces a cycling pattern that is clearly evident in the MMO-T75 results. This pattern is introduced because we region we are trimming from reaches the partition that covers the near optimal alternate solution. When we trim from this partition, we have fewer runs allocated to the partition and are more prone to select it as the best partition in error. However, the OCBA method coupled with the moving mesh method recognizes that we need to allocate more runs in this area. This improves our solution until we trimmed from this partition again.

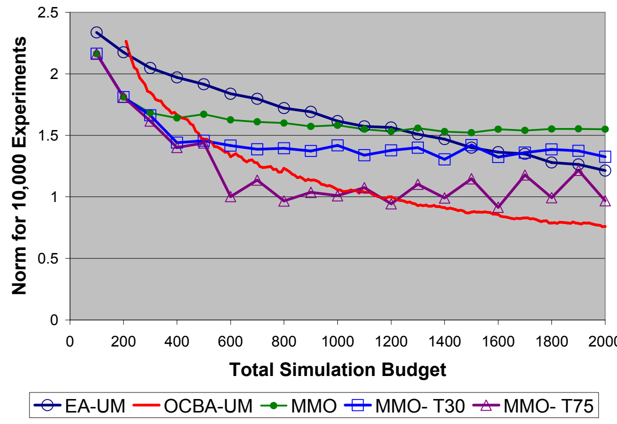

Figure 8: Experiment 3 with MMO Trimmed Methods

6 EXTENDING THE METHOD TO TWO DIMENSIONAL PROBLEMS

6.1 Moving Mesh Algorithm Modifications

The moving mesh method does not change dramatically when moving from a one dimensional problem to a two dimensional problem. The main difference is that, with two dimensional problems, there are numerous methods to construct the mesh. However, the purpose of this paper is to introduce the moving mesh method. Therefore, we used a simple rectangular mesh and a basic accounting scheme to move the mesh between each iteration. We still have k continuous partitions on a domain of length L and width W and we denote

Φ_i : the i-th partition

$\Omega_{i,j}$: the *i,j*-th boundary. For the two dimensional case, each Φ_i will have four boundaries with coordinates $(x_{(i,1)}, y_{(i,1)})$, $(x_{(i,2)}, y_{(i,1)})$, $(x_{(i,1)}, y_{(i,2)})$, and $(x_{(i,2)}, y_{(i,2)})$.

The new algorithm now becomes:

1. Determine k, the number of partitions, and then construct the mesh uniformly across the domain such that the partitions span the entire domain such that $x_{k,2} - x_{1,1} = L$ and $y_{k,2} - y_{1,1} = W$.
2. Randomly generate n_0 initial runs on each partition Φ_i . As in the one dimensional case, we used a uniform distribution to select the location of the run on the partition.
3. Allocate more runs to each Φ_i . In order to do this, we aggregate the information for all of the runs in each Φ_i to calculate the sample statistics required by the allocation method that we are using. When coupled with OCBA, we estimate σ_i^2 and calculate $\delta_{b,i}$ for each partition as presented in Section 2. We then allocate additional runs, ΔN_i , to each Φ_i according to Equations (2) and (3).
4. Keeping k, the number of partitions fixed, move $\Omega_{i,j}$ so that each Φ_i has the same number of runs or $N_i = N_j \ \forall \, i, j$. For this paper, we kept our mesh construction method simple. We first equally divided the runs in the x direction and established boundaries by equally dividing the distance between the last point in one sub-section and the first point in the next sub-section. We then took each x direction sub-section and equally divided the runs in the y direction and again establishing the boundaries equidistant from the last and first points of the resulting partitions.
5. Repeat steps 3 and 4 until we exhaust the computing budget.
6. After exhausting the computing budget, determine a point to represent the partition having the smallest sample mean performance measure.

An example of the first five steps using OCBA can be seen in Figures 9 and 10 below. The underlying function used is $f(x,y) = (x-6.6)^2 + (y-4.3)^2 + 4U(0,1)$ where $x, y \in (0,10)$ and the optimal solution is located at $(x,y) = (6.6, 4.3)$.

1. As shown in Figure 9, the domain is divided into 16 equal partitions.
2. We set $n_0 = 10$ for each Φ_i and randomly distribute (uniform distribution) the initial runs across each partition.

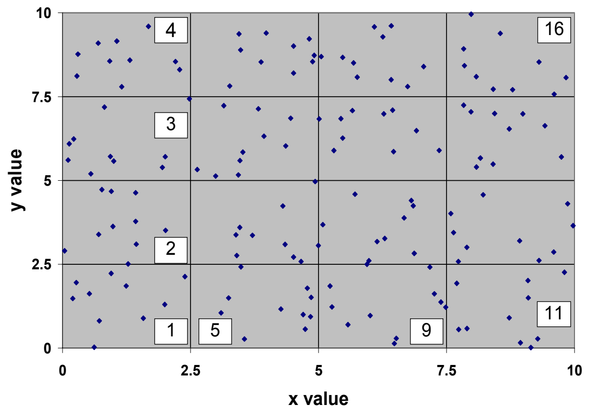

Figure 9: 2D Example, Steps 1 and 2

3. We calculated the mean and standard deviation of the runs in each interval and used OCBA to allocate a total of 160 more runs across the entire do-

main. In this case, the 16 partitions received $\Delta N_i = (2,1,2,1,4,8,3,1,6,84,8,3,4,23,7,3)$ new runs respectively. These runs are shown in Figure 10 with the new runs portrayed by triangles.

4. Keeping the numbers of partitions fixed, we then adjusted $\Omega_{i,j}$. Since we have allocated a total of 320 runs among the 16 designs, we adjust $\Omega_{i,j}$ so that $N_i = 20 \; \forall \, i$. Using this construct, we now have the new partitions shown in Figure 10.

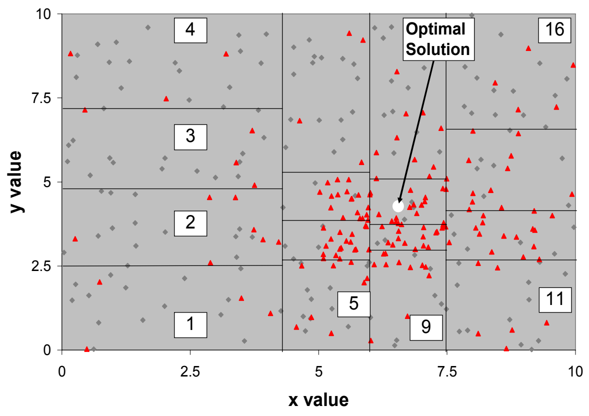

Figure 10: 2D Example, Steps 3 and 4

5. We would then repeat Step 3 which calculates the new mean and standard deviation for each interval and then uses OCBA to allocate new runs to each interval.

6.2 Experiment 4: Two Dimension Convex Function

This experiment is a two dimensional version of Experiment 1 and uses the following underlying function:

$$f(x,y) = (x-\xi_1)^2 + (y-\xi_2)^2 + 4U(0,1) \text{ where } \xi_1, \xi_2 \sim U(0,10)$$

and where $x, y \in (0,10)$.

The optimal solution is (ξ_1, ξ_2) so the error for each iteration is measured as $\sqrt{(x_b-\xi_1)^2+(y_b-\xi_2)^2}$. We again conduct 10,000 experiments and use EA-UM and OCBA-UM for comparison purposes. For EA-UM, we constructed a 10x10, 15x15, and 20x20 grids providing 100, 225, and 400 total designs respectively. By extending our simulation budget for each experiment to 2,200 runs, we also used 10x10, 15x15, and 20x20 grids for the OBCA-UM method. For MMO, we used a 4x4 construct for 16 total partitions.

Figure 11 contains the simulation results for EA-UM 10x10, OCBA-UM 10x10, OCBA-UM 15x15, OCBA 20x20, and MMO 4x4. The EA-UM results were very similar for the three different grids we used and converged very slowly. After applying the initial runs, the OCBA-UM methods converge rapidly to the best possible solution but are ultimately limited in performance by the width of the uniform mesh used. However, uniformly refining the mesh to obtain a better solution comes at a large cost in terms of initial runs for each design. We can see that MMO obtains rapid convergence in the first few iterations and then continues to slowly improve after that. Compared to OCBA-UM, MMO obtains the same results in vastly fewer runs.

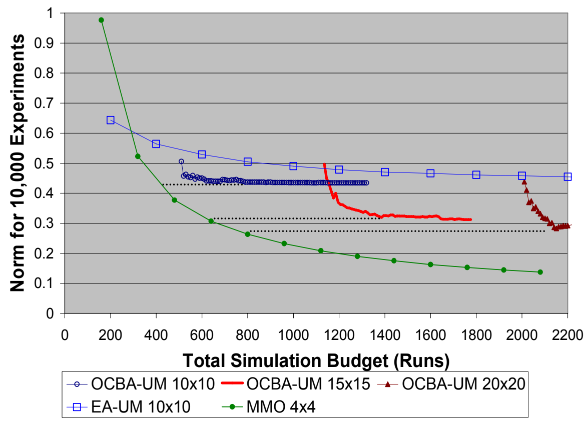

Figure 11: Results for Experiment 4 (2D Function)

7 CONCLUDING REMARKS

In this paper we introduced a moving mesh algorithm for simulation optimization across a continuous domain. During each iteration, the mesh movement is determined by allocating simulation runs to partitions of the domain that are critical in the process of identifying good solutions. The partition boundaries are then adjusted to equally distribute the allocation runs between each partition which reduces the size of promising partitions and increases the size of less desirable partitions. Comparisons with simulation optimization methods using points in the domain on a uniform mesh as designs show that our approach is promising. However, as we refine our approach, we may have to develop trimming heuristics to ensure the method continues to converge and improve the efficiency of our iterative mesh construction as we expand to higher dimensions.

ACKNOWLEDGMENTS

This work has been supported in part by NSF under Grant IIS-0325074, by NASA Ames Research Center under Grants NAG-2-1565 and NAG-2-1643, by NASA Langley Research Center and NIA under task order NNL04AA07T, by FAA under Grant 00-G-016, and by George Mason University Research Foundation.

REFERENCES

Adjerid, S. and J.E. Flaherty. 1986. A moving-mesh finite element method with local refinement for parabolic partial differential equations, *Computer Methods Applications for Mechanics and Engineering*, 55: 3-26.

Arney, D.C. and J.E. Flaherty. 1986. A two-dimensional mesh moving technique for time-dependent partial differential equations, *Journal of Computational Physics*, 67: 124-144.

Chen, C.H., J.Lin, E.Yücesan, and S.E.Chick. 2000. Simulation budget allocation for further enhancing the efficiency of ordinal optimization, *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 10: 251-270.

Law, A.M. and W.D. Kelton. 2000. *Simulation Modeling and Analysis*, McGraw-Hill, Inc.

Shi, L. and S. Olafsson. 2000. Nested partitions method for global optimization, *Operations Research*, 48 (3): 390 – 407.

Swisher, J., S. Jacobson, P. Hyden, and L. Schruben. 2000. A survey of simulation optimization techniques and procedures, *Proceedings of the 2000 Winter Simulation Conference*, ed. J.A. Joines, R.R. Barton, K. Kang, and P.A. Fishwick, 119-126. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

AUTHOR BIOGRAPHIES

MARK W. BRANTLEY is in the Department of Systems Engineering and Operations Research at George Mason University. He received his BS degree in Mathematical Sciences at the United States Military Academy and received MS degrees in Applied Mathematics and Operations Research from Rensselaer Polytechnic Institute. His research interests include simulation and optimization. His e-mail address is mbrantl1@gmu.edu.

CHUN-HUNG CHEN is an Associate Professor of Systems Engineering at George Mason University. He served as the Co-Editor of the *Proceedings of the 2002 Winter Simulation Conference* and the Methodology Analysis track coordinator for the 2003 and 2004 Winter Simulation Conference He received his Ph.D. from Harvard University in 1994. His research interests are mainly in development of very efficient methodology for simulation and optimization, and its application to engineering design and air traffic management. He is a member of INFORMS and a senior member of IEEE. His email address is cchen9@gmu.edu.