

Chapter 7

Large-Scale Simulation and Optimization

We extend the idea of efficient computing budget allocation to large-scale simulation optimization problems. The setting is the same general optimization problem defined in Chapter 1, which is to find a configuration or design that minimizes the objective function:

$$\min_{\theta \in \Theta} J(\theta), \quad (7.1)$$

where θ is a p -dimensional vector of all the decision variables, and Θ is the feasible region. What we meant by “large scale” is that Θ is discrete and large or continuous. In previous chapters, Θ is sufficiently small so that enumeration is possible. When Θ is large, the number of alternative designs is so high that simulating *all* designs becomes very expensive or even infeasible. Some sorts of sampling must be applied to explore the design space Θ .

For these problems, if one is willing to compromise on the optimality, ordinal optimization suggests that we need only to sample and simulate a relatively small number of designs in order to find a good one (cf. [Ho *et al.*, 1992; Dai and Chen, 1997; Luo *et al.*, 2001]). Instead of finding the best design, ordinal optimization concentrates on finding a good enough design within a significantly reduced computation time. Furthermore, ordinal optimization can give a measure

of the quality of the best design found. We denote

$p \equiv \frac{|G|}{|\Theta|}$: the population proportion of good feasible designs,

where G is the population of good feasible designs. $|\Theta|$ is extremely large. Any design in G is a “good” design. Instead of finding the best design in Θ , the goal is to find a good design in G quickly. If we randomly sample a design, then let the probability that this sample is in G be p . If we randomly sample m designs:

$$\begin{aligned} P\{\text{at least one of the selected designs is in } G\} \\ &= 1 - P\{\text{none of the selected } m \text{ designs is in } G\} \\ &= 1 - (1 - p)^m. \end{aligned}$$

If we want to guarantee that at least one of the selected designs is in G with a probability at least as high as P_{sat} , then

$$P_{\text{sat}} = 1 - (1 - p)^m. \quad (7.2)$$

Solving for m in Equation (7.2), one obtains

$$m = \lceil \ln(1 - P_{\text{sat}}) / \ln(1 - p) \rceil. \quad (7.3)$$

Table 7.1 shows how to interpret Equation (7.3) for different values of P_{sat} and p . For example, if one wants the probability of a selected best design within the top 0.1% (99.9 percentile) be as high as 99%, i.e., $P_{\text{sat}} = 0.99$ and $p = 0.001$, one needs only $m = \lceil \ln(1 - 0.99) / \ln(1 - 0.001) \rceil = 4602.8 \approx 4603$ sampling designs. Note that for a given P_{sat} , the number of samples needed goes up roughly by a factor of 10 in order to reduce p by a factor of 10. Also note that this number is independent of the size of the design space $|\Theta|$.

The numbers in Table 7.1 are relatively small, and at any rate much smaller than the size of the design space for any reasonably sized combinatorial problem. Thus ordinal optimization allows us to quickly isolate with a high probability a good design, even with a blind sampling scheme. For further details about ordinal optimization, please refer to Ho *et al.* [2007], which offers an excellent and comprehensive coverage.

Table 7.1. Required number of designs for applying ordinal optimization.

P_{sat}	Top 1%	Top 0.1%	Top 0.01%
50%	69	692	6932
90%	229	2301	23025
99%	459	4603	26049
99.9%	688	6905	69075
99.99%	917	9206	92099
99.999%	1146	11508	115124

Table 7.1 provides a guide on the number of designs which we have to sample in order to guarantee a selection of a good enough design. However, with stochastic simulation noise, we have to simulate all the sampled designs with multiple replications in order to find the best from these sampled designs. For example, if one wants the probability of a selected best design within the top 0.1% to be as high as 99%, one needs to sample only 4603 designs. However, we still have to simulate these 4603 sampled designs and find the best one. This is actually a standard OCBA problem. The OCBA algorithms presented in previous chapters can be applied to enhance the simulation efficiency.

Furthermore, there is room for enhancement beyond uniform random (blind) sampling scheme. In many applications, knowledge of the problem might be used to orient the direction of sampling/search. The information obtained from earlier samples may also be used to guide the search. This can be viewed as biased sampling, which can be achieved by using expert knowledge to reduce the search space, and using metaheuristics framework to explore the design space in a more effective way. The goal is to obtain better results with the same number of sampled designs. In this case, the required number of samples given in Table 7.1 serve as a lower bound when one searches for a good design.

Note that in some problems, the gradient search method can be employed to search the design space when the performance of the designs can be described using a metamodel. However, in this

chapter, our focus is on those problems where such a metamodel cannot be found easily.

7.1. A General Framework of Integration of OCBA with Metaheuristics

When the design space Θ is large, instead of simulating all possible designs, some optimization search methods can be applied to search the design space. By exploring the information in the design space, the expensive enumeration can be avoided. Further, some prescreening can be taken before the search starts so that the problem becomes simpler. We first illustrate the approach by considering a hypothetical simulation optimization example where there are 12 decision variables. Each decision variable has 10 alternative choices. In total, there are 10^{12} designs. It is certainly too expensive to simulate all the 10^{12} designs to find the best design.

Among these 12 decision variables, some of them are not critical or sensitive to the objective of the optimization problem. Those non-critical variables can be identified using some pre-screening techniques (cf. [Goldsman and Nelson, 1998; Wan *et al.*, 2003]) or design of experiments (cf. [Berger and Maurer, 2002; Sanchez, 2005]). Suppose 3 of the 12 decision variables are determined to be non-critical variables and so can be excluded from further optimization process. By focusing on the remaining 9 important variables, we have 10^9 designs. While 10^9 is much smaller than 10^{12} , it is still too expensive to simulate all the 10^9 designs in order to find the best design. Some sort of optimization search algorithms must be applied in order to iteratively search the design space to find the optimum as depicted in Figure 7.1. At each iteration of the search, several designs are selected for simulation. The simulation output information is used to guide the search for next iteration. Such simulation and search iteratively proceeds until a good or optimal design is found.

Such a framework involves two major issues:

- How should we iteratively search the design space Θ ? Metaheuristics aim to address this issue.

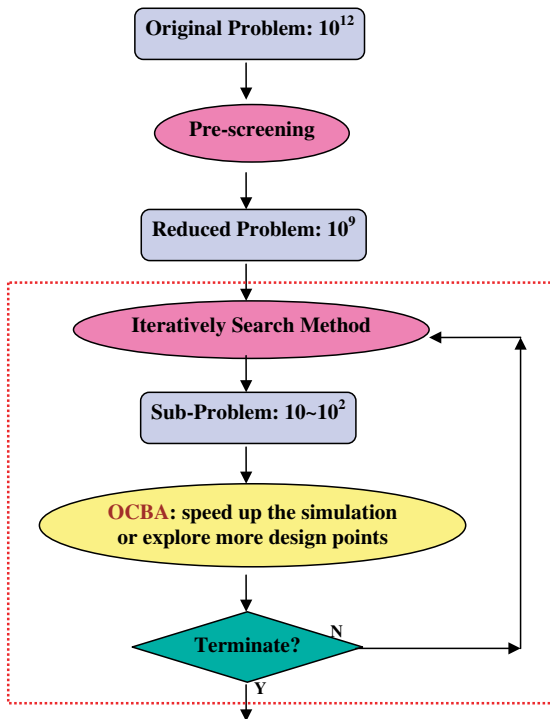


Figure 7.1. An illustrative example of integrating OCBA with search methods for large-scale simulation optimization problems. The box area is further explained in Figure 7.2.

- How should we efficiently simulate those designs generated in each iteration in order to provide the necessary information to guide the new search? Ideally, we want to simulate the generated designs in an intelligent way so that the total simulation cost is minimized.

This book focuses more on the second issue. We would like to find an intelligent way to allocate the simulation budget so that the necessary simulation output can be obtained efficiently to guide the search effectively given a metaheuristic. This budget allocation scheme should consider the information needed in the metaheuristic. OCBA can serve this purpose. The integration framework of OCBA

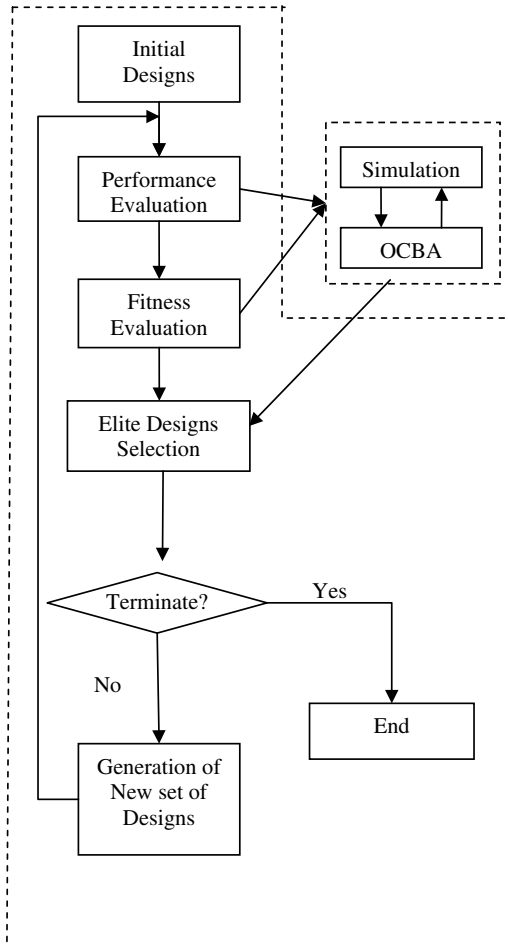


Figure 7.2. An example illustrating how OCBA helps a search method for large-scale simulation optimization problems.

and metaheuristics is shown in Figure 7.2. Most of the metaheuristics start with a population of designs, and elite designs will be selected from this population of designs in order to generate the better population during the search process. Evaluation via simulation is critical in determining the elite designs, and OCBA is able to help to conduct the evaluation effectively by allocating computing budget intelligently. In the remaining sections of this chapter,

we will provide examples of how OCBA can be integrated with the existing metaheuristics. Section 7.2 provides examples of problems with a single objective. The numerical testing in Section 7.3 demonstrates that the efficiency can be enhanced using this framework. Section 7.4 presents integration examples for problems with multiple objectives.

7.2. Problems with Single Objective

In this section, we will give a brief introduction to several metaheuristics or search methods and show how OCBA procedures can be easily integrated with them to enhance efficiency. The metaheuristics considered in this chapter include

- Neighborhood Random Search (Section 7.2.1),
- Cross-Entropy Method (Section 7.2.2),
- Population-based Incremental Learning (Section 7.2.3), and
- Nested Partitions (Section 7.2.4).

As we will show, the integration with OCBA is straightforward and effective. It is worth noting that the applicability of OCBA is not limited to the methods presented here. The same idea is applicable to many other metaheuristics or search methods.

7.2.1. *Neighborhood random search (NRS)*

We start with a simple random search method. This method can be implemented very easily. The primary assumption is that the neighborhood of an elite design has a higher chance of including the optimal design than other neighborhoods. Therefore we want to spend more efforts searching in neighborhoods of elite designs, and less efforts on neighborhoods of non-elite designs.

Specifically, in each iteration, k alternative designs are simulated and then the top- m designs are selected ($m < k$). For the next iteration, a large proportion of candidate designs is generated by sampling the neighborhood of the elite designs. The longer the distance, the smaller the probability a design is sampled. The remaining (smaller)

portion of samples are taken from the entire design space to ensure convergence to the global optimum. The algorithm is summarized as follows.

- Step 1. **Initialization.** Uniformly sample k candidate designs over the design variable space.
- Step 2. **Termination.** Stop, if the stopping criterion is met.
- Step 3. **Evaluation and Selection.** Simulate the sampled k alternatives and select a subset containing the top- m .
- Step 4. **Generation of New Population.** Sample 80% of the new k candidate designs in the neighborhood of the top- m elite designs. Another 20% are taken uniformly from the entire design space. Go back to Step 2.

In this algorithm, Step 3 is the most critical because it provides information to guide the search for the next iteration. It is also the most time consuming step because simulation for the k alternative designs must be performed in order to correctly select the elite subset of the top- m designs. If the simulation cost is not cheap, the computation cost for other steps is negligible, compared with that for Step 3. Then the overall efficiency depends on how efficiently we simulate the candidate designs to correctly select the elite set. Our goal herein is to allocate the computing budget in the most efficient manner so that the elite top- m can be identified. The OCBA- m algorithm presented in Chapter 5 works perfectly here.

7.2.2. *Cross-entropy method (CE)*

The Cross-entropy (CE) method (cf. [Rubinstein and Kroese, 2004]) was originally used for finding the optimal importance sampling measure in the context of estimating rare event probabilities. Later it was developed into a technique for solving optimization problems. It works with a parametrized probability distribution. In every iteration of CE, we will first generate a population of designs from a probability density function (pdf) with a certain parameter. After all the designs in this population have been evaluated (via simulation in our case), we will select the elite designs. These

elite designs will then be used to update the parameters of this pdf, which will be used to generate the population for the next iteration.

In updating the parameters, we need to solve an optimization model which minimizes the Kullback–Leibler divergence (or the cross entropy) between the unknown optimal sampling distribution and the parametrized distribution. This optimization model depends only on the designs in the elite subset. Hence to update the parameters, we need to identify the top- m design efficiently and correctly for each newly generated population.

The algorithm is summarized as follows.

- Step 1. **Initialization.** Initialize a sampling distribution with parameter P .
- Step 2. **Generation of New Population.** Sample k candidate designs using the sampling distribution with parameter P .
- Step 3. **Evaluation and Selection.** Simulate these sampled k alternative designs and select a subset containing the top- m .
- Step 4. **Parameter Updating.** Update P based on the selected top- m by solving the optimization model which minimizes the Kullback–Leibler divergence.
- Step 5. **Termination.** Go back to Step 2 if the stopping criterion is not met.

Like the Neighborhood Random Search method, in this algorithm, Step 3 is the most critical and time consuming step in which k designs must be simulated to correctly identify the elite subset of the top- m designs. The OCBA- m algorithm presented in Chapter 5 can be applied here.

7.2.3. Population-based incremental learning (PBIL)

The PBIL algorithm was developed for binary search problems [Baluja, 1994] and continuous optimization problems [Rudlof and Köppen, 1996]. The PBIL updates the sampling distribution P with a probabilistic learning technique using the estimated mean of an “elite” subset of good candidate designs in each iteration. This

algorithm is almost the same as the CE algorithm, except in Step 4, the PBIL learning principle is applied. The algorithm is summarized as follows.

- Step 1. **Initialization.** Initialize a sampling distribution P .
- Step 2. **Generation of New Population.** Sample k candidate designs using P .
- Step 3. **Evaluation and Selection.** Simulate these sampled k alternative designs and select a subset containing the top- m .
- Step 4. **Parameter Updating.** Update P based on the selected top- m and the PBIL principle.
- Step 5. **Termination.** Go back to Step 2 if the stopping criterion is not met.

Similar to NRS and CE methods, in Step 3 of this algorithm, k designs must be simulated so that the elite subset of the top- m designs can be identified correctly, which takes most of the computational effort. The OCBA- m algorithm presented in Chapter 5 is directly applied here to enhance efficiency.

7.2.4. *Nested partitions*

The Nested Partitions (NP) method has recently been proposed to solve global optimization problems (see [Shi and Olafsson, 2000, 2008; Fu *et al.*, 2008] for more details). The method can be briefly described as follows. In each iteration, a region considered most promising is assumed. We then partition this region into M subregions and aggregate the entire surrounding region into one. Therefore, within each iteration, we only look at $M + 1$ disjoint subsets that cover the feasible space. Each of these $M + 1$ regions is sampled using some random sampling scheme and the estimated performance function values at randomly selected design points are used to approximate a so-called promising index for each region. This index determines which region becomes the most promising one in the next iteration. The sub-region scoring highest on the promising index becomes the most promising region in the next iteration. The new most promising region is thus nested within the last. The method backtracks to a larger region, if

the surrounding region rather than a sub-region is found to have the best promising index. The new most promising region is then partitioned and sampled in a similar fashion. The partitioning continues until singleton regions are obtained and no further partitioning is possible. The partitioning strategy imposes a structure on the feasible region and is therefore important for the rate of convergence of the algorithm. If the partitioning is such that most of the good solutions tend to be clustered together in the same subregions, it is likely that the algorithm quickly concentrates the search in these subsets of the feasible region.

One common definition of promising index is the performance measure of the best sampled design in that subregion. Thus, the most promising region is the one containing the best sampled design among all the subregions. With this choice of a common promising index, determination of the most promising region is equivalent to the determination of the best sampled designs. The algorithm is summarized as follows.

- Step 1. **Initialization.** Let Θ be the promising region.
- Step 2. **Partition.** Partition the current most promising region into M sub-regions and aggregate the surrounding region into one.
- Step 3. **Generation of New Population.** Randomly sample q designs from each of the subregions and from the aggregated surrounding region; $k = (M + 1)q$.
- Step 4. **Evaluation and Selection of the Most Promising Region.** Simulate these k sampled designs and select the best among the k designs. The subregion containing this best design becomes the most promising region.
- Step 5. **Termination.** Stop, if the stopping criterion is met.
- Step 6. **Determination of Further Partition or Backtracking.** If one of the M subregions becomes the most promising region, the algorithm moves to this region for further partitioning in the next iteration. If the surrounding region contains the best design, the algorithm backtracks to a larger region. If the new promising region contains more than one design, go back to Step 2.

Step 4 is to simulate and select the best from the set of sampled designs, comprising the union of all sampled designs from each disjoint (sub)region. It is essentially a small and discrete simulation optimization problem discussed in earlier chapters. This step involving many replications of simulation usually consumes most computational effort. Our goal herein is to allocate the computing budget in a most efficient manner so that the best design can be identified. Hence, the OCBA method presented in Chapter 3 can be applied directly. Details about this integration of OCBA and NP can also be found in Shi and Chen [2000].

7.3. Numerical Experiments

To illustrate improvements in efficiency, we carried out numerical experiments for two typical optimization problems using some procedures presented in this chapter. Most of the numerical setting is the same as those presented in Chapters 4 and 5. Further details can also be found in Chen *et al.* [2008]. Three search methods are tested:

- Neighborhood Random Search (NRS),
- Cross-Entropy Method (CE),
- Population-based Incremental Learning (PBIL).

The OCBA-m procedure is integrated into each of the three search algorithms, and the resulting performance of the algorithm is compared with the same algorithm without using OCBA, in which case we apply equal simulation of all searched candidate designs.

Experiment 7.1. Griewank function

The Griewank function is a common example in the global optimization literature (cf. [Fu *et al.* 2006]), given in two-dimensional (2-D) form by

$$f(x_1, x_2) = \frac{1}{40}(x_1^2 + x_2^2) - \cos(x_1)\cos\left(\frac{x_2}{\sqrt{2}}\right) + 1,$$

where x_1 and x_2 are continuous and $-10 \leq x_1 \leq 10$, $-10 \leq x_2 \leq 10$. The unique global minimum of this function is at $(x_1^*, x_2^*) = (0, 0)$ and $f(x_1^*, x_2^*) = 0$. The additive noise incurred in stochastic simulation

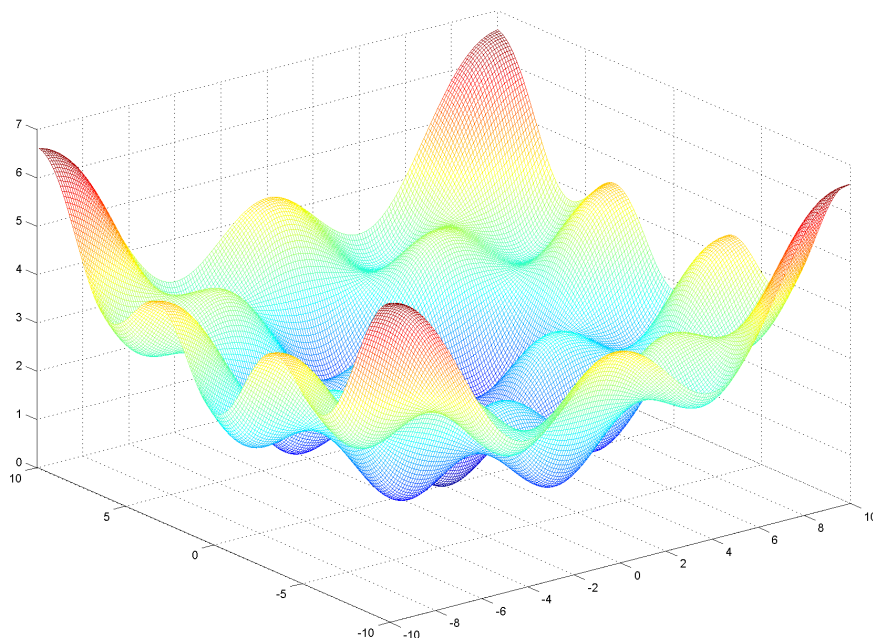


Figure 7.3. 2-D Griewank function tested in Experiment 7.1.

is $N(0, 1^2)$. Figure 7.3 gives an illustration of this function without simulation noise.

In numerical implementation, the stopping criterion for the evaluation-and-selection problem in Step 3 is when the posterior $APCS_m$ given by Lemma 3.2 is no less than $1 - 0.2 \cdot \exp(-q/50)$, where q is the iteration number. We set $k = 100$ and $m = 5$. In comparing the procedures, the measurement of effectiveness used is the average error between the best design thus far and the true optimal design over 200 independent experiments. The results are shown in Figure 7.4. The thick lines indicate the performance with OCBA-m for different optimization algorithms, while the thin lines show the performances without OCBA-m. Lines with different patterns represent the use of different optimization search algorithms.

We see that the optimality gap decreases for all procedures as the available computing budget increases. In this example, NRS performs better than PBIL, which does better than CE. However, OCBA-m significantly enhances the efficiency for all three search methods. For

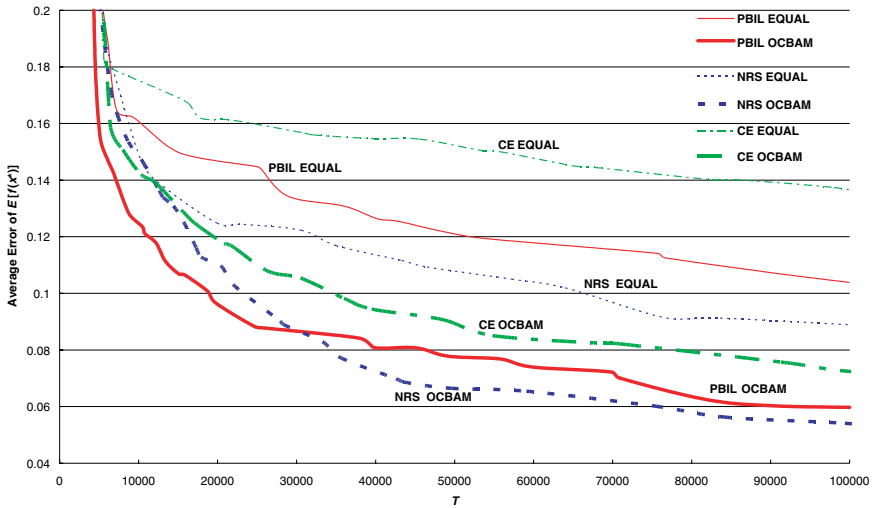


Figure 7.4. Performance comparison for three optimization search algorithms with and without OCBA-m for Experiment 7.1.

example, with integration of OCBA-m, NRS can achieve an average error of 0.1 using a computation cost of 22,800. Without OCBA-m, NRS spends a computation cost of 68,500 to achieve the same level of error. Similarly, the computation costs for PBIL to reduce the average error to 0.12 with and without OCBA-m are 11,500 and 53,700, respectively. The speedup factor of using OCBA-m is even larger if the target level of optimality is higher.

Experiment 7.2. Rosenbrock function

The Rosenbrock function is another common example in the global optimization literature (e.g., [Fu *et al.*, 2006]). It is a non-convex function with a “banana-shaped” valley given in 2-D by

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2,$$

where x_1 and x_2 are continuous and $-5 \leq x_1 \leq 5$, $-5 \leq x_2 \leq 5$. The global minimum of this function is at $(x_1^*, x_2^*) = (1, 1)$ and $f(x_1^*, x_2^*) = 0$. The additive noise incurred in stochastic simulation is $N(0, 10^2)$. Figure 7.5 gives an illustration of this function without noise.

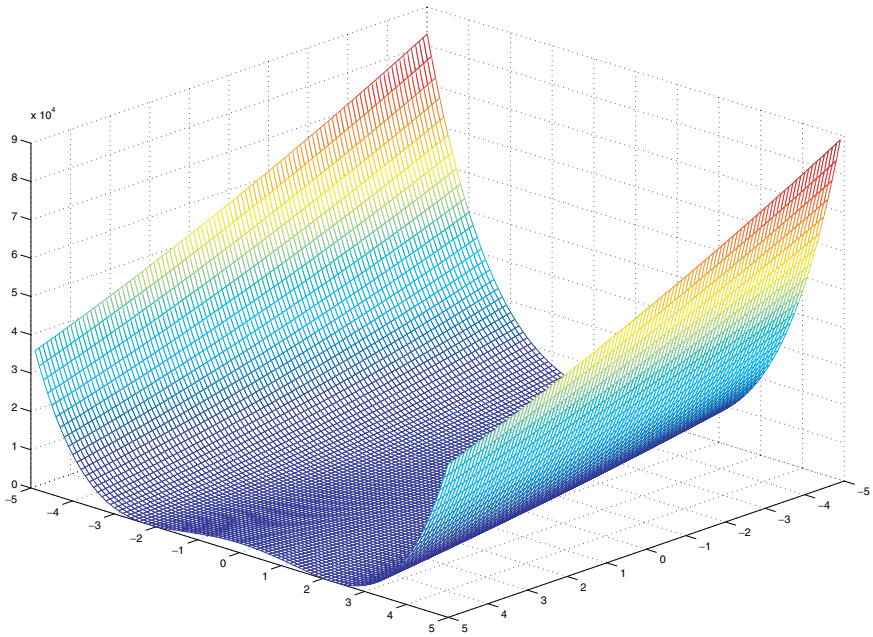


Figure 7.5. 2-D Rosenbrock function tested in Experiment 7.2.

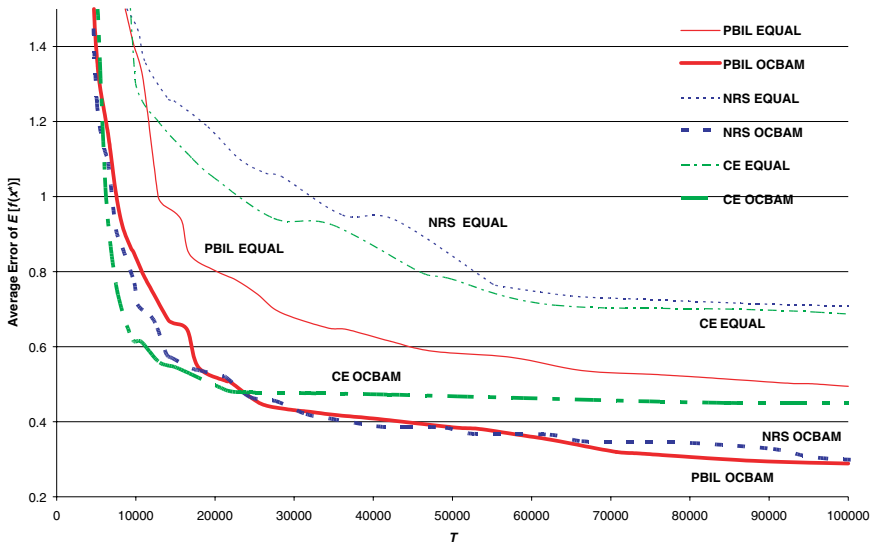


Figure 7.6. Performance comparison for three optimization search algorithms with and without OCBA-m for Experiment 7.2.

The numerical setting is the same as that in Experiment 7.1. The test results are shown in Figure 7.6. Unlike Experiment 7.1, the PBIL method has the best performance in this example. Although the order of optimization methods are different from that in Experiment 7.1, the level of efficiency enhancement using OCBA-m is very similar.

While different optimization algorithms perform differently in the preliminary numerical examples, the OCBA allocation significantly enhances the computational efficiency for each individual search algorithm.

7.4. Multiple Objectives

Section 7.2 deals with problems with a single objective function to optimize. In this section, we turn our attention to cases where we are facing multiple objectives, and we would like to find Pareto optimal solutions. The search method designed for the single-objective problem can be directly employed in this case except that we have to redefine the fitness function. In Chapter 6, we have defined the probability of non-dominating for each design as ψ , which can be used as the fitness function to rank the designs for multiple-objective problem. However, as there is no closed-form expression for ψ , we will use its upper bound to approximate this fitness function.

In the next two sub-sections, we will show how to integrate MOCBA presented in Chapter 6 with the metaheuristics. The metaheuristics considered here are

- Nested Partitions (Section 7.4.1), and
- Evolutionary Algorithm (Section 7.4.2).

7.4.1. *Nested partitions*

The method of NP has been introduced in Section 7.3.4. Two changes in the integrated NP procedure must be made. First, the performance measure for each design is changed to the probability of non-dominating, and so, like the single-objective case, the promising index of a subregion becomes the highest probability of non-dominating among all the designs in that subregion. Second, we need to keep an elite set which consists of all Pareto optimal solutions.

The algorithm is summarized as follows.

- Step 1. **Initialization.** Let Θ be the promising region, and Ω be the elite set.
- Step 2. **Partition.** Partition the current most promising region into M sub-regions and aggregate the surrounding region into one.
- Step 3. **Generation of New Population.** Randomly sample q designs from each of the subregions and from the aggregated surrounding region; $k = (M + 1)q$.
- Step 4. **Evaluation and Selection of the Most Promising Region.** Simulate these k sampled designs and select the Pareto optimal designs from them. Compare the new Pareto optimal designs with the designs in the elite set obtained in the previous iteration, and update the elite set so that it only consists of updated Pareto optimal solutions thus far. The subregion containing the best sampled design (the one with the highest probability of non-dominating) becomes the most promising region.
- Step 5. **Termination.** Stop, if the stopping criterion is met.
- Step 6. **Determination of Further Partition or Backtracking.** If one of the M subregions becomes the most promising region, the algorithm moves to this region for further partitioning in the next iteration. If the surrounding region contains the best design, the algorithm backtracks to a larger region. If the new promising region contains more than one design, go back to Step 2.

Step 4 of the algorithm is to simulate all the designs in order to identify the Pareto optimal designs. This step consumes most computational effort and the MOCBA method presented in Chapter 6 can be applied directly here to improve the simulation efficiency. Details about this integration of MOCBA and NP can also be found in Chew *et al.* [2009].

7.4.2. Evolutionary algorithm

Evolutionary Algorithm (EA) is an adaptive heuristic search algorithm which simulates the survival of the fittest among individuals

over consecutive generations for optimization problem solving. Based on an initial population randomly generated, at each generation, EA evaluates the chromosomes and ranks them in terms of their fitness; the fitter solutions will be selected to generate new offspring by recombination and mutation operators. This process of evolution is repeated until the algorithm converges to a population which covers the non-dominated solutions. EA has been successfully applied in solving multi-objective problems [Fonseca and Fleming, 1995; Hanne and Nickel, 2005].

To make EA work well for simulation-based problems where stochastic noise is a main concern, MOCBA is needed mainly in the following two aspects: fitness evaluation and elite population formation. The integration algorithm is summarized as follows.

- Step 1. **Initialization.** Randomly sample k designs from the design space Θ to form an initial population.
- Step 2. **Evaluation and Selection of the Pareto Optimal Designs.** Simulate these k sampled designs and select the Pareto optimal designs from them. Compare these Pareto optimal designs obtained in this iteration with the designs in the earlier elite set, and update the elite set so that it only consists of updated Pareto optimal solutions thus far.
- Step 3. **Elite Set Updating.** Compare the new Pareto optimal designs obtained in this iteration with the designs in the earlier elite set, and update the elite set so that it only consists of updated Pareto optimal solutions thus far.
- Step 4. **Termination.** Stop, if the stopping criterion is met.
- Step 5. **Generation of New Population by Crossover and Mutation Operation.** Randomly select two candidate designs from the elite set to perform the crossover operation. Repeat this until there are enough candidate designs to form the new population. Randomly select several candidates from this new population, and perform mutation operations. Go back to Step 2.

Step 2 of the algorithm involves many replications of simulation for all designs in the new population and so consumes most of the

computational effort. The MOCBA method presented in Chapter 6 can be applied directly here to improve simulation efficiency. Details about this integration of MOCBA and EA can also be found in Lee *et al.* [2008].

7.5. Concluding Remarks

As shown in all the previous sections, OCBA, OCBA-m, and MOCBA can be applied directly in the search methods. This is because these search methods need to find the best, top- m or Pareto optimal designs in order to determine the new search direction for the next iteration. However, some search methods may require more information when determining the new search direction, in which case the existing OCBA algorithms might not be sufficient, and so newer OCBA algorithms must be developed according to what is needed in the specific search method. In Section 8.2, we will provide such an example to illustrate how a new OCBA algorithm is developed. This example utilizes an extended cross-entropy method.

When the search method is chosen and its stopping criterion of the evaluation in each iteration is determined, the quality of the selected designs used to guide the new search is pretty much set. It implies that the efficacy of the search is fixed. The benefit of OCBA in this integration is the savings of computing time in evaluating the sampled designs. This saving in time can be utilized in a few ways:

- Terminating the search earlier,
- Using the time saved to generate more sampled designs in an iteration, or
- Running the search algorithm with more iterations.

Either of these should improve the search quality or efficiency, resulting in a better design found eventually.

In this chapter, the simulation optimization problem can be decomposed into a search part and evaluation part because the stopping criterion of the evaluation part is fixed. OCBA is applied to the evaluation part to ensure the selection quality is reached

in a minimum time. However, there exists a trade-off between the selection quality and search quality given a fixed total computing budget. Overall, there are three issues.

1. How many iterations should the search algorithm run?
2. How many designs should be sampled in each iteration?
3. How much budget should be allocated to evaluate the sampled designs?

Ideally we want to allocate the computing budget in an optimal way by considering all three issues together. Due to the complex nature of this problem, it remains a research topic.