Optimal Allocation and Splitting Among Designs in Rare Event Simulation

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Ben W. Crain

Director: John F. Shortle, Professor
Department of Systems Engineering and Operations Research

Summer Semester 2013
George Mason University
Fairfax, VA

Table of Contents

List of Tables

List of Figures

Abstract

OPTIMAL ALLOCATION AND SPLITTING AMONG DESIGNS IN RARE EVENT SIMULATION

Ben W. Crain, Ph.D.

George Mason University, 2013

Dissertation Director: Prof. John F. Shortle

This dissertation develops efficient algorithms, in theory and in implementation, for selecting, via simulation, the best design, or system, from a set of designs, where "best" is the design with the smallest probability of some (generally undesirable) outcome. Compared to standard techniques these algorithms improve the efficiency of simulation when the (undesirable) outcomes have very small probabilities, on the order of 1.0E-6, or smaller. Such outcomes are "rare events". The algorithms could also be used to estimate non-rare probabilities, although, in that case, their advantages over techniques not geared toward rare events diminish.

The designs in question differ in construction, or in the values of their parameters, but are such that their operations can be simulated as stochastic processes which terminate in a non-rare event (a set of non-rare outcomes), or a rare event (a set of rare outcomes). The task is to estimate, efficiently, the probabilities of the rare events, in order to select the design with the smallest one. Efficient algorithms are those which

produce estimators of the rare event probabilities with acceptable variances, within acceptable computational times. I do not define "acceptable variances". The focus is on algorithms which achieve better results, compared to standard methods, for a given amount of computational time (a given computing budget constraint). Better results are achieved by (approximately) maximizing the Probability of Correct Selection (PCS) of an algorithm, subject to a computing budget constraint. PCS is the probability that the algorithm will correctly identify the best design.

Simulation algorithms against which the new algorithms are compared include simple Monte Carlo (MC), Optimal Computing Budget Allocation (OCBA), fixed-effort Splitting, and the Optimal Splitting Technique for Rare-Event Simulation (OSTRE). These algorithms are reviewed, prior to the development of the two new algorithms: Single Optimization and OCBA+OSTRE.

The major contribution of this dissertation is the theoretical development and practical implementation of these new algorithms. The mathematical equivalence of these two new methods (in the sense that they attain, in theory, the same maximum PCS) is proven, and their computational complexities are compared. Numerical testing illustrates that they can out-perform standard techniques, and suggests that OCBA+OSTRE is better, in practice, than Single Optimization.

1. Introduction

This dissertation answers the question: How should simulations be conducted in order to optimize the probability of selecting the best system, or design, from a set of designs, when "best" is defined as the design with the smallest probability of the occurrence of a rare event?

1.1 Examples of Rare Events

A rare event is the occurrence of an event, generally undesirable, of very small probability. Small is not precisely defined, but typically means on the order of 1.0E-6, or smaller. Rare events can occur in many kinds of systems, throughout science, engineering, and socio-economic processes. See Rubino & Tuffin (2009) for a survey of some of these areas.

An early application of rare event estimation arose in nuclear physics.[1] Booth (2009) discusses the current need to estimate the probability, on the order of 1.0E-8 to 1.0E-10, of nuclear particles penetrating protective shields in nuclear reactors. Transportation systems provide other examples. Civil aircraft regulations require very low failure probabilities. Small probabilities must be estimated in telecommunication systems which operate by sending message units through network nodes. Node overflows should be rare events. Financial systems may be modeled as stochastic

[1] See Kahn & Harris, 1951, and Hammersley & Handscomb, 1964.

processes in which undesirable events (bank run, catastrophic loss of capital) should be rare. Cascading failures across a power grid can also be modeled as a stochastic process, with blackout as a terminating rare event.

1.2 Estimating Rare Events

Estimating the probability of a rare event is difficult. The difficulty is compounded when several such probabilities must be estimated, one for each of several (perhaps many) designs, in order to select the best design. Simple Monte Carlo[2] simulation is generally too inefficient for that task. Inefficient in the sense that generating estimates with acceptably small variances will often require excessive amounts of computing time. Bucklew (2004) calculates that, if the rare event probability is on the order of 1.0E-6, with 95% confidence that the error is no greater than 5%, the number of required simulation runs would be on the order of 1.6E+9. A computer capable of doing 1000 runs a second would require about 18 days to complete the simulation.

Shortle and L'Ecuyer (2010) also present calculations illustrating how impractical simple Monte Carlo estimation can be. If the probability to be estimated is 1.0E-9 and the computer can do 1000 runs per second, achieving a relative error of 10% (defined as the standard deviation of the estimator divided by the true probability) would require about 3 years of continuous computation.

[2] Monte Carlo is a general term that may, depending on the context, signify a broad range of simulation techniques. It could mean simulation per se. By Monte Carlo I mean the simple method of conducting a number of "runs" (realizations, or sample paths, of the process) and estimating probabilities by counting outcomes.

Rare event probabilities less than 1.0E-6 are not uncommon. Bucklew notes that the problems arising from simulations of this magnitude may go beyond the required computing time. Such a large number of runs "will impose severe demands on the random number generator. Unfortunately (for the simulation designer) error probabilities of this order are typical in digital communications systems and many other systems of engineering and scientific interest."

For rare event estimation special simulation techniques are usually required, techniques that are much more efficient that Monte Carlo. The two main approaches to rare event estimation are called importance sampling and splitting. Importance sampling[3] works by appropriately changing the sampling distribution to make the rare event more probable, and thus easier to estimate, then re-scaling the result to recover the correct probability. It is not pursued in this research, which focuses on the occurrence of rare events in the evolution of stochastic processes. Importance sampling can be applied to processes such as Markov chains, by making the importance sampling change of measure depend on the underlying process. But that is considerably more difficult to implement than the splitting technique, on which this research focuses.

Splitting is the general name of a corpus of techniques, all of which involve partitioning the sample space of the stochastic process in a manner that permits the efficient estimation of probability estimators for subsets (called "levels") of the sample space. An unbiased estimate of the rare event probability can then be constructed from

[3] See Heidelberger, 1993, or Glynn, 1994.

these level estimators, as described in chapter 4. For an overview of splitting techniques see L'Ecuyer *et. al*., 2006, and L'Ecuyer *et al*., 2009.

Splitting techniques can work well (can be quite efficient) when applied to a single rare event probability. They can also be applied to the problem of selecting the best among several designs, though efficiency suffers when the computational effort must be spread across several designs. This dissertation offers new methods for attaining greater efficiency in the selection of the best of several designs by optimizing the allocation of computational effort across designs and within designs. Exactly what it means to optimize "across" and "within" designs will be made clear in the course of this research.

1.3 The Contribution of This Dissertation

There is, in the current literature, a good method for selecting, via simulation, the best design when the probabilities in question are not rare. This method --- based on the theory of the Optimal Computing Budget Allocation (OCBA) – optimizes the allocation of computing effort across designs. (See Chen & Lee, 2011).

There is, in the current literature, a good method for estimating a single rare event probability (for a single design). This method – the Optimal Splitting Technique for Rare Event (OSTRE) simulation – optimizes the allocation of computing effort "within" a design, among the levels of a single design[4]. (See Shortle, Chen, *et al*, 2012.)

[4] The technique of splitting a single design into "levels" will be clarified in chapter 4.

But there is, in the current literature, no method, no algorithm, for simultaneously optimizing computing effort across and within an arbitrary number of designs, in order to select the best design. None is needed when the probabilities are not rare. For that task OCBA works well. But OCBA will not suffice when the probabilities are rare.

This dissertation provides the theory, and the algorithms, for extending efficient rare event estimation to an arbitrary number of designs, by optimizing computing effort across and within designs. It is an extension of OCBA, which optimizes across but not within designs, and of OSTRE, which optimizes within but not across designs. (The basic framework from which OSTRE is derived permits a straightforward extension to two designs, but not beyond two.) This research develops new theory and techniques to extend the optimization of computational effort across and within designs to an arbitrary number of designs and levels. It develops two new methods, called Single Optimization and OCBA+OSTRE, and proves that they are mathematically equivalent.[5] It shows that OCBA+OSTRE is more appealing than Single Optimization, in that it is easier to implement and requires less computational overhead. It tests the new methods, on a specific test model, and shows that, in these tests, they outperform standard splitting (including OSTRE) in selecting the best design. The two methods are also tested against each other. OCBA+OSTRE is shown to be better, on the test model, than Single Optimization.

[5] An earlier version of this research developed these two methods, and showed their equivalence, for the simplest case of two designs, two levels for each design. See Crain et.al, 2011.

I conclude this research with a conjecture as to why OCBA+OSTRE appears to work better than Single Optimization, despite their mathematical equivalence.

1.4 Organization of This Dissertation

Chapters 2 through 5 set the stage for the new methods I develop by reviewing existing methods, discussing their strengths and weaknesses, and testing them on the test model. There is nothing novel in this review. It draws on existing literature, referenced in the successive chapters. These chapters provide, for readers unfamiliar with this terrain, the essential background for understanding the contributions of this dissertation. Those contributions are developed in chapters 6 through 9.

1.5 Chapter Summaries

Chapter 2, The Probability of Correct Selection: Choosing the Best, defines the problem, explains rare events and stochastic processes, introduces Monte Carlo methods, introduces the model on which tests throughout the research are conducted, and illustrates the poor performance of Monte Carlo in estimating rare events.

Chapter 3, Review of OCBA, addresses the Optimal Computing Budget Allocation theory and algorithm, reviews its derivation, and discusses its application to rare event estimation. OCBA is tested against Monte Carlo. See Appendix 2 for a review of relevant literature on OCBA.

Chapter 4, Review of Splitting, explains the splitting method for estimating rare event probabilities, and discusses its application to the selection of the best design. Splitting is tested against OCBA. See Appendix 3 for a review of relevant literature on Splitting.

Chapter 5, Review of OSTRE, presents the derivation of the Optimal Splitting Technique for Rare-Event Simulation, the assumptions it requires, and its application to the selection of the best design. OSTRE is tested against Splitting.

Chapter 6, Single Optimization, develops the theory and implementation of an algorithm designed to maximize the Probability of Correct Selection by optimizing, in a single set of calculations, the allocation of computing effort across all the splitting levels of all the designs. The assumptions and approximations underlying this algorithm are clarified and justified. The implementation is treated in some detail, and a complexity analysis of the algorithm is undertaken. Single Optimization is tested against Splitting and against OSTRE.

Chapter 7, OCBA+OSTRE, proposes the major innovation of the dissertation. Single Optimization, developed in chapter 6, is an application of standard optimizing techniques to a new problem, the selection of the best design. OCBA+OSTRE is a new technique applied to this new problem. It combines the OCBA algorithm with the OSTRE algorithm in a novel two-step technique (an OCBA step followed by an OSTRE step). This combination requires more than a simple adjoining of the two approaches. New theory and new implementation must be developed to make the OCBA and OSTRE steps consistent with each other, and to make OCBA+OSTRE mathematically equivalent to Single Optimization. A complexity analysis of OCBA+OSTRE is conducted.

Chapter 8, OCBA+OSTRE vs Single Optimization: Mathematical Equivalence, proves that the two methods are mathematically equivalent in the sense that, in theory,

their solutions attain the same optimum allocation of computing effort among all levels of all designs.

Chapter 9, OCBA+OSTRE vs Single Optimization: Implementation**,** tests OCBA+OSTRE against Single Optimization. Though chapter 8 proved them to be mathematically equivalent, these tests suggest that OCBA+OSTRE may be better in practice. I conjecture that OCBA+OSTRE is superior to Single Optimization not only because it is easier to implement and faster in execution, as a comparison of their complexity analyses suggests, but also because it can be implemented more "accurately". Exactly what this means, and why OCBA+OSTRE may be more accurate, is clarified in chapter 9.

Chapter 10, Conclusions, summarizes the main characteristics of the various simulation techniques reviewed and developed in this research. A final graph plots the competitive techniques – Splitting, OSTRE, Single Optimization, and OCBA+OSTRE – showing their relative performance in maximizing PCS over a common budget constraint.

2. The Probability of Correct Selection: Choosing the Best

2.1 The Rare Event of a Stochastic Process

Consider a stochastic process that commences in an initial set (a set of possible initial states) and terminates in a set of states *S* (which could include the initial set), or in a set of states *R*. Let the probability of the process terminating in *R* be very small. *R* is a rare event. The probability of terminating in *S* is not small. The task is to estimate, efficiently, via simulation, the probability of termination in *R*.

In theory stochastic processes can be infinite, never terminating. But in practice we are generally interested in finite processes. We can only simulate in finite time. I consider only processes which terminate the first time they enter *R*, though the algorithms developed could be adapted to processes which terminate in *R* (or *S*) after transiting *R* (or *S*) a finite number of times.

The task is to estimate, efficiently, the probability of termination in *R*. It could be estimated by simple Monte Carlo, which can be quite effective when the probability is not rare. If the process terminates in the rare event a "hit" has occurred. If not there is no hit. The estimator of the probability of the rare event is the number of hits divided by the number of runs.

It is useful to illustrate an abstract concept, such as a stochastic process, with a concrete example. Throughout this dissertation I employ a specific model of a stochastic process, for illustration and for testing algorithms. I introduce it here.

2.2 The Test Model

The model is a queue with two tandem servers.

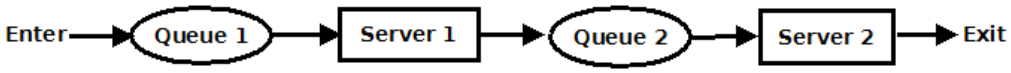

Figure 1: Two Server Tandem Queue

If queue 1 is empty arriving customers go directly to server 1. Otherwise they add to the customers in queue 1. Upon finishing server 1 they enter server 2, if empty. Otherwise they add to the customers in queue 2. Upon finishing server 2 they exit the system. The queues increase or decrease as customers arrive and as they complete service, on a first-in-first-out basis. At any point the system could be in any of these states:

--- System empty, no one in either server or queue.

--- A customer in one or both servers, both queues empty.

--- A customer in one or both servers, one or both queues not empty.

The system can be completely defined by two state variables. Designate queue 1 and server 1 as subsystem 1, queue 2 and server 2 as subsystem 2. Variables S1 and S2 are the numbers of customers in subsystems 1 and 2. If S1 = 0, subsystem 1 is empty. If

S1 = 1, there is a customer in server 1, none in queue 1. (Queue 1 is empty because a customer facing an empty server instantly enters the server.) If S1 = n > 1, there is 1 customer in server 1, n-1 customers in queue 1. These characterizations apply equally to subsystem 2.

(S1, S2) designates the current state of the system. From any (S1, S2) the process has three possible transitions (moves): to (S1+1, S2) if the next event is an arrival; to (S1-1, S2+1) if the next event is completion of service in server 1; to (S1, S2-1) if the next event is completion of service in server 2.

The system could be unbounded in S1 or S2 , or both. To create a terminating rare event I bound it at 50, in both variables. The process terminates at the rare event if S1 = 50 or S2 = 50. It terminates at the non-rare event if it returns to (0, 0). The system, with S1 on the horizontal axis, S2 on the vertical axis, may be depicted as:

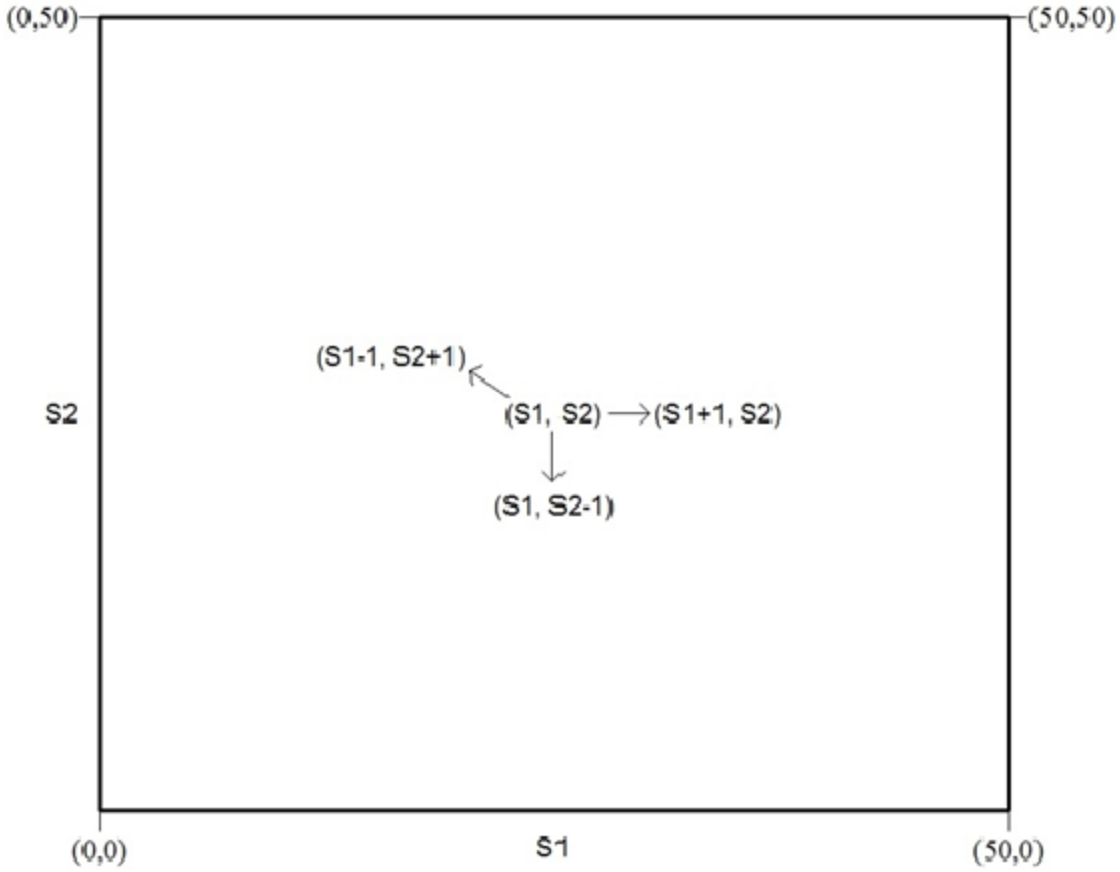

Figure 2: Test Model, with Transitions

When the process is on the horizontal or vertical axis (S1 = 0 or S2 = 0) only two transitions are possible, since neither S1 nor S2 can be -1. If the process is at (0, 0) only one move is possible, to (1, 0).

Conceptually the origin of a run (sample path of the process) is (0, 0). But, since the only move possible from (0, 0) is to (1, 0), in practice the run starts at (1,0), it being pointless to code a deterministic move. If a run returns to (1, 0) it continues. It terminates only at (0, 0) or in the rare event (the S1=50 or S2=50 boundary).

Examination of the possible transitions reveals that the process can never reach (49, 50) or (50, 50). It could reach (49, 50) only from (48, 50) or (50, 49), but those are terminating states. Similar reasoning implies it cannot reach (50, 50).

This system is an absorbing Markov chain, with absorbing states (0, 0) and any state with S1 = 50 or S2 = 50, except (49, 50) and (50, 50). As an absorbing Markov chain it can be solved exactly. Its exact probabilities (and more) can be calculated analytically. Probabilities can be calculated not just for the entire design (the design probabilities), but also for the level probabilities that arise under splitting. (See chapter 4 on splitting for an explanation of levels and level probabilities.) In addition, the average number of transitions from the origin to absorption (termination of a run) can be calculated. Winston (2004, p. 945) explains how these calculations are made. (See Appendix 1 for the exact probabilities for all designs.)

The interplay of three random variables determines the transitions: the time until the next arrival and the times until completion of service in the servers. They are distributed exponentially, with rates λ, μ_1, μ_2, respectively, and expected values:

$$\text{E [time until next arrival] } = \frac{1}{\lambda}\ , \quad \text{E [time until completion of service } k \text{] } = \frac{1}{\mu_k}.$$

The parameters λ, μ_1, μ_2 can be set so that termination on the boundaries is a rare event, termination at the origin a non-rare event. Sample paths of the stochastic process can be depicted on the two-dimensional graph. After wandering around a bit this path terminates back at (0,0):

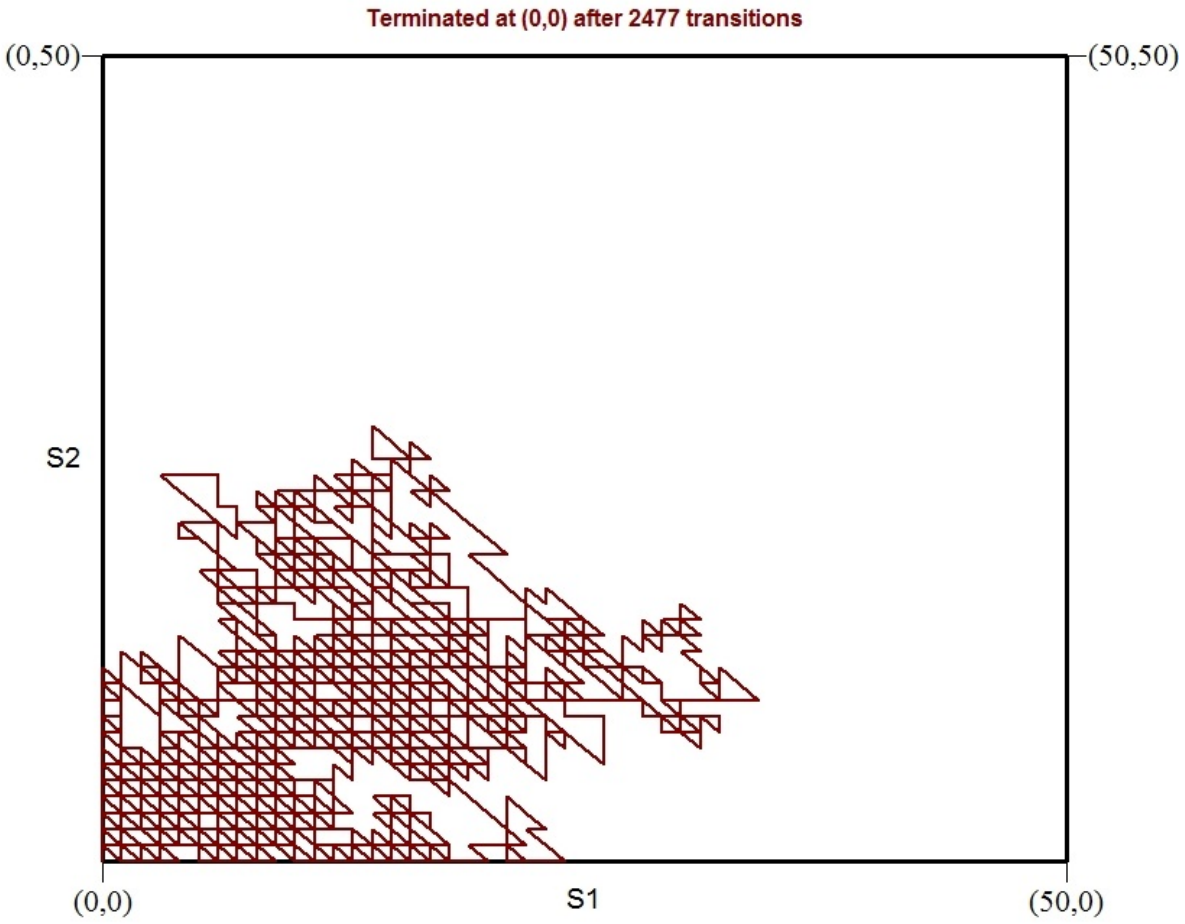

Figure 3: MC Sample Path Terminating at (0, 0)

This path takes a long time but finally terminates at (50, 3):

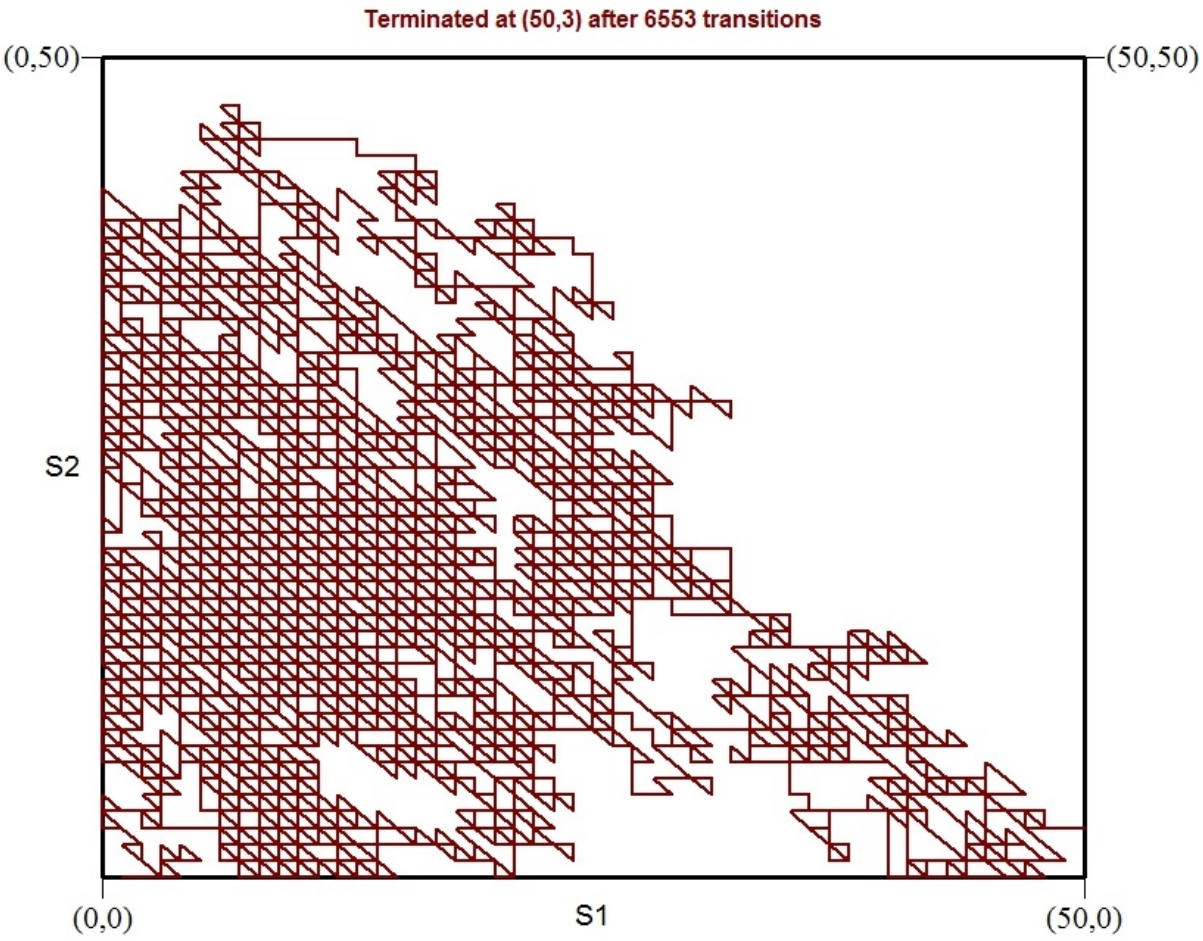

Figure 4: MC Sample Path Terminating at (50, 3)

Simple Monte Carlo simulation of this process yields an unbiased estimator of the rare event probability. But MC is inefficient. Its variance can be made arbitrarily small by making the number of runs sufficiently large. But attaining an acceptably small variance is often not feasible in practice. With MC the number of runs required for an

acceptably small variance can be prohibitively large.

2.3 Multiple Designs

Consider a "design" as a system, of virtually any kind, whose behavior, or performance, can be simulated by computer models. A common problem is to compare several systems which differ in some pertinent design characteristics or parameters, in order to select the best such system, according to criteria defining the "best". Simulation of these systems may be the only feasible approach to choosing the best. Common terminology refers to these systems as "designs". They differ in some design characteristics important for system performance.

If there is no (or very little) randomness in the system a single simulation run can suffice to analyze system performance. In general, however, some non-negligible randomness is inherent in the systems. It must be modeled by appropriate probability distributions. Several, generally many, runs of the simulation are then required to generate output adequate for statistical analysis.

The task is to design the simulation to enhance the probability of making the best choice. That is known, formally, as the Probability of Correct Selection, or PCS. The task is to maximize the PCS.

Assuming the simulation is well designed and its randomness adequately modeled, the main tool for maximizing PCS is the number of runs, or replications, of the simulation. The more runs, the higher the PCS. The PCS could, mathematically, be driven to 1 (absolute certainty) by an infinite number of runs. In practice there is some

computational limit, which may be expressed directly in terms of the number of runs, or, more generally, in the computational time available for simulation. PCS is maximized subject to some constraint on computational effort, generally called a "computing budget". Mathematically it is a constrained optimization (maximization) problem.

The decision variable in this optimization problem is not the total number of runs. It is clear that the budget constraint should be exhausted. One should do as many runs as possible within that constraint. The decision variables are the allocations of those runs among designs. (And even, as will be seen below, within designs.) If the budget constraint permits 1000 runs, how should they be divided among, say, 4 designs?

The simple, naïve, answer is: equally. Simulate each design with 250 runs. That allocation is almost certainly wrong, in the sense that it almost certainly fails to maximize PCS. Absent extreme luck, any arbitrary allocation will so fail. The optimization problem requires explicit analysis, not luck.

2.4 PCS and the Test Model

Any simulation technique subject to a budget constraint will perform better as that constraint expands. The test model illustrates how much better, how much faster. It has 10 designs. A test can be run on 4, 6, 8, or 10 of them. (See Appendix 1 for the parameters and probabilities of these designs.) The test is to choose the best from a set of 4, 6, 8, or 10. The designs differ in terms of their parameters. The arrival rate, λ, is the same for all, set at 1. The two service rates, μ_1 and μ_2, are given differing values, such that the probabilities of the rare event differ among designs.

Design 1 has the smallest rare event probability, designs 2-10 increasingly larger probabilities. Since the test model can be solved exactly, analytically, the real probabilities are known. We thus know whether or not each simulation gets it right, i.e., estimates design 1 to have the smallest probability.

The test simulations start with a small amount of time, T (in seconds) , called the initialization. For instance, let T = 20. A simulation is done, consisting of as many runs as possible within those 20 seconds. Based on that simulation, at that point, the best is chosen. Since we know the correct answer we can grade the choice: right or wrong. The time, T, is then increased by a fixed amount, the time increment. The simulation is continued, by the amount of the time increment. (It is not a new simulation, starting over from 0, but a continuation of the existing simulation.) At the new T (initial T + time increment) the best of the estimated rare event probabilities is again selected, that selection again scored as correct or incorrect. T is then increased by another time increment, the simulation continued to the new T, the estimated smallest probability, at that new T, again scored as correct or incorrect. This process is continued through a fixed number of time increments. The initialization plus all the time increments consumes the total computing budget.

The program that implements this test model counts “time” only as the time consumed by the actual simulation runs. That makes it more comparable to practical applications, in which simulation time typically dwarfs computational overhead.

We thus have, at each new T throughout the expanding budget constraint, a record of whether or not the simulation "got it right" at that point. We then repeat the entire simulation, from T = 0 to T = total budget, recording, at each increment, whether or not the estimated smallest probability at that point is the correct smallest probability. Each repetition of the entire simulation is called an Experiment. After many experiments we have, at each time increment (including the initialization) an estimated PCS: the number of correct selections at that point as a percentage of the number of experiments. This estimated PCS is plotted, on the vertical axis, against the time points (initialization and increments), on the horizontal axis. The resulting plot is a curve displaying an estimated PCS over an increasing computing budget constraint. It should, in principle, increase monotonically: you can, in theory, always do better with more budget. In practice it is but an estimate, subject to randomness, so it need not always increase at each increment. But the estimate, over many experiments, should show an overall steady increase, with but minor jaggedness.

One facet of these tests requires clarification. I impose this rule: if, at any increment, including initialization, the estimated probability of any design is 0, the algorithm is judged to have made an <u>incorrect</u> selection at that point. This rule will be necessary for the optimizing algorithms, which will require strictly positive estimated probabilities. Though not necessary for non-optimizing algorithms, such as simple MC, I impose the rule on them in order to make valid comparisons with optimizing algorithms. Consequently, plots of MC generally show it with a 0% PCS over the first few time

increments – a result of the fact that, at those early points in the simulation, one or more of the designs had recorded no hits, and thus returned an estimated probability of 0.

The following plot illustrates this test. It shows the estimated PCS for simple MC, over 6 designs. The initialization period is 10 sec, then 238 increments of 5 sec., for a total computing budget of 1200 sec. This is repeated 100 times (100 experiments), requiring 33 hours of actual computing time.

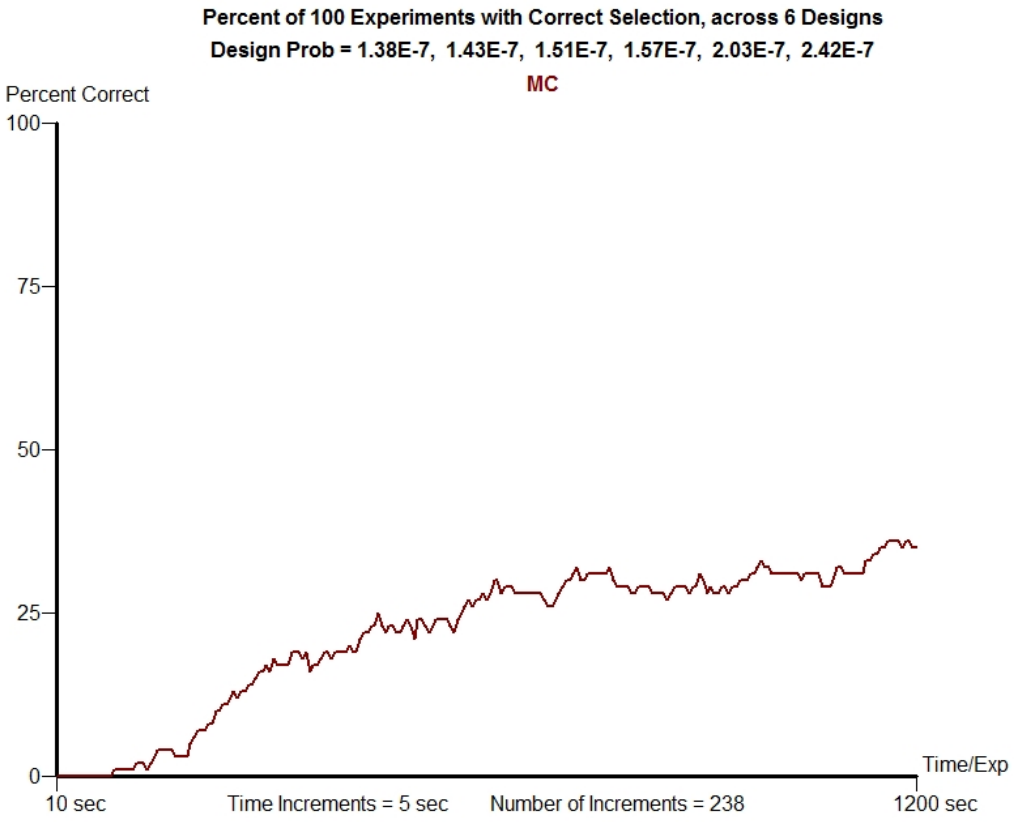

Figure 5. MC on the Test Model: Real Time Simulation

These types of plots will be most useful for comparing competing algorithms. The estimated PCS for two algorithms plotted on the same chart illustrate which technique is better, for that particular test. Estimated PCS for better techniques should increase more quickly than for poorer techniques. Though such tests prove nothing, a putative superiority of one technique over another should be re-examined if it clearly fails this comparison test.

2.5 Pseudo Simulation

As noted, the test model can be solved, analytically, for the rare event probabilities. It can also be solved, analytically, for the average number of transitions in a sample path (run). With an estimate, based on numerous experiments on the computer on which these tests were performed, of the average time-per-transition, it is possible to construct a good estimate of the average time-per-run in each of the ten designs.[6] The accuracy of these estimates is quite good, based on a large sample of runs, but also somewhat irrelevant. Any plausible estimates would do just as well, since they are employed only for comparative purposes, for which the only requirement is to use the same estimates across all comparisons. Alternatively, they could be thought of as the results produced by a computer whose speed produces exactly the estimated time-per-transition.

With these data in hand it is possible to conduct tests relatively quickly. Consider the MC plot above (Figure 1). It required 33 hours of actual computing time to perform

[6] Average time-per-run estimates can also be constructed for splitting levels, which are explained in chapter 4.

only 100 experiments. An alternative, statistically equivalent method, which I call "pseudo simulation", can be carried out in seconds. To understand pseudo simulation, consider the following method for estimating Monte Carlo's PCS:

A simple MC run produces a "hit" (the rare event occurs) or a "miss" (the run returns to the origin before hitting the rare event). The probability of a hit is the same on each run. The distribution of the hits is binomial:

$$Hits \sim \text{Binomial}\ (p, N)$$

where p = probability of a hit, N = number of runs.

With a given budget constraint, T, and a given average time-per-run, the average number of runs is easily calculated: $N = T$ / (average time-per-run), rounded. An entire simulation can thus be mimicked by drawing from the binomial distribution with parameters p and N. That is Pseudo Simulation. The real time test (Figure 4) required 33 hours for 100 experiments. Under pseudo simulation 10,000 experiments can be carried out in but a few seconds.

Pseudo simulation of more complicated techniques is not as quick as MC pseudo simulation, because they contain computational overhead absent in MC. Moreover, in more complicated techniques the probability distributions from which drawings are made are not known distributions, such as the binomial. They must be empirically constructed, from the probabilities that can be calculated for the absorbing Markov chain. This is a laborious task, but, once the distributions are constructed and coded into the program running the test model, sampling from them is quick and straightforward.

Pseudo simulation, even on the most complex algorithms, dramatically reduces the real computer time required to make comparisons on the test model. I greedily exploit it. All tests in this research (except Figure 4) employ pseudo simulation.

2.6 MC on the Test Model

I illustrate pseudo simulation with simple MC, i.e., equal allocation of computing time across designs. Consider a large computational budget: a 1 hour initiation, 22 half-hour increments, a total of 12 hours per experiment. Simulating across 6 designs, with equal allocation (2 hours per design.), over 10,000 experiments, yields:

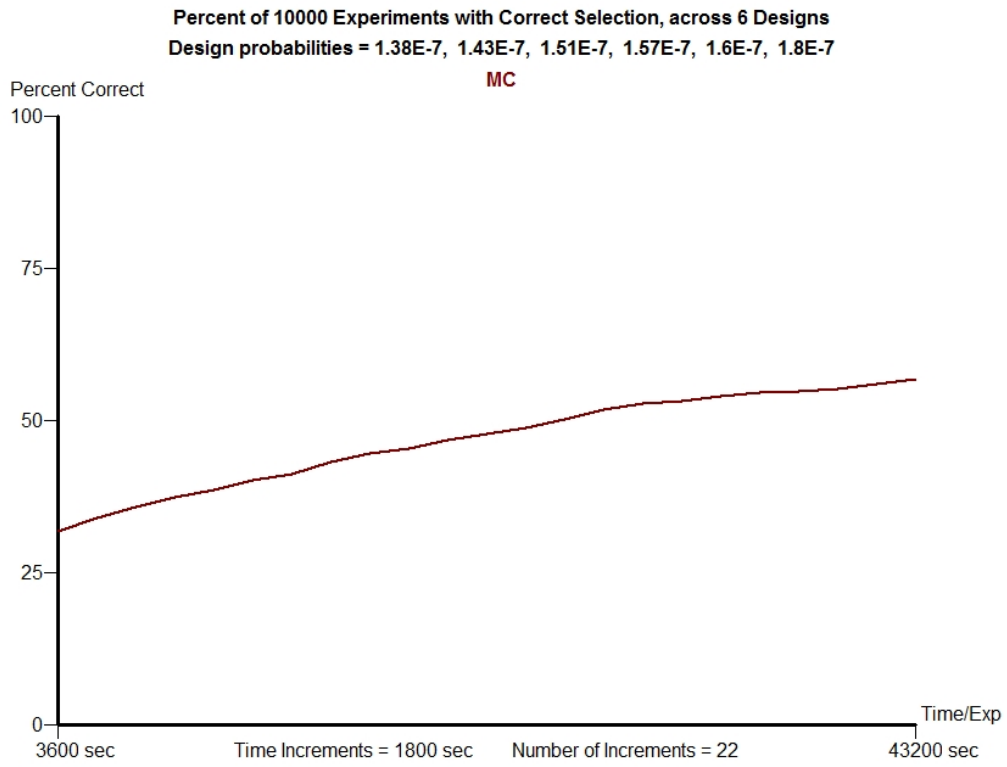

Figure 6: Simple MC, Large Budget

At the initial budget of one hour the PCS was about 32% . It made the correct selection only about one-third of the time. After 12 hours it was only 57%.

This budget constraint (12 hours total) is massively larger than the constraint against which I test other simulation techniques, namely, a 10 sec. initiation, with 238 time increments of 5 sec. each, for a total budget of 20 minutes. Simulating MC for this budget, over 6 designs, with 2000 experiments, yields:

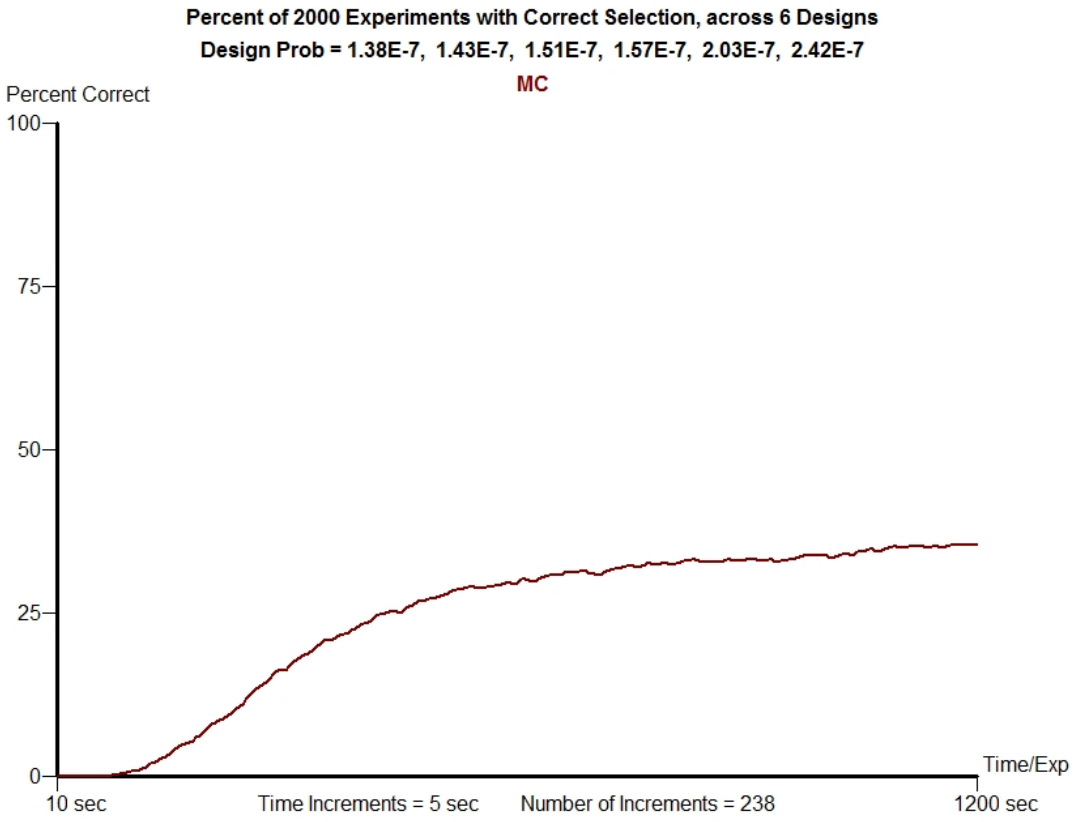

Figure 7: Simple MC, Small Budget

This is the same test as the real time test in Figure 4, which had 100 experiments. This test utilizes pseudo simulation, permitting 2000 experiments to be conducted very quickly. The results are essentially the same, allowing for the greater effect of randomness in 100 experiments, compared to 2000. MC does not reach even a 1% PCS until about 120 sec. (To understand why it starts at 0, recall the rule, explained above, that the algorithm is deemed to have made an incorrect selection if, at any point, the estimated probability for any design at that point is 0.) At termination it has attained only 35% PCS. Randomly choosing a design would generate about 17% (1 in 6). Testing MC across 8 or 10 designs would produce even worse results, probabilistically, for the same budget constraint.

This chapter has introduced the Probability of Correct Selection, the test model, and simple Monte Carlo simulation of rare event. It has demonstrated that MC is not, in general, up to the task of estimating rare event probabilities

What next? A review of techniques designed to maximize PCS across multiple designs, but not specialized for rare events. A review of techniques specialized for rare events, but not designed to maximize PCS across multiple designs. Then, finally, the purpose of this dissertation: the development of efficient techniques to maximize PCS across multiple designs.

3. OCBA[7]

Constants

D = number of designs

T = computing budget (in time)

p_i = probability of the rare event in design i (the design probability)

b_i = average time per run in design i

Random Variables

$\hat{p}_i$ = estimator of p_i (the design estimator)

Functions

$\sigma_i^2(N_i)$ = variance of $\hat{p}_i$

Decision Variables

N_i = number of simulation runs in design i

How should computational effort, or time, be allocated among designs? The Optimal Computing Budget Allocation, or OCBA, developed in Chen & Lee (2011), answers this question. OCBA allocates the computing budget to maximize (approximately) the Probability of Correct Selection (PCS). See Appendix 2 for a review of relevant literature on OCBA.

OCBA is based, intuitively, on a simple insight. The probability of selecting the best design would be enhanced if the simulation concentrated on the most competitive designs (the best and its close competitors). If some designs are far from the best, some

[7] A short summary of OCBA literature is provided in Appendix 2.

close to the best, it is best to spend more computational effort on the best and those close to the best, less on those far from the best. Relatively poor estimates for uncompetitive designs will generally do no harm, since the simulation would have to be very wrong about an uncompetitive design to make it appear to be the best. If the probabilities of designs 1 and 3 are 0.1 and 0.9, respectively, it will hardly matter if the estimate of design 3 is off by a half. An estimated 0.45 for design 3 will, with very high probability, still be larger than the estimate for design 1. Design 3 is not competitive. Suppose the probability of design 2 is 0.46. It is very competitive. Estimates of designs 1 and 2 must be very accurate to preclude incorrect selection. Clearly, almost all the computing effort must be devoted to designs 1 and 2. Design 3 cannot be totally neglected because, at the outset, the algorithm doesn't know that it is uncompetitive. But that fact can emerge, probabilistically, as the algorithm evolves. At the outset all designs deserve equal attention. But the algorithm should, as it evolves, detect the probable uncompetitiveness of design 3, and begin to shift computing effort to designs 1 and 2.

How, exactly, should it do that? Without a theory to guide implementation there is only judgment or heuristics. OCBA provides the theory on which practical algorithms for implementing this intuition can be built.

OCBA thus addresses the very problem posed in this dissertation: How to maximize the PCS when selecting the best among a set of designs. Why the need for a new approach? Because OCBA is not efficient when applied to rare events.

3.1 OCBA Theory

The theory behind OCBA is quite general, applicable to many kinds of designs and different criteria for the "best". This overview focuses on its application to the selection of the design with the smallest probability of some rare event.

OCBA theory answers the question: If we knew the real values of all the relevant parameters what would be the optimum allocation of runs (or time) to designs? If we could solve the optimization problem prior to any simulation we could make the optimum allocation at the outset and not have to adjust anything as the algorithm evolves. But we don't know the real values of the relevant parameters, so cannot solve the optimization problem. We are condemned to a strategy of gradual adjustment of allocations as the algorithm proceeds. The theory of the optimal allocation is based on the real values of the parameters. The implementation must employ estimates of the parameters as proxies for the real things.

OCBA theory assumes that, within each design, simple Monte Carlo is used to estimate the rare event probability. The outcome of each run in the simulation of design *i* is a Bernoulli random variable, with probability p_i of a hit on the rare event, with expected value p_i and variance $p_i(1-p_i)$. If the number of runs is N_i the number of hits is binomial:

$$Hits \sim Binomial(p_i, N_i)$$
$$E[Hits] = N_i p_i \quad Var[Hits] = N_i p_i (1-p_i)$$

The design estimator (the estimator of the rare event probability) is the number of hits divided by the number of runs. It is a scaled binomial (a binomial multiplied by a constant):

$$\hat{p}_i = \frac{Hits}{N_i} \qquad E\left[\hat{p}_i\right] = p_i \qquad Var\left[\hat{p}_i\right] = \frac{p_i\left(1-p_i\right)}{N_i}$$

The distribution of $\hat{p}_i$ is assumed to be approximately normal:

$$\hat{p}_i \sim Normal\left[p_i, \sigma_i^2\right] \qquad \sigma_i^2 = \frac{p_i\left(1-p_i\right)}{N_i}$$

This assumption follows from the normal approximation of the binomial. The number of hits is binomial and approximately normal (for large N_i), so the design estimator, a scaled binomial, is also approximately normal, with mean and variance adjusted accordingly.

At this point it should noted that the derivation of OCBA in Chen & Lee (2011) proceeds somewhat differently from the approach I present here. Their derivation is based on the variance of a single run in design *i*, which is $p_i\left(1-p_i\right)$, whereas this derivation is based on the variance of the design estimator $\hat{p}_i$, which is $\frac{p_i\left(1-p_i\right)}{N_i}$. The end results are equivalent, though expressed in slightly different notation. I adopt this alternative framework for OCBA to be consistent with the derivation of my new technique, OCBA+OSTRE, in chapter 7.

The best design is designed design 1. This entails no loss of generality. Some design is the best, and "1" is simply the label given to that design. Maximizing the PCS, subject to the budget constraint, over *D* designs, is expressed as:

$$\underset{N_i}{Max}\, P\left[\hat{p}_1 < \hat{p}_2, ..., \hat{p}_1 < \hat{p}_D \mid p_1 < p_2, ..., p_1 < p_D\right] \quad \text{s/t} \quad \sum_{i=1}^{D} b_i N_i, \quad N_i \geq 1$$

The event to be maximized is the intersection of *D*-1 mutually dependent events , $\bigcap_{i=2}^{D}\left(\hat{p}_1 < \hat{p}_i\right)$. Stating this probability as a closed-form function that can be maximized by standard optimization methods is too difficult. An approximation must be adopted.

3.2 Bonferroni Approximation

Bonferroni inequalities can be stated in various forms. The one useful in this context (see Chen & Lee, 2011, p. 37) is developed as follows. Let A_i, $i = 2,...,D$, be events on a probability space with measure P. Then:

$$P\left[\bigcap_{i=2}^{D} A_i\right] \geq 1 - \sum_{i=2}^{D}\left(1 - P\left[A_i\right]\right).$$

Let A_i be the event $\left(\hat{p}_1 < \hat{p}_i\right)$. Then:

$$PCS = P\left[\bigcap_{i=2}^{D}\left(\hat{p}_1 < \hat{p}_i\right)\right] \geq 1 - \sum_{i=2}^{D}\left(1 - P\left[\left(\hat{p}_1 < \hat{p}_i\right)\right]\right) = 1 - \sum_{i=2}^{D} P\left[\left(\hat{p}_1 > \hat{p}_i\right)\right]$$

The Bonferroni inequality provides a lower bound to the PCS:

$$\text{PCS} \geq 1 - \sum_{i=2}^{D} P\left[\left(\hat{p}_1 > \hat{p}_i\right)\right].$$

Note that this is a lower bound on a probability. It guarantees no particular outcome for any single simulation of the stochastic process. The optimization problem is re-stated as:

$$\underset{N_i}{Max} \sum_{i=2}^{D} \left(1 - P\left[\left(\hat{p}_1 > \hat{p}_i\right)\right]\right) \quad \text{s/t} \quad \sum_{i=1}^{D} b_i N_i = T, \quad N_i \geq 1$$

It is convenient to state the problem as an equivalent minimization:

$$\underset{N_i}{Min} \sum_{i=2}^{D} P\left[\hat{p}_1 > \hat{p}_i\right] \quad \text{s/t} \quad \sum_{i=1}^{D} b_i N_i = T$$

$$\underset{N_i}{Min} \sum_{i=2}^{D} P\left[\hat{p}_1 - \hat{p}_i > 0\right] \quad \text{s/t} \quad \sum_{i=1}^{D} b_i N_i = T$$

The requirement that the $N_i \geq 1$ can be ignored. Chen & Lee (2011) show that the N_i that solve the optimization problem must be non-negative. In fact, they must be greater than 0, else the variance of $\hat{p}_i$ would be infinite.

With the $\hat{p}_i$ (including $i = 1$) assumed normal, and σ_i^2 (including $i = 1$) their variances, the probabilities $\hat{p}_1 - \hat{p}_i$ are distributed as

$$\hat{p}_1 - \hat{p}_i \sim Normal\left[p_1 - p_i, \sigma_1^2 + \sigma_1^2\right].$$

Then: $$P\left[\hat{p}_1 - \hat{p}_i > 0\right] = P\left[\frac{\hat{p}_1 - \hat{p}_i - (p_1 - p_i)}{\sqrt{\sigma_1^2 + \sigma_i^2}} > \frac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}}\right] = P\left[Z > \phi_i\right]$$

where Z = standard normal random variable, and the functions ϕ_i are defined as:

$$\phi_i \equiv \frac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}}$$

The minimization problem can now be expressed as:

$$\underset{N_i}{Min} \sum_{i=2}^{D} P\left[Z > \phi_i \right] \quad \text{s/t} \quad \sum_{i=1}^{D} b_i N_i = T$$

or:

$$\underset{N_i}{Min} \sum_{i=2}^{D} \int_{\phi_i}^{\infty} \exp\left(-\frac{x^2}{2} \right) dx \quad \text{s/t} \quad \sum_{i=1}^{D} b_i N_i = T \ .$$

The Lagrangean is:

$$L = \sum_{i=2}^{D} \int_{\phi_i}^{\infty} \exp\left(-\frac{x^2}{2} \right) dx + \lambda \left(\sum_{i=1}^{D} b_i N_i - T \right)$$

The Optimality Conditions:

$$\frac{\partial L}{\partial N_1} = -\sum_{i=2}^{D} e^{-\frac{\phi_i^2}{2}} \frac{\partial \phi_i}{\partial N_1} + \lambda b_1 = 0$$

$$\frac{\partial L}{\partial N_i} = -e^{-\frac{\phi_i^2}{2}} \frac{\partial \phi_i}{\partial N_i} + \lambda b_i = 0 \quad i = 2,...,D$$

The optimality equations reduce to:

$$\frac{1}{b_1} \sum_{i=2}^{D} e^{-\frac{\phi_i^2}{2}} \frac{\partial \phi_i}{\partial N_1} = \frac{1}{b_2} e^{-\frac{\phi_2^2}{2}} \frac{\partial \phi_2}{\partial N_2} = ... = \frac{1}{b_D} e^{-\frac{\phi_D^2}{2}} \frac{\partial \phi_D}{\partial N_D}$$

These are the complete optimality equations, except for the budget constraint. We can ignore it because the task is to use the theory to guide allocations within the implementation of the algorithm, not to solve the optimization problem explicitly. Within that implementation the budget constraint is automatically satisfied. The algorithm stops when the time is exhausted. An explicit requirement that the N_i be integers is unnecessary. In the implementation they are necessarily integers. Treating them in theory as real numbers does no harm. The ultimate rounding is negligible, given that the numbers will be quite large. In practice the algorithm cannot be expected to satisfy the optimality equations exactly. In that context rounding error is trivial. These considerations, concerning the budget constraint, positivity, and integers, will apply equally to all optimization problems in this research.

Expressed in full, with all the partial derivatives expanded, the optimality equations can be rather unwieldy. In practice a simplified, asymptotic version is typically employed. (See Chen & Lee, 2011, pp. 46, 57.) But note that it is expressed in terms of somewhat different variables and notation from those I employ, as explained above.) These permit the optimal N_i^* to be found analytically, given a budget constraint. This simplified version is the one employed in my tests of OCBA.

This derivation of OCBA lays the groundwork for the new methods developed in this dissertation. They will be derived in similar fashion, but will differ considerably in detail and complexity.

3.3 OCBA Implementation

Implementation proceeds by substituting estimated parameters (such as $\hat{p}_i$) for the real parameters (such as p_i) of the theory. At the outset there are no $\hat{p}_i$, so the algorithm must first be "initialized": a (modest) portion of the total computing budget is T_D devoted to a standard MC simulation of each design. During initialization no attempt is made to optimize anything. The initialization time is allocated among the designs equally.

When the required estimates are in hand, at the end of initialization, optimization commences. At that point, and at regular intervals thereafter (updates), the simulation is halted so the algorithm can decide, based on the current estimates of all required parameters, where the next, generally small, amount of computing time (or number of runs) should be allocated. The optimality equations guide this decision. At the optimum they are satisfied (the equations equate), but almost certainly not at each update. The theoretical equalities are, at each update, inequalities. The algorithm tries to allocate the next (small) amount of computational effort so as to best push them toward equality. It asks: Given the current state of these inequalities, where (to which design) should the next dose of computing time (or runs) go, in order to make them "less unequal" at the next update. At each update an allocation is made, the simulation implements that allocation, then pauses for another update. This continues until the total computing budget is exhausted.

If the allocations are well chosen the optimality terms should become less and less unequal. Over an infinite computing horizon they should converge to equality. In finite time that cannot be expected, but the updatings should move them continuously toward equality. (This does not mean that every single update will reduce the inequalities, but that the tendency should be strongly in that direction.) The actual PCS, at the end of the simulation, should be closer to the optimum than the PCS of an arbitrary allocation with no optimization. Not the optimum, but closer. This does assume, without proof, that closer is better. That is, as the inequalities are reduced, the PCS increases monotonically.

It will surely be objected, at this point, that what the algorithm is attempting to do – reduce the inequalities in the optimality relationships – is not a well-defined task. There is no one inequality, but a vector of inequalities. In fact, several vectors, depending on how the equations are arranged. What is the measure of overall inequality that the algorithm tries to reduce? Some norm of one of the vectors? It is plausible (but unproven) that the choice does not much matter. It certainly shouldn't matter asymptotically. But we simulate finitely, not asymptotically. We assume that any reasonable measure of the inequality to be reduced at each update will work well enough in practice. This assumption seems to be borne out in practice. A method for updating OCBA is discussed below, under section 3.5 on Asymptotic Approximations.

3.4 False Leads

An awkwardness bedevils algorithms that try to optimize the PCS. Each update is guided by the underlying theory (the optimality equations). To apply that theory the current estimates of all the parameters must be used in lieu of the real, unknown

parameters. But the current estimates are, almost certainly, wrong. That is not crippling, per se. They should improve, in a probabilistic sense, as the simulation progresses. But there is one kind of mistake the algorithm can make that can be more serious.

The problem is best explained by example. Let there be 3 designs, with probabilities 0.1, 0.2, 0.3. (The issue is the same for rare and non-rare events.) Suppose, at an update, the current estimates are 0.11, 0.26, 0.25. Designs 2 and 3 are "reversed", in the sense that design 3 is, incorrectly, estimated to be smaller than design 2. But that error has no impact on the way the algorithm implements the theory. Within the theory designs 2 and 3 are interchangeable. They play identical roles within the structure of the theory, and thus within the structure of the algorithm. Interchanging them has no impact on the algorithm's attempt to reduce the inequalities of the optimality relationships.

But suppose the estimates are 0.16, 0.15, 0.25, for designs 1, 2 and 3, respectively. Now design 2 is the estimated best. The theory, however, is developed on the assumption that design 1 is the best. This matters because, in the theory, the best design plays a much different role than all the other designs. The probability of the best design (designated as design 1) appears in every event in the intersection of events comprising the event whose probability is to be minimized. (See the statement of the minimization problem, page 30.) The probability of the best design, and only the probability of the best design, is a parameter in every term in the optimality equations. Its role in the structure of the theory, and thus in the structure of the algorithm, is very different from the roles of other designs. Interchanging it with any other design can have serious consequences.

The algorithm doesn't know that design 1 is the best design. (We know, for the test model, but can't tell it.) It must assume, in applying the theory, that its estimated best design is the true best design. It must assign its estimated best design to play the role of the theory's best design, designated "design 1". The implementing code must assign to the p_1 of the theory whichever estimated probability is the smallest. If $\hat{p}_2$ is the estimated smallest probability, then not only must it play the role of the theory's p_1, but $\hat{b}_2$ must play the role of the theory's b_2, and likewise for all other relevant parameters.

This adjustment, which must be implemented, if required, at every update, implies not just extra coding. It can substantially distort the algorithm's allocations. If the algorithm incorrectly estimates $\hat{p}_2$ to be the real smallest probability, the allocations based on that estimate could be seriously misdirected. This I call a False Lead.

Every algorithm trying to maximize the PCS is almost certain to generate false leads at some updates, especially the early ones, where estimated probabilities are most prone to large error. False leads mean, in effect, that the algorithm tries to optimize "the wrong thing". The resulting allocations are likely to move the optimality relationships away from, not toward, equality.

But false leads are not fatal. Any new allocation, however suboptimal, improves some estimate (in a probabilistic sense). Eventually these improvements can overcome false leads, as the algorithm evolves. Significant progress toward equality of the optimality equations will be slowed by false leads, but the algorithm has the potential to

recover from them and (re)commence the allocation of computing time in a manner that decreases the inequalities of the optimality relationships.

False leads are most likely, and least damaging, when they occur between a few highly competitive designs. If designs 1 and 2 are very competitive, the allocations to them will be very similar, even if design 2 is, at some update, falsely estimated to be the best. False leads are most damaging when they falsely designate a rather uncompetitive design as the best. But such false leads will not be common, and can be more easily reversed than false leads among competitive designs.

In general false leads are an inevitable part of an optimizing algorithm, but not necessarily serious enough to undercut their effectiveness, as illustrated in the tests performed in this research. Chen et al. (2003) present empirical evidence that OCBA, in practical implementation, can actually outperform a theoretically optimal allocation of runs based on known parameters. That is, the theoretically optimal allocations are derived from known parameters, and so are exact, whereas OCBA allocations depend on evolving estimates of those parameters. OCBA is, then, vulnerable to false leads, but is nonetheless sufficiently robust, in practice, to overcome them and, in some tests, equal or outperform the theoretical optimum.

Non-optimizing algorithms also generate false leads, in the sense that, at any point in the evolution of the algorithm, its current estimate of the best design can be wrong. The difference between non-optimizing and optimizing algorithms is that a false lead in a non-optimizing algorithm has no bearing on the allocation of computing time. The algorithm simply continues with whatever arbitrary allocation is built into it. False leads

do not, in any sense, undercut the effectiveness of the continued simulation. For optimizing algorithms they certainly can, by generating allocations based on optimizing "the wrong thing". This suggests that, in practice, optimizing algorithms may perform more poorly than non-optimizing ones in the very early stages of the simulation. If they start out with false leads, it may take a bit of time to overcome them. But the power of optimizing algorithms lies in their capacity to so overcome, relatively quickly, and their ability to minimize false leads thereafter.

3.5 Asymptotic Approximations

Chen & Lee (2011) recommend a simplification of the complete optimality equations. By considering the asymptotic behavior of the simulation, as the computing budget becomes infinitely large, they reduce them to expressions from which the (approximately) optimal N_i can, at each update, be easily calculated. With that calculation in hand, how should the next allocation be made? A simple rule is to devote the next amount of computing time to the design whose actual N_i is, proportionally, the farthest short of its optimum. Another would be to divide the effort among all the N_i short of their optima in proportion to the degree by which they fall short. Either method can be effective. (Chen & Lee, 2011, recommend proportional allocations among all the N_i.) Over a large budget there should be no discernible difference between them. Test simulations of OCBA, presented below, are based on the first method: allocate all effort, at each update, to the design "most starving", the one whose N_i proportionally lags the

farthest behind its optimum. The allocation at each update is small, compared to the total number of runs. In this research the new allocation at each update is 100 runs.

3.6 OCBA and Rare Events

OCBA does not perform well on rare events. It allocates computing effort among designs. Within designs it uses MC to estimate the design probability. But MC is inefficient at estimating rare event probabilities. The significant improvement OCBA can bring to the simulation of multiple designs when the probabilities are not rare largely disappears when they are rare. It is overwhelmed by the curse of Monte Carlo.

I compare OCBA to simple MC (equal allocation) with a computing budget comprised of a 10 sec. initialization and 238 increments of 5 sec. each, producing a total budget of 20 minutes. The simulation is across 10 designs, with 2000 experiments.

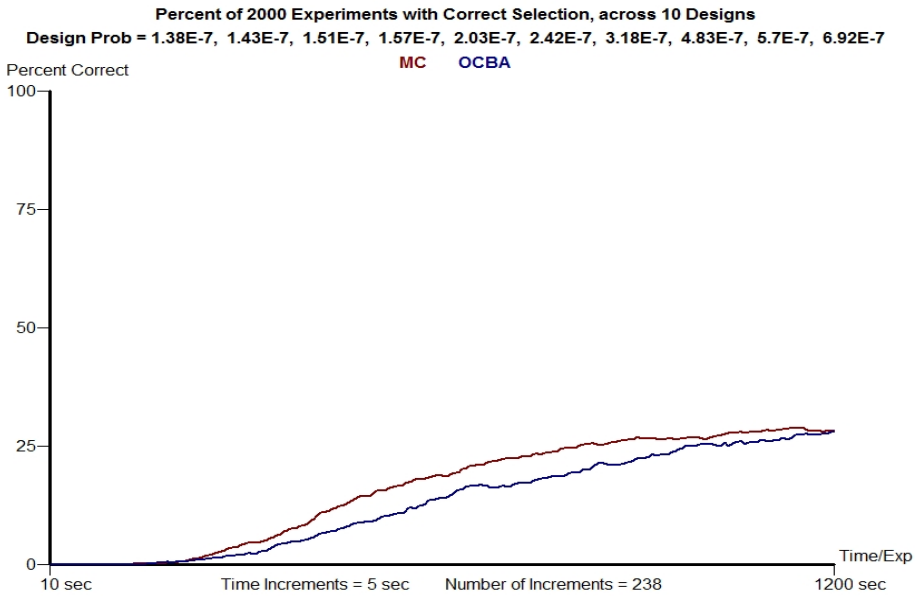

Figure 8: OCBA vs MC, 20 Minute Budget

In this test OCBA is worse than simple MC, until the very end of the budget. That is due to the relatively small budget. It is not due, in these tests, to OCBA's extra computational overhead, as the test program counts only simulation time, not computation time. OCBA almost certainly suffers, at the outset, from false leads, and there is not enough time for OCBA's updating to overcome them and move its allocations significantly toward the optimum. Can OCBA recover, given more time? Here is a test over 2 hours, with initialization and time increments at 30 sec:

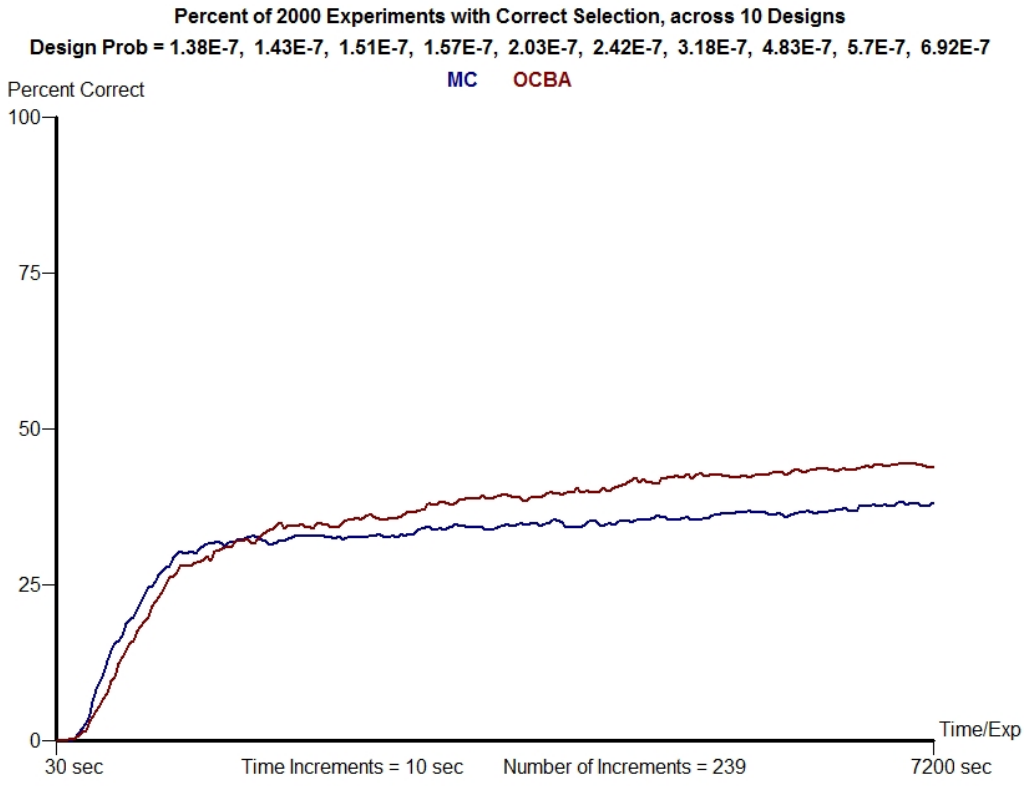

Figure 9. OCBA vs MC, 2 Hour Budget

The algorithm is sufficiently robust to recover from false leads, given sufficient time. The advantage OCBA should enjoy, and does enjoy after recovering from false leads, lies in the fact that, unlike MC, it does not "waste" much time on uncompetitive designs. It can spend more time on competitive designs. This test was run over 10 designs to illustrate that advantage. The last few of these designs are rather uncompetitive. Compare how MC and OCBA allocate time, and runs, on this test:

Table 1: MC vs OCBA Allocations

	MC			OCBA	
Design	Time(sec)	Runs	Probability	Time(sec)	Runs
1	720	161,119,225	1.38E-7	1815	406,173,533
2	720	159,933,056	1.43E-7	1679	372,940,869
3	720	159,635,527	1.51E-7	1413	313,242,343
4	720	159,039,276	1.57E-7	1221	269,772,202
5	720	154,632,176	2.03E-7	421	90,330,113
6	720	151,698,086	2.42E-7	246	51,889,232
7	720	147,558,817	3.18E-7	148	30,268,558
8	720	140,612,381	4.83E-7	95	18,527,267
9	720	137,751,376	5.7E-7	84	16,089,068
10	720	134,850,883	6.92E-7	78	14,670,900

The allocation is in terms of time. The runs are determined by the amount of time allocated to a design and the average time-per-run of a design. The higher the rare event

probability, the longer the average time-per-run for a design. Thus the runs under MC decrease monotonically, by rather small amounts. The runs under OCBA decrease very sharply, after the first 3 or 4 designs. It wastes very little time on the higher designs. The probability of the uncompetitive design 10 is about 5 times greater than that of design 1, but MC allocates design 10 about 10 times more, in time and runs, than does OCBA.

Despite OCBA's much more efficient allocation of effort, it still performs poorly. After 2 hours of simulation it made the correct selection only 28% of the time. It should be emphasized that this poor performance results solely from the application of OCBA to rare events, for which it was not designed.

What if the probabilities are non-rare, but quite competitive? With highly competitive designs OCBA offers little advantage over MC. I test OCBA vs MC over 10 non-rare designs, some fairly competitive:

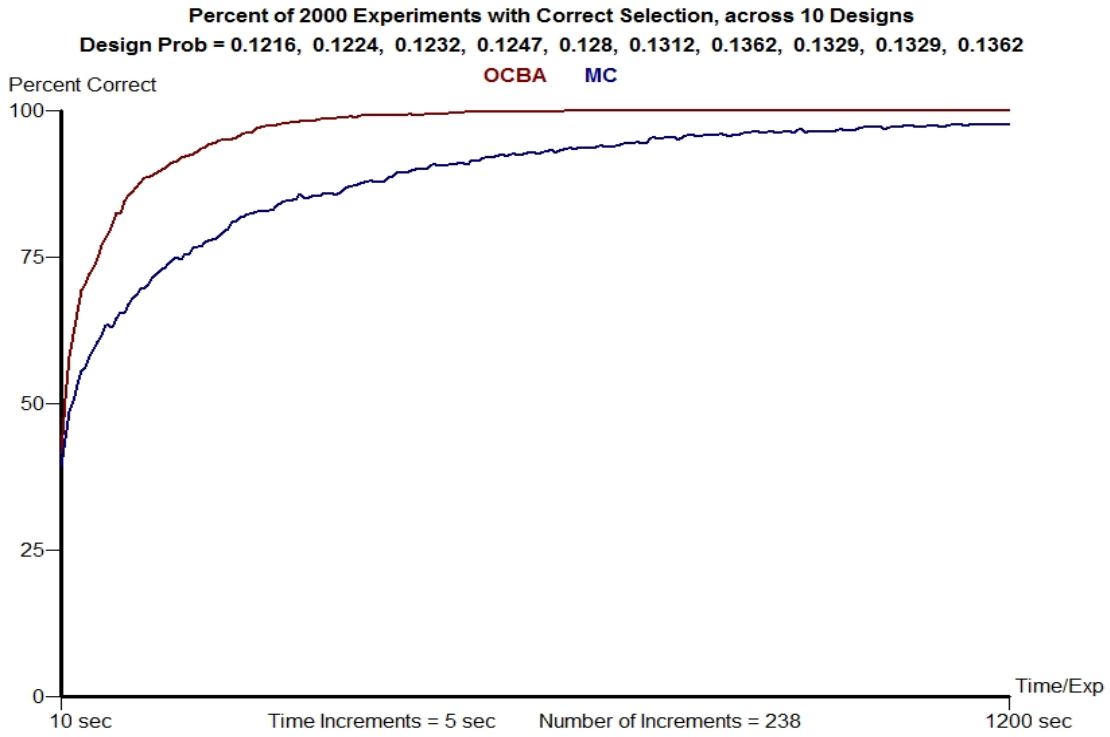

Figure 10. OCBA vs MC, Non-Rare Probabilities

OCBA quickly outperforms MC. Iit can be quite efficient when the probabilities are non-rare. It is only for rare events that other simulation techniques are needed.

4. Splitting[8]

Constants

L_j = level j, $j = 1,\ldots,\mathrm{L}$

p = probability of the rare event

p_j = level probability for level j (probability of a run in L_j hitting L_j)

q_{kj} = probability of starting a run in L_j from state k in L_{j-1}

p_{kj} = probability of a run starting in state k of L_{j-1} hitting L_j

σ_k^2 = variance of a run from state k of L_{j-1}

b_j = average time of a run in L_j

T = computing budget, in time

Random Variables

$\hat{p}$ = estimator of p

$\hat{p}_j$ = estimator of p_j

$\hat{q}_{kj}$ = estimator of q_{kj}

$\hat{p}_{kj}$ = estimator of p_{kj}

n_{kj} = number of runs starting from state k in L_{j-1}

h_{kj} = number of hits on L_j from runs starting from state k in L_{j-1}

h_j = number of hits on L_j

X_{kj} = Bernoulli random variable for the outcome of a run from state k in L_{j-1}

Functions

$f_{kj}(x)$ = probability mass function (*pmf*) for X_{kj}

$f_j(x)$ = probability mass function (*pmf*) for the outcome of a run from L_{j-1}

Decision Variables

n_j = number of runs in level j, subject to constraint that $\sum_j b_j n_j = T$

[8] A short summary of Splitting literature is provided in Appendix 3.

Splitting is the general name of an ensemble of techniques designed to efficiently estimate rare event probabilities. The state space is divided (split) into subsets that partition it. One of the subsets is the rare event. The subsets are constructed so that a sample path, to reach the rare event, must first transit the other subsets.

Splitting is easily illustrated on the test model. It divides the state space into 4 non-intersecting subsets plus their boundaries. The state space is split at the boundaries, called "levels": L0 = 0, L1 = 12, L2 = 15, L3 = 38, L4 = 50. In some contexts it may be useful to designate levels by subscripts: L_1, L_2, etc. The subsets are:

L0: the single state (0, 0).

All states for which (S1 < 12) , (S2 < 12), except (0, 0).

L1: all states for which S1 = 12 or S2 = 12.

All states for which (12 < S1< 25) and (12 < S2 < 25).

L2: all states for which S1 = 25 or S2 = 25.

All states for which (25 < S1< 38) and (25 < S2 < 38).

L3: all states for which S1 = 38 or S2 = 38.

All states for which (38 < S1 < 50) and (38 < S2 < 50).

L4: all states for which S1 = 50 or S2 = 50.

This splitting of the test model is depicted as:

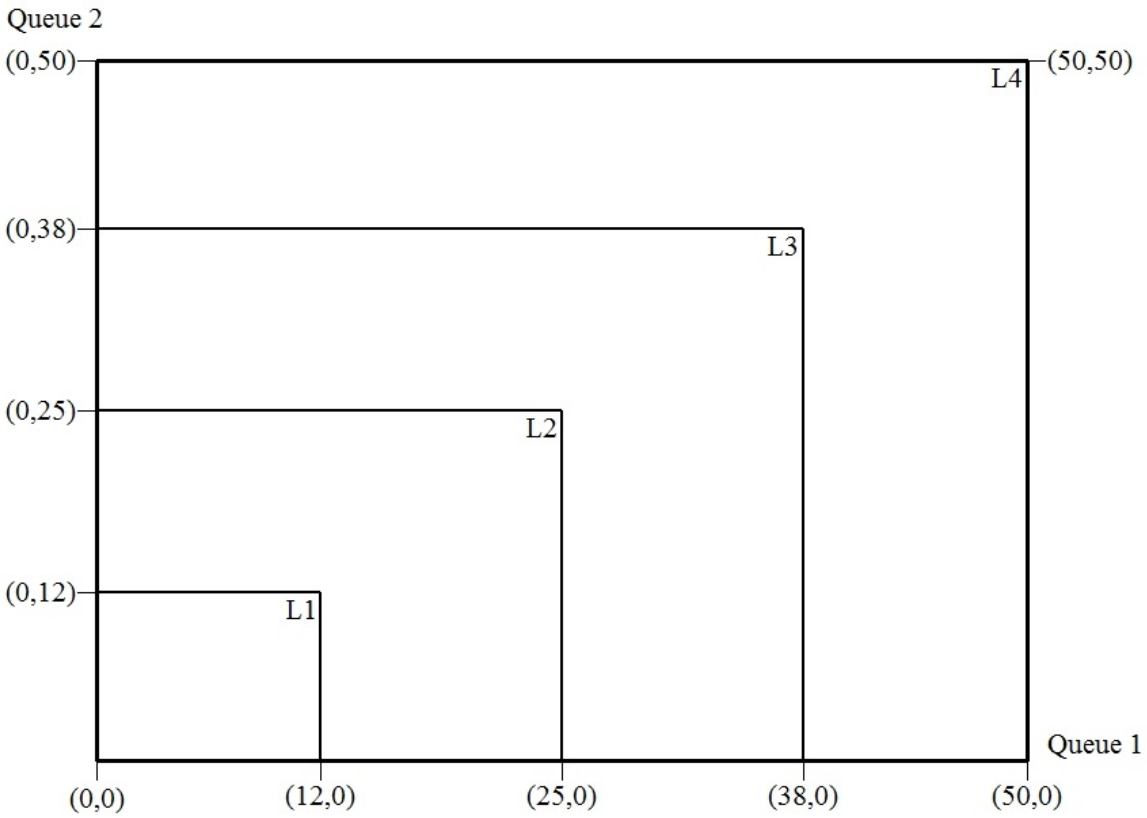

Figure 11: Splitting of Test Model into 4 Levels

Unlike Monte Carlo, the runs in a splitting simulation start in some level and always terminate in the next highest level, or in L0, the origin. Terminology relating to runs and levels needs clarification. As defined above, a level is a boundary splitting the state space into subsets which, together with the levels, partition the state space. The levels are L0, L1, L2, etc. The phrase "in a level" refers to states in that level. For example, (12, 3) is "in L1", (11, 3) is not. But it is convenient, though with some abuse of terminology, to speak of "runs in a level". This will mean runs commencing in the

next “lower” boundary. Runs “in L2”, for instance, refer to runs commencing in L1. Runs starting at L0 are runs “in L1”. Runs starting in L2 are runs “in L3”, etc. The phrase “in L3” can thus mean a state in the boundary L3, or a run starting in a state in the boundary L2. It should be clear from the context whether the reference is to a state in a boundary or a run starting in a boundary. It should be noted that runs in, say, L3 may never actually enter a state in the subset “above” L2. To be a run “in L3” it need only start in the boundary L2. From that point its first move, and all subsequent moves, could be in states “below” L2, until termination at (0, 0).

The following graphs illustrate runs in each of the levels, some terminating at the boundary, some at the origin. The probabilities generating these runs are non-rare:

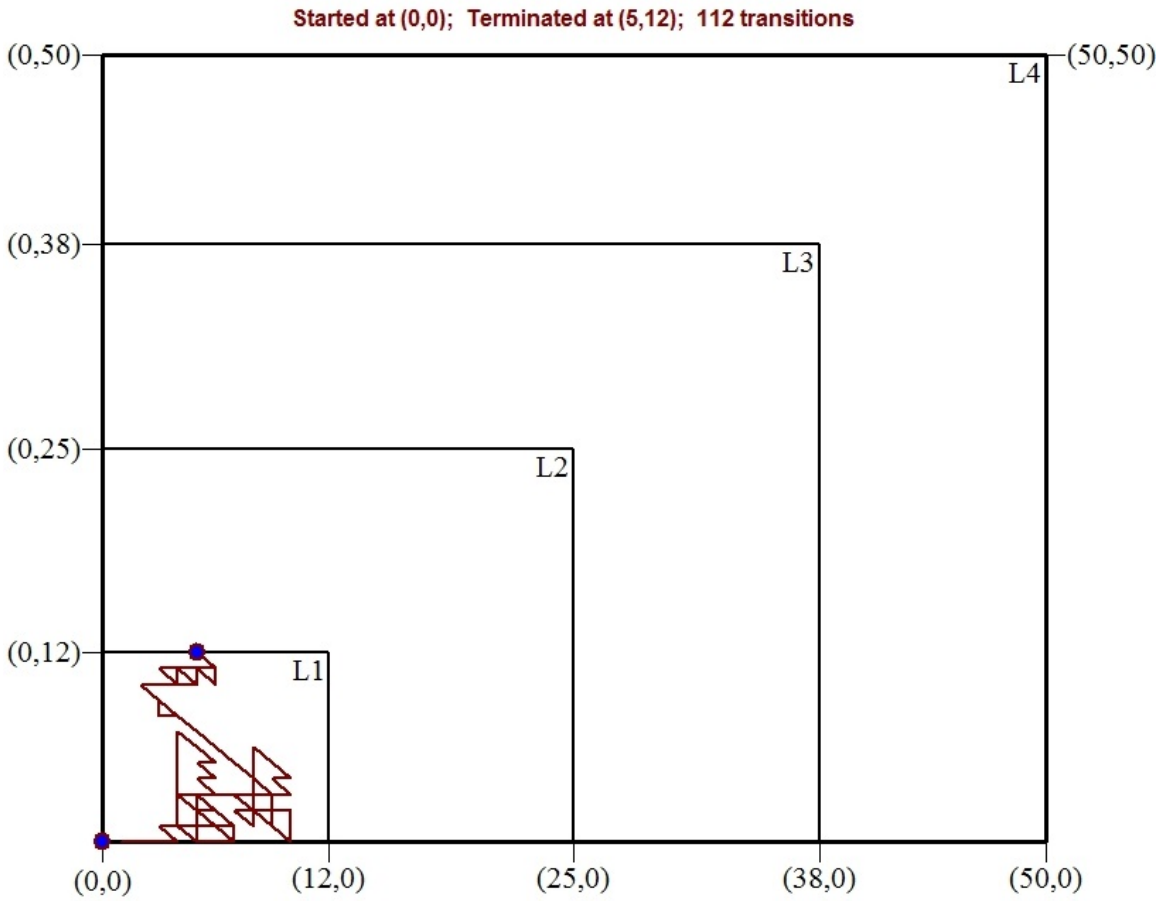

Figure 12: Sample Run in L1, Terminating in L1

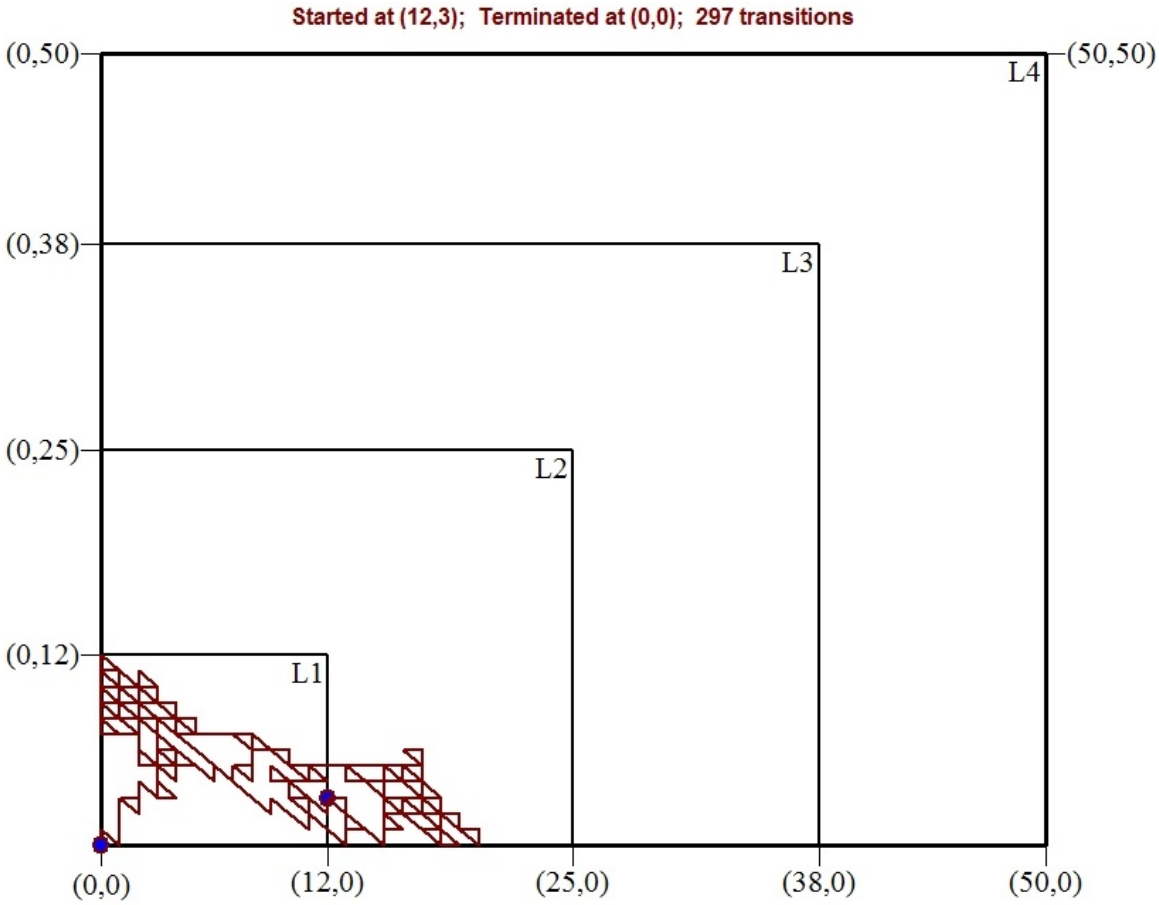

Figure 13: Sample Run in L2, Terminating at the Origin

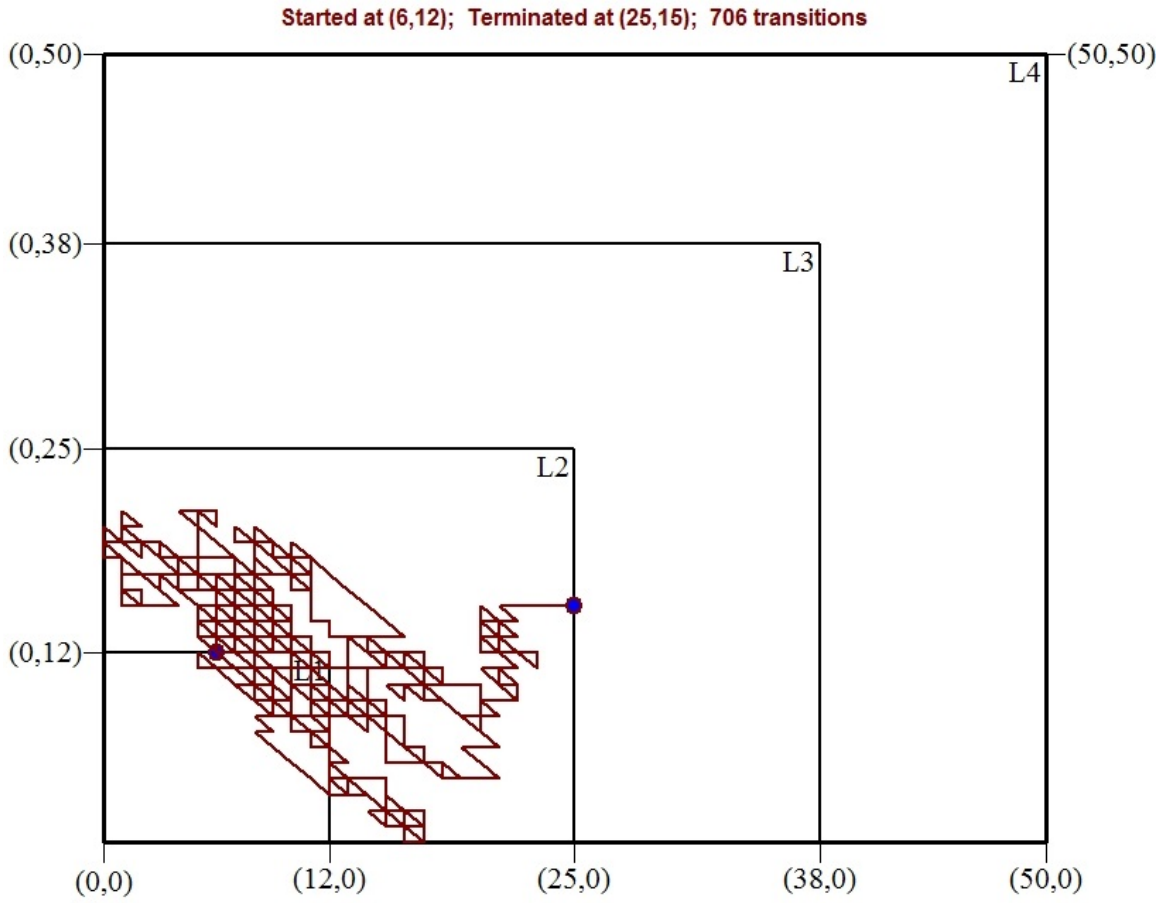

Figure 14: Sample Run in L2, Terminating in L2

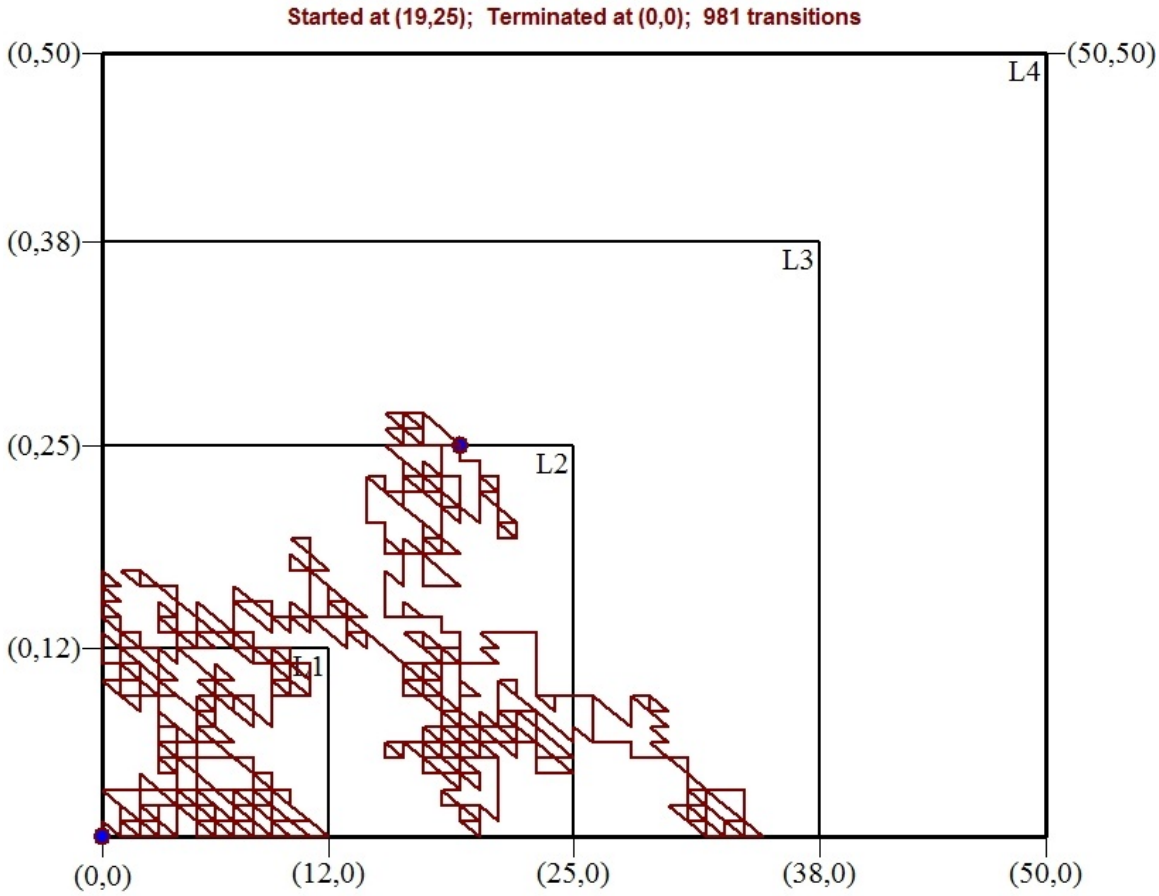

Figure 15: Sample Run in L3, Terminating at the Origin

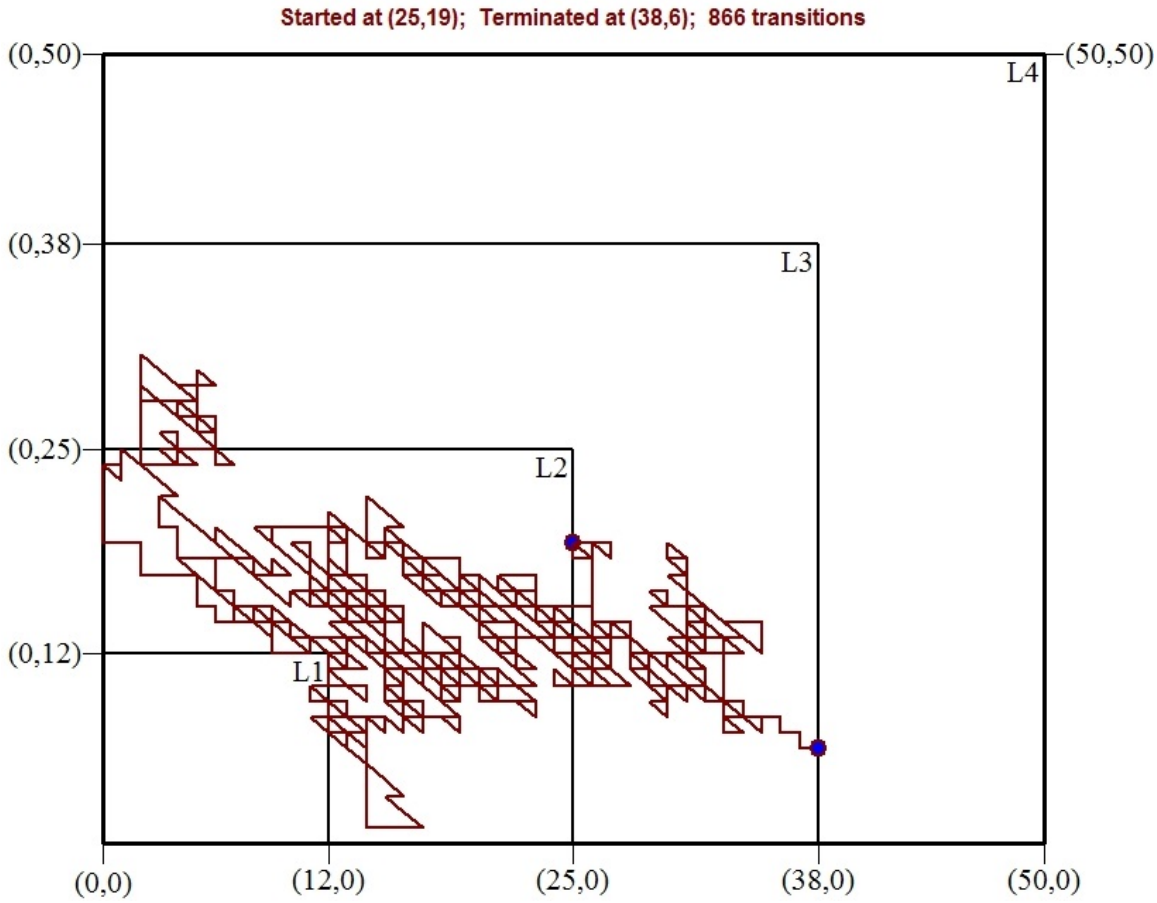

Figure 16: Sample Run in L3, Terminating in L3

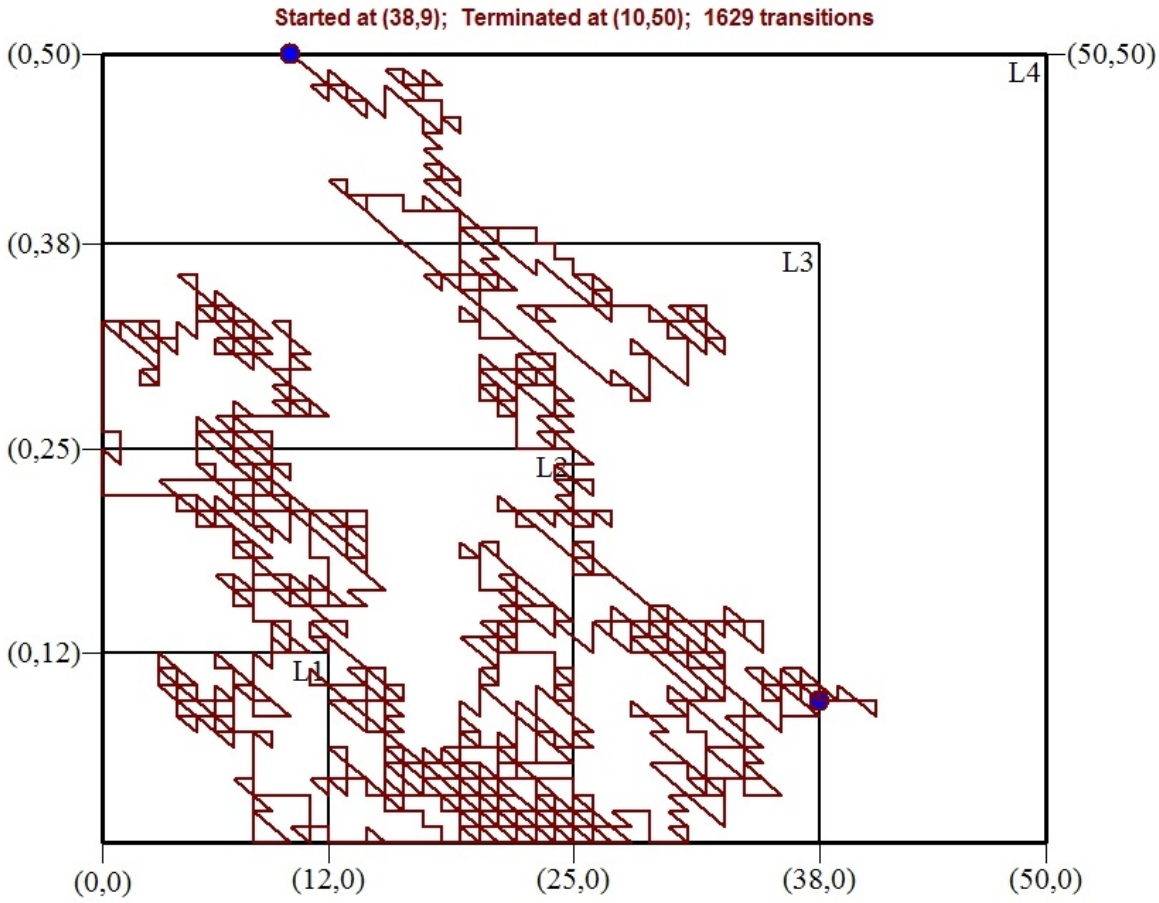

Figure 17: Sample Run in L4, Terminating in L4

This last example, depicting a sample run in level 4 terminating in the rare event, is an illustration of the fact that runs can spend considerable time "below" their starting points but nonetheless terminate in the next higher level. Generally speaking, the "higher" a run's starting point, the more time it will take, on average, to complete its path and terminate. Loosely put, it has more space to wander around in before terminating.

The rare event in the test model is L4. It is obvious that an MC run – a run commencing at the origin and terminating only in L4 or back at the origin – can only hit L4 if it first transits all the other subsets and, in particular, first hits L1, L2, and L3. Splitting constructs levels to ensure this property holds.

Any number of levels could be chosen and placed anywhere, as long as they partition the state space so that a MC run can reach the rare event can only by first transiting all the other subsets. The number and placement of the levels is the most important decision variable in the design of standard splitting. There is theoretical guidance for these choices, discussed below (see page 65).

4.1 Estimating Level Probabilities

Let the design probability be p, its estimator $\hat{p}$. (Splitting applies to a single design, so no subscript is required to designate the design.) Splitting estimates p by first estimating the probabilities that a run in a level will hit that level. Keeping in mind the distinction between a run "in a level" and a run "hitting a level" it makes perfect sense to ask, for each level $j = 1,\ldots,L$:

What is the probability that a run in L_j will hit L_j[9]? Let that be p_j. The p_j are the "level probabilities". The overall probability that an MC run (commencing at the origin and terminating only at the rare event or back at the origin) will hit the rare event can be expressed as the product of the level probabilities:

[9] It is redundant to include the phrase "before hitting L_0" since the run terminates the first time it hits L_0 or L_j.

p = (probability of a run in L1 hitting L1) x
(probability of a run in L2 hitting L2) x
(probability of a run in L3 hitting L3) x
(probability of a run in L4 hitting L4).

$$p = p_1 p_2 p_3 p_4 .$$

Equivalent constructions of p can be made for any number of levels and corresponding level probabilities. Splitting estimates p by estimating the level probabilities, then multiplying:

$$\hat{p} = \hat{p}_1 \hat{p}_2 \hat{p}_3 \hat{p}_4 .$$

In sec 4.4 this estimator is shown to be unbiased.

Level estimators are constructed by dividing, for each level, the number of hits by the number of runs. Let

n_j = number of runs in level j

h_j = number of hits on level j

Then

$$\hat{p}_j = \frac{h_j}{n_j}$$

Insight into the effectiveness of splitting can be gained by from the following thought experiment. Level estimators could also be constructed from simple MC runs. Impose levels on the state space. Do a simple MC run, starting at the origin and terminating at the rare event or back at the origin. That is, do not stop the run when it hits any of the levels, but record those levels it does hit. (Record only the first time it hits

them.) Suppose it hits L1, L2, L3, but not L4. That generates a record of 1 run in each level, and 1 hit on the first 3 levels. Do a large number of runs and construct the record of level runs and level hits. Level estimators could be constructed from this record, and a design estimator calculated as their product. That design estimator would be exactly the same as the simple MC estimator (the estimator calculated without the levels). To see this:

$$\hat{p} = \hat{p}_1 \hat{p}_2 \hat{p}_3 \hat{p}_4$$

$$\hat{p} = \frac{h_1}{n_1} \frac{h_2}{n_2} \frac{h_3}{n_3} \frac{h_4}{n_4}$$

Note that, in this MC experiment, the number of runs in any level (above level 1) is exactly the number of hits at the prior level: $n_2 = h_1,\ n_3 = h_2$, etc. Thus

$$\hat{p} = \frac{h_1}{n_1} \frac{h_2}{h_1} \frac{h_3}{h_2} \frac{h_4}{h_3} = \frac{h_4}{n_1},$$

which is precisely the MC estimator, since n_1 is the number of runs from the origin, h_4 the number of hits on the rare event.

MC suffers from the fact that it generates fewer and fewer runs at higher levels. It launches n_1 runs from the origin. The probability that any such run reaches L1 is less than 1, often much less. So n_2, the runs in L2, will be less, often much less, than n_1. The same reasoning applies to L3 and L4. The runs at each level will be less than those at the prior level, and, with rare events, much less: $n_1 >> n_2 >> n_3 >> n_4$. (The non-zero probability they could be equal is too small to take seriously.) This means that the

MC artificially constructed level estimators, at levels above 1, will become more and more unreliable, their variances larger and larger. The design estimator $\hat{p} = \hat{p}_1 \hat{p}_2 \hat{p}_3 \hat{p}_4$ will have a large variance (relative to what is desired for efficient estimation) because the variances of the estimators for $\hat{p}_2$, $\hat{p}_3$, $\hat{p}_4$ will be, increasingly, (very) large, because they will be based on smaller and smaller numbers of runs.[10]

How does "real" splitting improve upon this artificially constructed splitting of simple MC? They appear the same, when expressed as

$$\hat{p} = \frac{h_1}{n_1} \frac{h_2}{n_2} \frac{h_3}{n_3} \frac{h_4}{n_4}.$$

The genius of real splitting is that it breaks the link between the number of hits on a level and the number of runs in the next level. In MC these are the same. In splitting they are not. The number of runs in each level becomes a decision variable which can be made arbitrarily large (thereby making the variance of the level estimator arbitrarily small), independent of the number of hits on the prior level. (Or, equivalently, the time devoted to simulating level *j* is independent of the time for level *j*-1.) n_2 can be made as large as desired, independently of h_1. How is this done? That is, how is it done so as to maintain the one good thing Monte Carlo does deliver, unbiased estimators?

4.2 Fixed Splitting

[10] This does not imply that the variance of $\hat{p}$ is the product of the variances of the level estimators, only that it increases as the variance of a level estimator increases.

Splitting[11] comes in two several flavors, the primary ones called "fixed" splitting and "fixed effort" splitting. They set the number of runs in each level in different ways. A run in MC simply continues, as a single run, until it dies. Not so in fixed splitting. A run that hits L1 spawns a number of separate runs, each launched from the point in L1 hit by the initial run. Each of those separate runs then continues until it hits L2, or dies at the origin. Those that hit L2 then launch additional separate runs, each of which continues until it hits L3, or dies at the origin. This procedure continues until all the runs, through all the levels, spawned by the single run from the origin, have terminated. Then another single run from the origin is launched and the process repeated. By controlling the number of new runs launched at each level (from the runs that reach it from the prior level) the number of runs at each level can be made much larger than they would be under simple Monte Carlo. The total number of new runs in any level over the entire simulation is not, itself, a control variable, since it is the product of the number of new runs launched from each hit on that level and the number of hits. The number of hits is a random variable, so the total number of runs, over the entire simulation, is a random variable. But it – the total number of runs -- can be made a large multiple of the number of hits, whereas, in MC, it is always exactly the number of hits. The variances of the level estimators, above level 1, can thus be made much smaller than under an artificial splitting of an MC simulation.

[11] For an overview and analysis of the varieties of splitting, see L'Ecuyer et. al. in Rubino and Triffin (2009).

There are various versions of fixed splitting. A common one is RESTART (Villen-Altamirano, 1994, 2006), which contains a scheme for killing runs that have low probability of reaching the next level. Optimizations of fixed splitting have been developed by Lagnoux-Renaudie (2008). A review of splitting in general, with emphasis on fixed splitting, is found in Rubino & Triffin (2009).

This potted overview of fixed splitting elides much in theory and implementation. It skirts the key question – how to maintain unbiasedness while lowering variance -- but I explore it no further, as this research focuses exclusively on "fixed effort" splitting.

4.3 Fixed Effort Splitting

Fixed effort splitting is explained by reference to the test model. The generalization to other types of stochastic processes is straightforward.

Broadly speaking, fixed splitting and fixed effort splitting differ in the sense that "depth first" differs from "breath first". Fixed splitting is depth first. It pursues the fate of a single run until it, and all the offspring runs it spawns at higher levels, have died. Then it launches a new single run from the origin and repeats the procedure. Fixed effort splitting is breath first. It completes all the desired runs in L1 before doing any runs in L2. Then all the runs in L2 before any runs in L3. And so on.

Level estimators are calculated from the "runs in the level" and the "hits on the level". Consider the estimator for level *j*. Using the terminology adopted above, the runs in level *j* all start from some state in L_{j-1}. They score a hit if they hit some state in L_{j}. Otherwise they die at the origin

The estimator for level *j* is

$$\hat{p}_j = \frac{h_j}{n_j}.$$

To generate h_j the simulation launches runs from states in L_{j-1}. Which states? How are they chosen?

The critical issue is the fact that, in general, the probability of a run hitting L_j depends on the state in L_{j-1} from which the run starts. In general those probabilities differ, state to state. Let:

q_{kj} = the probability that a run in L_j will start in state *k* (in L_{j-1})

p_{kj} = the probability that a run from state *k* will hit L_j

p_j = the probability that a run from L_{j-1} will hit L_j

Then

$$p_j = \sum_k q_{kj}\, p_{kj}$$

Neither q_{kj} nor p_{kj} is known. They must be estimated, <u>indirectly</u>:

$$\hat{p}_j = \sum_k \hat{q}_{kj}\, \hat{p}_{kj}$$

They are not estimated explicitly. Only the level estimator that can be calculated:

$$\hat{p}_j = \frac{h_j}{n_j}.$$

These two versions of the level estimators should be equivalent. They are made equivalent by the way in which the simulation selects states in L_{j-1} from which to launch the runs n_j.

The runs in L_j should be probabilistically equivalent to what simple Monte Carlo would have produced. That means that q_{kj}, the probability that a run will start in state *k* in L_{j-1}, should be the probability that an MC run that hits L_{j-1} will hit it at state *k*. We can estimate that probability from the hits on L_{j-1} under splitting. Their empirical distribution is an estimate of the distribution of the hits on L_{j-1} under MC.

Consider q_{k1}, the probability that a run in L1 starts at state *k* in L0. In the test model there is only one state in L0, so $q_{(0,0),1} = 1$. In general, however, there could be more than one state in L0. The simulation would need to calculate, or estimate, q_{k1}.

Now consider q_{k2}, the probability that a run in L2 will start at state *k* in L1. It can be estimated from the distribution of the states, in L1, that were actually hit by runs in L1. For example, in the test model (12, 0) and (12, 1) are states in L1. If 0.15 of the hits on L1 hit (12, 0), and 0.1 hit (12, 1), then we have the estimates:

$$\hat{q}_{(12,0),2} = 0.15 \qquad \hat{q}_{(12,1),2} = 0.1$$

Over all the states in L1 we have:

$$\sum_k q_{kj} = \sum_k \hat{q}_{kj} = 1$$

The simulation does n_2 runs in L2. If it launches $\hat{q}_{k2}\, n_2$ (rounded) from each state *k* it will approximate the real $q_{k2}\, n_2$, and will be statistically equivalent to MC in the sense that $E\left[\hat{q}_{k2}\right] = q_{k2}$. This line of reasoning then applies to $\hat{q}_{kj}$ through all levels *j*, each $E\left[\hat{q}_{kj}\right] = q_{kj}$ following from the unbiasedness of the prior level.

$\hat{q}_{kj}$ is calculated as the proportion of the hits on L_{j-1} that hit state *k*. That proportion is known since, under fixed effort splitting, all the runs in L_{j-1} (and thus hits on L_{j-1}) are completed before any runs are launched in L_j. The algorithm must, therefore, keep a record (list, array) of the states hit in L_{j-1}. That record must include replications. If state *k* is hit 100 times, there should be 100 copies of state *k* in the record. Or, equivalently, the record could include the percentages of hits on each state *k*, or a count of the number of hits on each state *k*.

This record keeping, though burdensome in terms of computational overhead and use of memory, can only be dispensed with if the q_{kj} are the same for all *k*, and known to be so. At any point in the implementation of splitting two such records are required: the set of entrance states for the level in which runs are currently being conducted, and a new set, for the next level, created by current hits on the next level.

The states in that record are called entrance states. The set of entrance states for L_j is the set of states in L_{j-1} hit by runs in L_{j-1}, with replication. Given that set, the

algorithm estimates q_{kj} in practice by launching each run in L_j from a state randomly selected from the set. The random selection is with replacement.

Define:

h_{kj} = number of hits, on L_j, from runs commencing in state *k* of L_{j-1}

n_{kj} = number of runs, in L_j, commencing in state *k* of L_{j-1}

Then: $h_j = \sum_k h_{kj}$

Given that a run in L_j starts from a state randomly selected from the saved set of entrance states in L_{j-1}:

$$\hat{q}_{kj} = \frac{n_{kj}}{n_j} \qquad \hat{p}_{kj} = \frac{h_{kj}}{n_{kj}}$$

$$\hat{p}_j = \sum_k \hat{q}_{kj}\,\hat{p}_{kj} = \sum_k \frac{n_{kj}}{n_j}\frac{h_{kj}}{n_{kj}} = \frac{1}{n_j}\sum_k h_{kj} = \frac{h_j}{n_j}$$

The two formulations of the level estimator are equivalent if each run is selected at random from the saved set of entrance states. The larger the set of entrance states the better the estimator (the smaller its variance), since the larger the set, the better it empirically approximates the true q_{kj}.

Large sets can pose a burden on memory, but it should be noted that, in this simple form of splitting, a set of entrance states need be saved only to supply the base from which to launch runs at the next level. The set of entrance states in L_{j-1} is needed

for runs in L_j but not thereafter, since all runs in each level are completed before any runs in the next level are started. This will not be true of the optimizing algorithms to be considered. As will be seen, they must save all sets of entrance states at all levels throughout the implementation of the algorithm.

4.4 The Design Estimator Is Unbiased

The question posed earlier – how can the unbiasedness of the estimators be maintained while their variances are decreased? -- can now be answered. The Monte Carlo design estimator is unbiased. Garvels (2000) proves that the splitting design estimator is also unbiased. For *L* levels:

$$E[\hat{p}] = E[\hat{p}_1 \hat{p}_2 \cdots \hat{p}_L] = p$$

Unbiasedness requires a method for selecting each run in L_j from a state in L_{j-1} in a way that is probabilistically equivalent to the random selection under fixed effort splitting described above. Garvels' proof is stated in the context of fixed splitting, but generalizes to fixed effort splitting. It does not require that the level estimators be independent.

Lower variances are achieved under splitting precisely because the number of runs in any level becomes a decision variable that can be set at any value without sacrificing unbiasedness. The larger they are set, the smaller the variances of the level estimator, the smaller the variance of the design estimator.

4.5 Distribution of the Level Estimators

Let X_{kj} be the outcome of a run in level *j*, from entrance state *k*. X_{kj} = 1 if the run hits L_j, 0 otherwise. X_{kj} is a Bernoulli random variable, with probability p_{kj}, mean p_{kj} and variance $p_{kj}\left(1-p_{kj}\right)$. The number of hits, h_{kj}, over n_{kj} runs, is a binomial random variable with mean $n_{kj}\ p_{kj}$ and variance $n_{kj}\ p_{kj}\left(1-p_{kj}\right)$. The estimator, $\hat{p}_{kj}=\frac{h_{kj}}{n_{kj}}$, is a scaled binomial (a binomial multiplied by a constant).

This simplicity does not, in general, extend to a run in level *j* without specification of the entrance state from which it starts. Let X_j be the outcome of such run: X_j = 1 if the run hits L_j, 0 otherwise. The outcome is binary, but the probability of that outcome is not the same on every run. Thus X_j is not Bernoulli. It has a mixture distribution.[12]

Let

$$f_{kj}\left(x\right) \ = \ \text{the Bernoulli } pmf \text{ of } \ X_{kj}$$

$$f_j\left(x\right) \ = \ \text{the } pmf \text{ of } \ X_j$$

Recalling that q_{kj} is the probability that a run in L_j starts in state *k* (of L_{j-1}), the *pmf* of X_j is

$$f_j\left(x\right) \ = \ \sum_k q_{kj}\ f_{kj}\left(x\right)$$

[12] For a treatment of mixture distributions, see Hogg, McKean & Craig, 2005, p. 189

with moments

$$E\left[X_j\right] = \sum_k q_{kj}\, p_{kj} = p_j$$

$$Var\left[X_j\right] = \sum_k q_{kj}\sigma_k^2 + \sum_k q_{kj}\left(p_{kj} - p_j\right)^2$$

where σ_k^2 = the variance of a run from state *k*, which is $p_{kj}\left(1-p_{kj}\right)$.

$f_j\left(x\right)$ can be understood as follows: To generate a drawing from this distribution, first select the *k* from the distribution of the q_{kj}. Then generate a drawing form the corresponding $f_{kj}\left(x\right)$. That outcome is a drawing from $f_j\left(x\right)$.

The variance can then be expressed as:

$$Var\left[X_j\right] = \sum_k q_{kj}\left(p_{kj} - 2p_{kj} + p_j^2\right)$$

In the exceptional case where all q_{kj} are equal (all $p_{kj} = p_j$), this variance reduces to

$$Var\left[X_j\right] = p_j\left(1-p_j\right),$$

the variance of a Bernoulli random variable.[13]

Thus, when all the q_{kj} are the same, the number of hits from n_j runs in level *j* will be binomially distributed, and $\hat{p}_j$ will be a scaled binomial. It will not be a scaled binomial if they are not equal. It will have the same mean as the scaled binomial, but a

[13] To derive this, recall that $\sum_k q_{kj} = 1$.

different variance, which will depend on the q_{kj} and p_{kj} . This dependence will pose a problem for the optimizing algorithms, discussed below.

4.6 The Importance Function

The literature on splitting is fond of a formalism called the "importance function". It is, formally, a function from each state of the stochastic process to a number, with the property that all states in the subset between levels L_{j-1} and L_j have values less than those in the subset between levels L_j and L_{j+1}, for all j. States in L_j have a common value, which is greater than any value in the subset between L_{j-1} and L_j and less than any value in the subset between L_j and L_{j+1}.

The importance function formalizes the basic property splitting must have: a partitioning of the state space (see page 45). From an explicit importance function one could deduce the number of levels and their location, or placement. The literature claims, correctly enough, that the importance function is the most important decision to be made in constructing a splitting simulation. This means the important questions are: How many levels should there be? Where should they be? What shape should they have?[14]

For "standard" splitting these questions are the most important by default. They are the only questions, given that I use "standard" splitting to mean fixed effort splitting that allocates an equal number of runs (or amount of computing time) to each level. Assuming a given computing budget there are no decision variables left, after the number

[14] Poor choices for the importance function can yield poor results, even if the choices seem sensible. See Glasserman et al., 1998.

and placement of the levels are decided. Of course the simulation designer could create additional decision variables by deciding on some other allocation (other than equal) of effort among the levels, guided, perhaps, by judgment or heuristics. But for standard splitting the only things to decide are how many levels to create, where to put them, and how they should be shaped.

These questions can be important. Too few levels will not harness the potential efficiency of splitting, too many can become computationally burdensome. Poor placement can also undermine importance. Some theoretical guidance is available. Garvels and Kroese (1998) show that there should be about *m* level estimators, all equal at about

$$p_k \approx e^{-2}.$$

We then have

$$p = \prod_{k=1}^{m} p_k = \prod_{k=1}^{m} e^{-2} = e^{-2m} \quad \Rightarrow \quad m = \frac{-\log(p)}{2}$$

These results are based on the very restrictive assumption that the outcomes of all runs, at all levels, are independent, identically distributed random variables. With this assumption all level probabilities are the same -- an assumption so extreme that, in any practical problem, the theoretical result is essentially useless. Moreover, without prior knowledge of p -- precisely the thing the simulation is designed to estimate! -- these estimates cannot be directly employed. One might take an informed guess at what p is expected to be, or do some preliminary runs to get a rough estimate, then choose *m* accordingly. That alone, however, is no help in designing the placement or shape of the

levels, since one would have to know, in advance, how they would generate approximately equal level estimators.

Absent such prescience it would be normal to locate the levels at approximately equal distances from each other, though for some stochastic processes the concept of the "distance" of one level from the other may not be well defined. In the test model the distances could be defined in several ways. I choose the simplest by setting them the levels at 12, 25, 38, and 50 for the number of customers in either subsystem. (See sec. 2.2 for a description of this test model). Since the exact design probabilities for the designs in the test model are known, the optimal *m* could be approximated. For design 1, with $p =$ 1.37988924E-7, we have:

$$m = \frac{-\log(p)}{2} \approx 3.4 .$$

Either 3 or 4 levels would thus be defensible choices for *m*. But where should they be placed to equalize the level probabilities?

Optimizing algorithms, such as those I consider below, may be thought of as attempts to compensate for suboptimal placements of the levels by optimizing the number of runs (or time) per level.

4.7 Splitting vs OCBA

Though splitting was developed for application to a single design, it can be used to simulate multiple designs, for the purpose of selecting the best, by simulating them independently and comparing the results. The simulation of each design consumes whatever portion of the computing budget is allocated to it. Splitting per se provides no

guidance to that allocation. The simple choice, absent some heuristic to guide the allocation, would be equal amounts to each design.

OCBA optimizes the allocation of the computing budget across designs, but suffers from the poor performance of MC as the simulation technique within a design. Splitting is a better method for simulating designs with rare events, but lacks a workable rule for the optimal allocation of the computing budget across designs. Splitting nonetheless significantly outperforms OCBA at selecting the best design, in the context of rare events, as demonstrated in the following test. Splitting, with 4 levels, is compared to OCBA in selecting the best of 8 designs. Under splitting the allocation of the 20 minute computing budget is equal across the designs. The PCS is estimated on the basis of 2000 experiments.

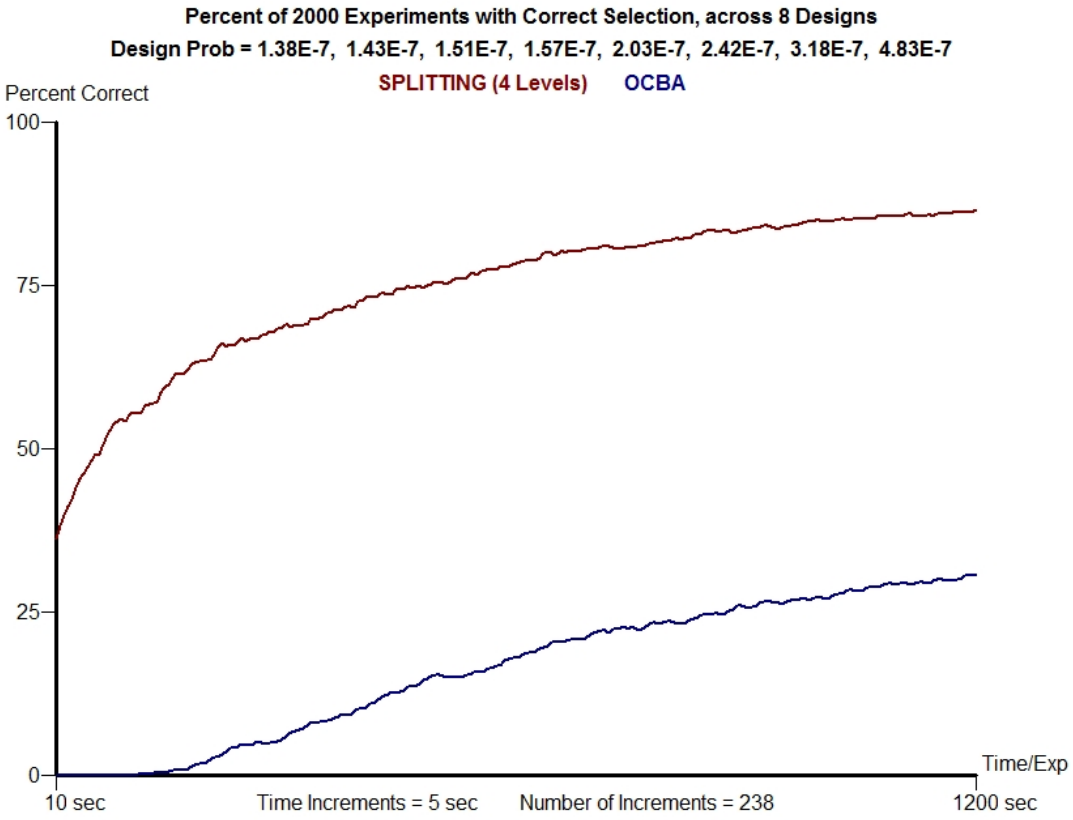

Figure 18: Splitting vs OCBA, 8 Designs

5. OSTRE

Constants

L = number of levels

L_j = level j, $j = 1,\ldots,L$

p = probability of the rare event

p_j = level probability for level j (probability of a run in L_j hitting L_j)

q_{kj} = probability of starting a run in L_j from state k in L_{j-1}

p_{kj} = probability of a run starting in state k of L_{j-1} hitting L_j

b_j = average time for one run in L_j

T = computing budget (for 1 design), in time

Random Variables

$\hat{p}$ = estimator of p

$\hat{p}_j$ = estimator of p_j

$\hat{b}_j$ = estimator of b_j

h_j = number of hits on L_j

Decision Variables

n_j = number of runs in level j, subject to the constraint that $\sum_j b_j n_j = T$

The Optimal Splitting Technique for Rare-Event Simulation (OSTRE), developed by Shortle, Chen, *et al*. (2012)[15], is a technique for optimizing the allocation of runs among levels within a single design. Though applicable to any design that can be

[15] As of this writing this dissertation, which extends OSTRE by combining it with OCBA and applying it to multiple designs, is the only additional literature on OSTRE.

simulated by splitting, it was developed specifically to enhance the efficiency of rare event estimation.

OSTRE is a refinement of fixed effort splitting. Under standard fixed effort splitting the computing budget assigned to *n*-1 out of *n* levels of the design is a decision variable. (The allocation to the remaining level is then determined by the budget constraint.) Absent any guidance from judgment or heuristics it would be natural to allocate the computational effort (time or runs) equally among the levels. The splitting examples and tests in this research are based on equal allocation among the levels.

Instead of equal allocation OSTRE tries to optimize the allocation, to minimize the variance of the estimator of the rare event probability. The key is to express that variance in a form that can be minimized.

As noted above, the splitting variance is a difficult creature to capture. To pin it down precisely would require knowledge of the q_{kj} (the probability that a run in L_j is launched from state *k* in L_{j-1}) and of the p_{kj} (the probability that such a run would hit L_j), for all *j* and *k*. Implementing an algorithm based on the minimization of the exact variance would require estimating all these probabilities. In practice that would likely be a bridge much too far. A tractable approximation to the variance is required.

These difficulties disappear if all the p_{kj} are equal. There is then but a single probability of a run in L_j hitting L_j: for all *k*, $p_{jk} = p_j$, the unconditional level

probability (not conditioned on the starting state). If this is true the level estimators, the $\hat{p}_j$, are distributed as scaled binomials, with reasonably tractable variances.

5.1 OSTRE's Major Assumption

To attain tractable level variances OSTRE assumes that the probability that a run in L_j will hit L_j is independent of its starting state. OSTRE solves the exact minimization problem only if that assumption is exactly true. Otherwise it solves a (closely?) related problem, and its solution is an approximate solution. A good approximation does not require that all the p_{jk} be almost equal. Even if some p_{jk} differ considerably from the average, the approximation could still be quite good if the probabilities of runs being launched from those starting states, the q_{jk}, are remote.

Under this assumption the level estimators are independent: $\hat{p}_j$ is independent of $\hat{p}_{j-1}$ because, since it matters not from which states runs in L_j commence, it matters not which states in L_{j-1} are hit by the runs in L_{j-1} that generate $\hat{p}_{j-1}$. In fact, $\hat{p}_j$ would be unbiased, with the same variance, if all the runs in L_j commence in states not hit by <u>any</u> runs in L_{j-1}.

Even if OSTRE's major assumption is not well approximated OSTRE might still be worth implementing. Attaining the optimum is a theoretical objective. In practice the implementation of the algorithm will almost never attain it, no matter how rigorous the theory, no matter how justified the assumption. In practice the purpose is to improve the efficiency of rare event estimation, compared to splitting based on an arbitrary (e.g.,

equal) allocation of computational effort among the levels. OSTRE could still be useful, by delivering some improvement, even if its major assumption does not closely hold.

5.2 The OSTRE Assumption on the Test Model

How well does the test model approximate the major OSTRE assumption? This can be examined, empirically. We know the exact level probabilities in the test model. The variances of the level estimators can therefore be calculated under the OSTRE assumption and compared to sample variances generated by many standard splitting simulations of the test model. The following comparison is performed on design 1, which is simulated 1000 times (with 1 million runs at each level of each simulation):

Table 2. OSTRE Assumption Variance vs Sample Variance

	Variance under OSTRE Assumption	Sample Variance from Simulations
Level 1	3.65 E-8	3.8 E-8
Level 2	1.55 E-8	2.21 E-8
Level 3	1.28 E-8	2.58 E-8
Level 4	1.74 E-8	6.5 E-8

The close agreement between the exact and sample variances of level 1 is no surprise. Level 1 satisfies the OSTRE assumption. Every run in L_1 starts from the same point, (0, 0), so has the same probability of hitting L_1. The estimators for levels 2-4, however, do not satisfy the OSTRE assumption, though the approximation is not bad. The approximation appears weak at level 4. This should degrade OSTRE's performance

somewhat. And not just OSTRE's performance. The new techniques introduced in this research are also based on the OSTRE assumption. If they perform well, that will suggest that these optimizing algorithms are not highly sensitive to that assumption.

5.3 OSTRE Derivation

Under the OSTRE assumption the level hits are binomials and the level estimators are scaled binomials.

With h_j = number of hits on L_j , n_j = number of runs in L_j :

$$h_j \sim Binomial\left[p_j, n_j \right] \qquad \hat{p}_j = \frac{h_j}{n_j}$$

$$E\left[\hat{p}_j \right] = p_j \qquad Var\left[\hat{p}_j \right] = \frac{p_j\left(1-p_j\right)}{n_j}$$

A closed form expression for the variance of the design estimator, with *L* splitting levels, can be derived:

$$Var\left[\hat{p} \right] = Var\left[\hat{p}_1 \bullet\bullet\bullet \hat{p}_L \right]$$

Under the OSTRE assumption this can be expanded and expressed as:

$$Var\left[\hat{p} \right] = \left[p_1^2 + \frac{p_1\left(1-p_1\right)}{n_1} \right] \bullet\bullet\bullet \left[p_L^2 + \frac{p_L\left(1-p_L\right)}{n_L} \right] - \left[p_1 \bullet\bullet\bullet p_L \right]^2$$

The OSTRE optimization problem[16] becomes

[16] It should be noted that OSTRE minimizes the "exact" problem (given the OSTRE assumption): it minimizes the variance of the design estimator. This contrasts with OCBA, which minimizes the Bonferroni approximation to its exact problem.

$$\underset{n_k}{Min}\ Var\left[\hat{p}\right] \quad \text{s/t} \quad \sum_{k=1}^{L} b_k n_k = T \ ,$$

where $b_k =$ average time of a run in level *k*, and *T* = computing budget (in time).

It can be shown that this is a convex minimization problem. Its optimality equations are

$$n_i\, b_i\left(1+\frac{n_i\, p_i}{1-p_i}\right) \;=\; n_j\, b_j\left(1+\frac{n_j\, p_j}{1-p_j}\right) \quad i, j \in \{1,...,L\} \quad L = \text{ number of Levels}.$$

Chen & Shortle suggest a simplification of these equations, justified when $n_j >> \left(1-p_j\right)/\, p_j$. The equations then reduce to:

$$\frac{n_i^2\, b_i\, p_i}{1-p_i} \;=\; \frac{n_j^2\, b_j\, p_j}{1-p_j}$$

$$\frac{n_i}{n_j} \;=\; \sqrt{\frac{b_j\, p_j\left(1-p_i\right)}{b_i\, p_i\left(1-p_j\right)}}$$

With this simplification, together with the budget constraint, the optimal n_i are easily found. I use this form of the optimality equations in implementing OSTRE.

This simplification can also be derived in a different way, one that generalizes to the new techniques I propose. This is not a minor issue. Quite the contrary, it will be central to the demonstration that those techniques are mathematically equivalent. The simplification is based on a truncation of the expansion of the variances of the level estimators. See Appendix 4 for details and a proof that this truncation generates the same simplification of the optimality equations as the assumption that $n_j >> \left(1-p_j\right)/\, p_j$.

5.4 OSTRE Implementation

Implementing OSTRE differs in one significant feature from implementing standard fixed effort splitting. Under standard splitting all the runs at a level are completed prior to any runs at the next level. This lightens the charge on memory from the necessity of maintaining a record of the entrance states into a level, since that record can be deleted once all the runs that depend on it are completed. At any point in time it is necessary to maintain only two such lists: the set of entrance states for the level in which runs are being conducted, and the set being built up, for the next level, from runs hitting the next level.

This ease vanishes in OSTRE. As with all the optimizing algorithms in this research, sets of entrance states must be built up, and maintained, at all levels, throughout the entire implementation or the algorithm.

OSTRE begins, as does OCBA and all the optimizing algorithms, with an initiation phase, required to generate initial estimates of the parameters required by the optimization. (For OSTRE, the $\hat{p}_j$ and $\hat{b}_j$.) Then, at regular updates, the optimality equations (simplified form) are solved, generating optimal values for the n_j. The budget constraint used to generate each solution is the portion of the total computing budget consumed at the point of the update. The next allocation of runs is then made to the level which will make the largest contribution to reducing the actual inequalities of the optimality relationships. As with OCBA there are several ways to determine where that allocation should go. With frequent updates, and an adequate total computing time, all

reasonable methods should work well. I apply a simple one: give, at each update, the next allocation of runs to the level whose current runs fall the farthest short, proportionately, from their optimum, as determined by the current solution of the optimality equations.

Since, at each update, the next allocation of runs can be made to any level, the current set of entrance states into that level must be on hand. As those runs are then made they will generate new hits on the next level, thereby expanding the set of entrance states for that level. The complete sets of entrance states, for all levels, are maintained, and expanded, throughout the implementation of the OSTRE algorithm.

In addition to burdening memory, this implementation procedure introduces a small distortion. In OSTRE as in standard splitting the entrance state from which a run is launched is selected randomly from the set of entrance states. In standard splitting that set is always complete. It was constructed from all prior hits on that level. Since the random sampling is with replacement, at each selection of an entrance state for a new run every state in the set of entrance states (counting copies, or replications) has the same probability of being selected. Consequently, $\hat{q}_{kj}$ is an unbiased estimator of q_{kj}. But not so in OSTRE. At each OSTRE update it will conduct new runs at some level, thereby potentially generating new hits that create new entrance states. The sets of entrance states expand as the algorithm proceeds. Each time a set is randomly sampled it contains, potentially, new entrance states, states that were not present that previous time it was sampled, or not present in the same proportion. Thus, at each sampling not every entrance state (counting copies) will have the same probability of being selected. States

which entered the set early in the evolution of the algorithm will have been there longer (through more updates) than states which entered later. By the end of the algorithm they will have had, cumulatively, more chances to have been selected than states entering later in the process. Systematically more chances, that is, than warranted by their true probability of being selected (the true q_{kj}). In any complete set of entrance states the proportion comprised by state *k* will likely not be the true q_{kj}, but its estimator under random sampling, $\hat{q}_{kj}$, will be unbiased. Not so the OSTRE $\hat{q}_{kj}$, though the bias may be very slight. Correcting, or ameliorating, this bias could be a topic for future efforts to construct good OSTRE implementations. It is not pursued in this research. All tests involving OSTRE (which include the new algorithms developed below) are conducted with no effort to counteract this bias.

OSTRE does not suffer, as does OCBA, from false leads. False leads arise only in the context of optimizing allocations among designs, not within levels of a design. Within OCBA theory the putative best design plays a special role. It is not interchangeable with other designs, so getting it wrong could severely distort allocations guided by the theory. Within OSTRE theory no level plays a special, or different, role, compared to the other levels. No level "leads", so there can be no false leads, no matter how badly the algorithm errs in its estimates.

5.5 OSTRE and Multiple Designs: OSTRE vs Splitting

OSTRE was developed to efficiently estimate the rare event probability of a single design. There is an immediate extension to two designs, but not beyond that, in

the original research. The initial impetus for this dissertation was precisely to extend OSTRE to an arbitrary number of designs.

OSTRE can nonetheless be applied to a PCS problem, for any number of designs, in the same manner as standard splitting: arbitrarily allocate the time (runs) among the designs, and estimate each design independently of the others. As with standard Splitting, the tests below make the natural allocation of equal effort across all designs.

The comparison is between OSTRE and standard splitting, across 6 designs. OSTRE should improve upon Splitting if it surmounts the weaknesses described above: failure of its major assumption and bias in its selection of entrance states.

OSTRE's performance should not improve as the number of designs increases, since, like standard splitting, it allocates computing effort equally among designs. The total computing budget is 20 minutes, over 2000 experiments. Note that the graph's lower bound is 35%, not 0% .

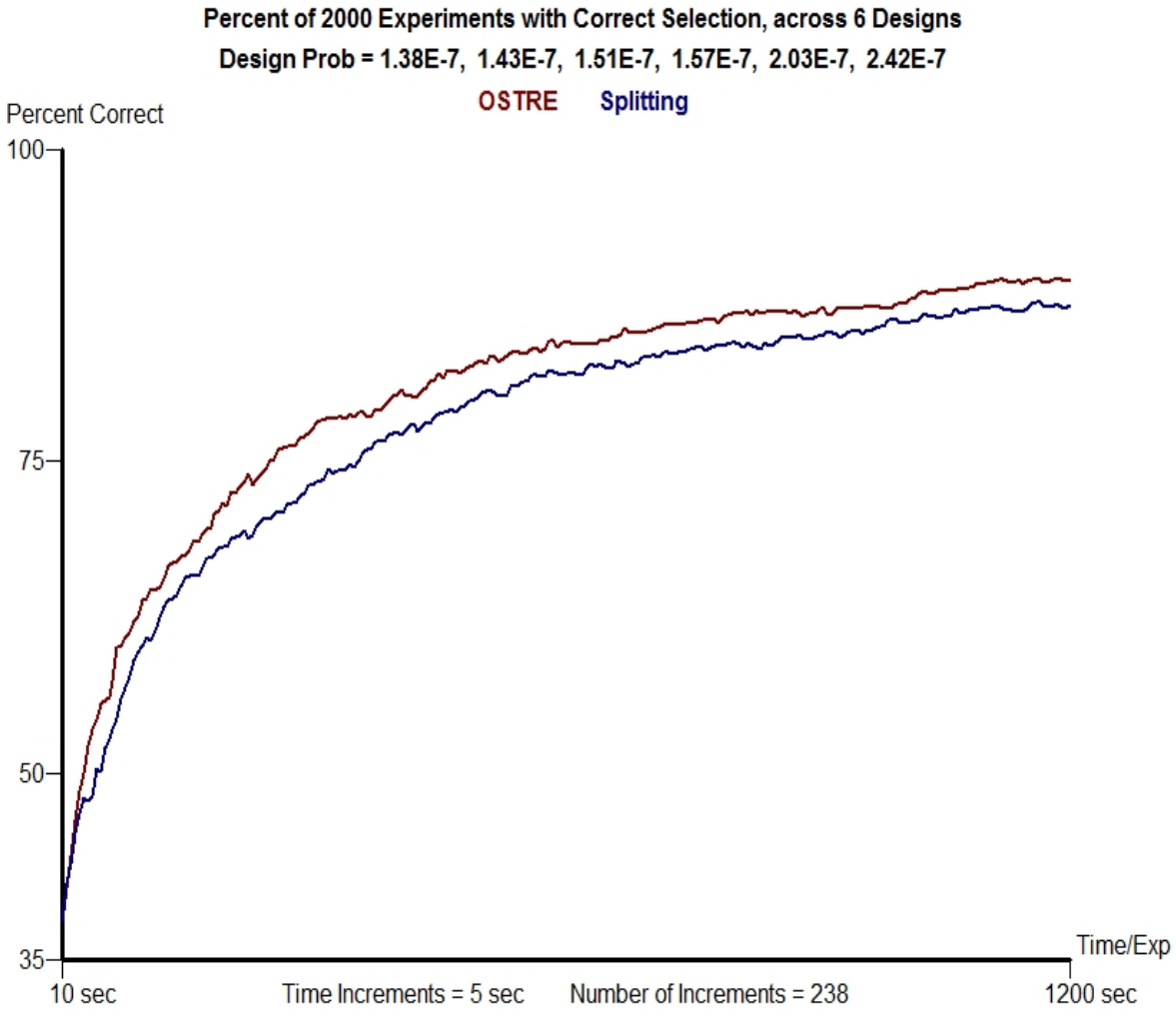

Figure 19. OSTRE vs Splitting, 6 Designs

OSTRE wins, and the margin of victory is greater than it might appear on the graph. OSTRE reaches 75% PCS at 235 sec., splitting at 360 sec. OSTRE maintains its superiority over splitting for 8 and 10 designs:

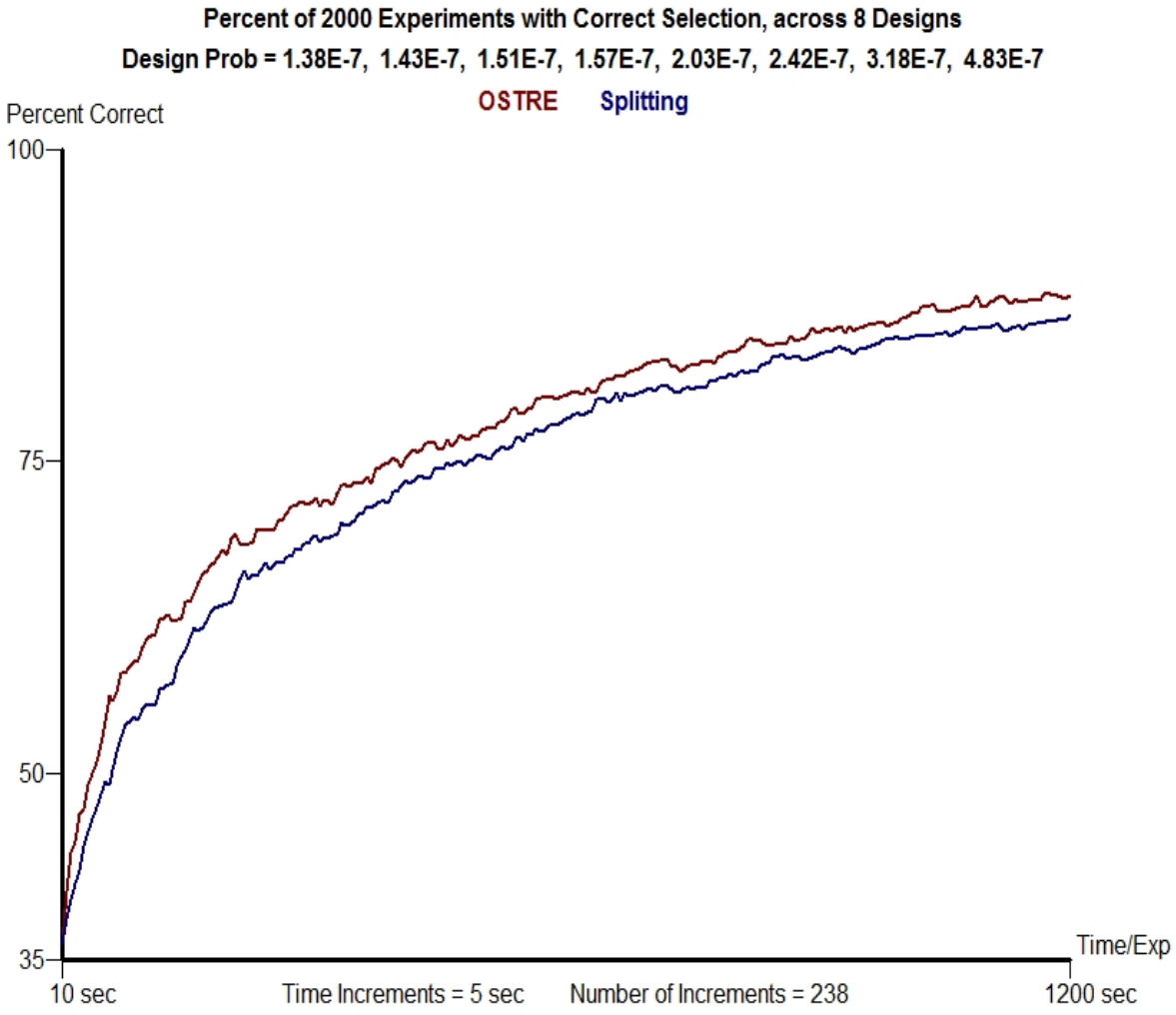

Figure 20. OSTRE vs Splitting, 8 Designs

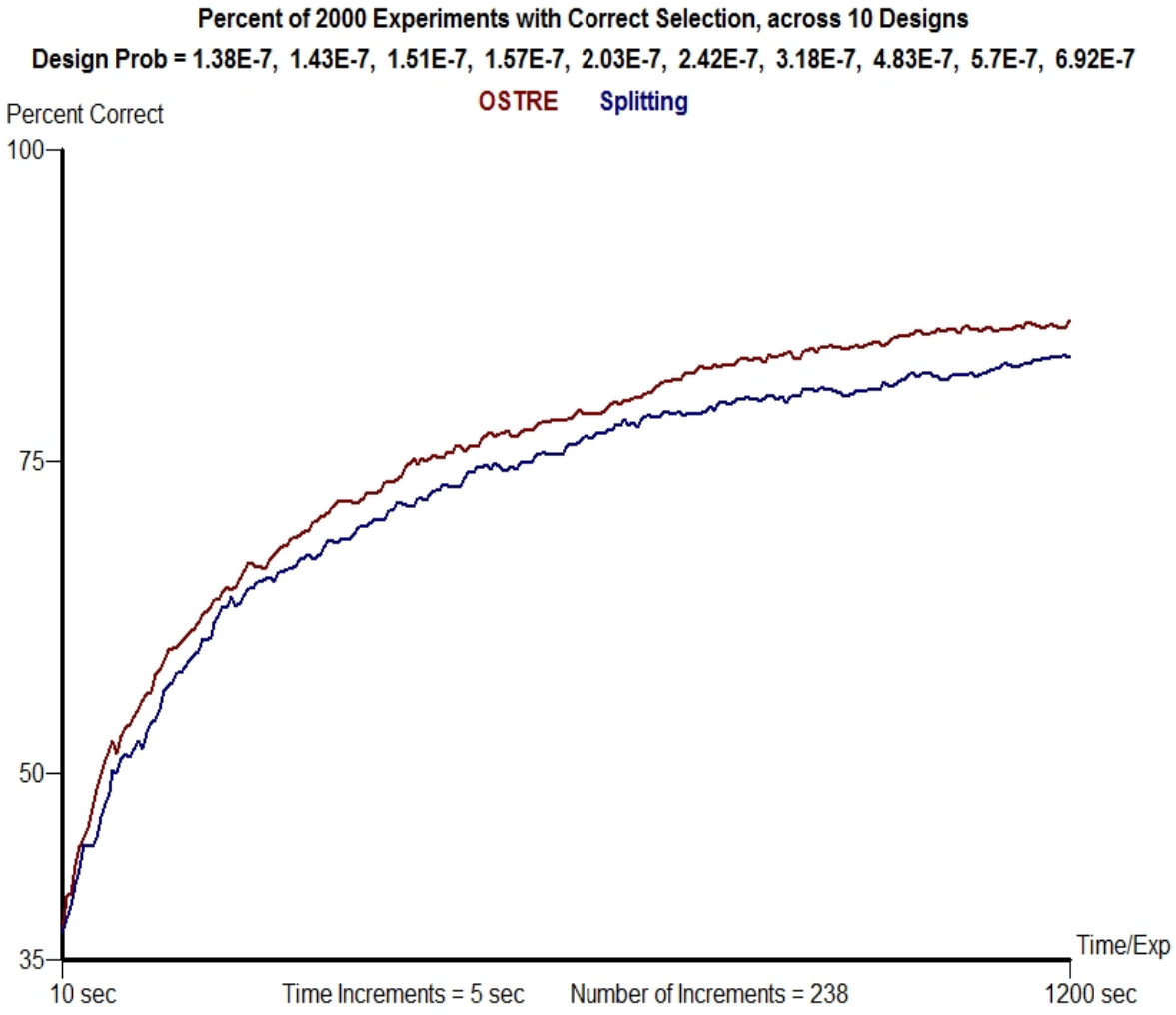

Figure 21. OSTRE vs Splitting, 10 Designs

6. Single Optimization

Constants

D = number of designs

L = number of levels

L_{ij} = level j in design i.

p_i = probability of the rare event occurring in design i

p_{ij} = probability of hitting level L_{ij}, starting in level L_{ij-1}

b_{ij} = average time of a run in L_{ij}

T = budget constraint, in time

Random Variables

h_{ij} = number of hits in L_{ij}

$\hat{p}_i$ = estimator of p_i

$\hat{p}_{ij}$ = estimator of p_{ij}

Decision Variables

n_{ij} = number of runs in L_{ij}

$t_{ij} = b_{ij} n_{ij}$ = time for L_{ij}

In theory the standard approach to maximizing the PCS is a single optimization over all the decision variables -- the simultaneous allocation of time (or runs) directly to all the splitting levels of all the designs. I call this Single Optimization, to contrast it with the two-stage approach developed in the next chapter.

Assuming, wlog, that design 1 is the best, the optimization problem may be formulated as:

$$\underset{n_{ij}}{Max}\, P\left[\hat{p}_1 < \hat{p}_2, ..., \hat{p}_1 < \hat{p}_D \mid p_1 < p_2, ..., p_1 < p_D\right] \quad \text{s/t} \quad \sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij} n_{ij} = T, \quad n_{ij} \geq 1$$

This is almost identical to the OCBA maximization problem, except that the maximization is over the runs allocated to the splitting levels, not to the designs. The event to be maximized is the intersection of *D*-1 mutually dependent events , $\bigcap_{i=2}^{D}\left(\hat{p}_1 < \hat{p}_i\right)$. As with OCBA, stating this probability as a closed-form function that can be maximized is too difficult. As with OCBA, the Bonferroni approximation is adopted.

The Bonferroni inequality provides a lower bound to the PCS:

$$\text{PCS} \geq 1 - \sum_{i=2}^{D} P\left[\hat{p}_1 > \hat{p}_i\right].$$

Maximizing the expression on the RHS of this inequality generates, via the Bonferroni approximation, the highest lower bound on the PCS that can be attained, within a given budget constraint and a given number of splitting levels. (The splitting levels, number and placement, are given prior to the optimization.) This is a lower bound on a probability. It guarantees no particular outcome for any single outcome of the stochastic process.

It is convenient to state the optimization problem as a minimization, by noting that the RHS is maximized when $\sum_{i=2}^{D} P\left[\hat{p}_1 > \hat{p}_i\right] = \sum_{i=2}^{D} P\left[\hat{p}_1 - \hat{p}_i > 0\right]$ is minimized.

The problem becomes:

$$\underset{n_{ij}}{Min} \sum_{i=2}^{D} P\left[\left(\hat{p}_1 - \hat{p}_i > 0\right)\right] \quad \text{s/t} \quad \sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij} n_{ij} = T$$

The requirements that the $n_{ij} \geq 1$, and that they, in practice, be integers, can be ignored, for the same reasons these requirements on N_i were ignored in OCBA.

In practice this optimization problem is not solved analytically. As with OCBA the theory guides the algorithm in its allocation of new runs to the various levels at frequent updates. Single optimization's updating procedure is the same as that for OCBA, with the (major) exception that the updating allocates runs to the levels, not the designs. The theory guiding this updating is developed below.

6.1 Normal Distribution of the Design Estimators

Minimizing the objective function requires the distribution of the random variables to be approximated by a known distribution -- a function to which standard optimizing techniques can be applied. For these random variables a normal approximation can be justified, as follows.

The OSTRE Assumption

The (approximate) normality of the design estimators, $\hat{p}_i$, is built upon the (approximate) normality of the level estimators, $\hat{p}_{ij}$. For Single Optimization I adopt the OSTRE assumption that the probability of a run in L_{ij} hitting L_{ij} is the same for each possible entrance state in L_{ij-1}. Under this assumption the number of hits on L_{ij} is binomial, the level estimator a scaled binomial. We have the same formulation as in OSTRE, except that the variables and parameters must now be indexed by design (by *i*)

as well as by level (by *j*). The formulation applies to all levels across all design. For *D* designs and *L* levels:

$$h_{ij} \sim Binomial\left[p_{ij}, n_{ij} \right] \qquad i = 1,...,D \qquad j = 1,...,L$$

$$E\left[h_{ij} \right] = n_{ij}\, p_{ij} \qquad Var\left[h_{ij} \right] = n_{ij}\, p_{ij} \left(1 - p_{ij}\right)$$

$$\hat{p}_{ij} = \frac{h_{ij}}{n_{ij}} \qquad E\left[\hat{p}_{ij} \right] = p_{ij} \qquad Var\left[\hat{p}_{ij} \right] = \frac{p_{ij}\left(1 - p_{ij}\right)}{n_{ij}}$$

By the De Moivre-Laplace theorem, a binomial random variable with mean $n_{ij}\, p_{ij}$ and variance $n_{ij}\, p_{ij}\left(1 - p_{ij}\right)$ converges, in probability, to a normal random variable with the same mean and variance. A normal random variable with that mean and variance yields "excellent approximations"[17] to the binomial for values of n_{ij} and p_{ij} for which $n_{ij}\, p_{ij}\left(1 - p_{ij}\right) \geq 10$. Values for n_{ij} and p_{ij} typical in rare event simulations will easily satisfy that rule of thumb, even over quite modest computing budgets. In the test model, with 4 splitting levels, all p_{ij} exceed 0.01. Standard splitting, on the test model, generates a minimum of about 1 million runs in a level (and generally more), over a computing budget (for a single design) of about 4 minutes. Under these conditions we have $n_{ij}\, p_{ij}\left(1 - p_{ij}\right)$ at about 10,000.

[17] Ghahramani, 2005, p. 268.

Approximating the binomial distribution of h_{ij} by a normal distribution with the same mean and variance is thus entirely justified. It follows that the level estimator, $\hat{p}_{ij}$, may also be modeled, with strong justification, as a normal random variable:

$$h_{ij} \sim Normal\left[n_{ij}\, p_{ij},\ n_{ij}\, p_{ij}\left(1-p_{ij}\right)\right]$$

$$\hat{p}_{ij} = \frac{h_{ij}}{n_{ij}}$$

$$\hat{p}_{ij} \sim Normal\left[p_{ij},\ \frac{p_{ij}\left(1-p_{ij}\right)}{n_{ij}} \right]$$

The design estimators are products of their respective level estimators:

$$\hat{p}_i = \prod_{j=1}^{L} \hat{p}_{ij} \qquad i = 1,...,D.$$

The (approximate) normality of the design estimators follows from the claim that the product of normal random variables, of the type typical of level estimators, is approximately normal. *For each design i the design estimator, $\hat{p}_i$, is a normal random variable plus a sum of random variables each of which can be made arbitrarily close to 0 by sufficiently large n_{ij}.* The following discussion supports this claim:

Technically a constant is a random variable, albeit a trivial one, so the statement that a random variable converges to 0, in any mode of convergence, makes sense. Its mean and variance then both converge to 0.

Any random variable normally distributed can be written as the sum of its mean and a properly scaled standard normal. Thus, the level estimators, modeled as normal, for each design *i* and level *j*, can be expressed as:

$$\hat{p}_{ij} = p_{ij} + \sqrt{\frac{p_{ij}\left(1-p_{ij}\right)}{n_{ij}}}\, Z_{ij} \qquad Z_{ij} \sim Normal\left[0,1\right].$$

Recall that, under the OSTRE assumption, the level estimators are independent, so the distribution of each level estimator can be stated unconditionally, without reference to any other level estimator.

The product of the $\hat{p}_{ij}$ may be expressed as the sum of a number of terms:

$$\prod_{j=1}^{L} \hat{p}_{ij} = \prod_{j=1}^{L} p_{ij} + \sum_{j=1}^{L} c_{ij} \sqrt{\frac{p_{ij}\left(1-p_{ij}\right)}{n_{ij}}}\, Z_{ij} + Higher\,Order\,Terms$$

where the c_{ij} are constants, the product of the members of some subset of $\{p_{i1},..., p_{iL}\}$, and *Higher Order Terms* is the sum of terms containing products of more than one n_{ij} in the denominator. As all n_{ij} become large, the *Higher Order Terms* become negligible, relative to the other terms, which are constant or contain only one n_{ij}. Clearly, <u>all</u>

$n_{ij} \to \infty \;\Rightarrow\; \prod_{j=1}^{L} \hat{p}_{ij} \to \prod_{j=1}^{L} p_{ij}$, a constant. But, loosely put, the *Higher Order Terms* converge to 0 faster than the terms with only one n_{ij}. In the setting of rare event estimation the n_{ij} are finite but assumed to be large enough to justify ignoring the *Higher*

Order Terms. (For an analytical treatment of the convergence of the *Higher Order Terms*, see Appendix 6).

This justification requires <u>all</u> the n_{ij} to increase with an increasing computational budget: $T \uparrow \Rightarrow n_{ij} \uparrow,\ all\, i, j$. See Appendix 5 for proof that this is true asymptotically, $T \to \infty \Rightarrow all\ n_{ij} \to \infty$, under the condition that the n_{ij} maintain their optimum ratios as *T* increases. I assume it holds for finite T large enough to generate efficient rare event estimation.

This approximation to the normal is easily illustrated in the case of 2 independent normal random variables:

$$X_1 \sim N\left[\mu_1, \sigma_1^2\right], \quad X_1 = \mu_1 + \sigma_1 N[0,1] = \mu_1 + \sigma_1 Z_1$$
$$X_2 \sim N\left[\mu_2, \sigma_2^2\right], \quad X_2 = \mu_2 + \sigma_2 N[0,1] = \mu_2 + \sigma_2 Z_2$$
$$X_1 X_2 = \mu_1 \mu_2 + \mu_2 \sigma_1 Z_1 + \mu_1 \sigma_2 Z_2 + \sigma_1 \sigma_2 Z_1 Z_2$$

In rare event estimation the random variables in question are level estimators. Call them, suppressing the unneeded design index *i*, $\hat{p}_1$ and $\hat{p}_2$.

$$E\left[\hat{p}_j\right] = \mu_j = p_j, \quad Var\left[\hat{p}_j\right] = \sigma_j^2 = \frac{p_j\left(1-p_j\right)}{n_j}, \quad n_j = \text{ runs at level } j$$

Under the OSTRE assumption $\hat{p}_1$ and $\hat{p}_2$ are independent, so

$$\hat{p}_1 \hat{p}_2 = p_1 p_2 + p_2 \sqrt{\frac{p_1\left(1-p_1\right)}{n_1}} Z_1 + p_1 \sqrt{\frac{p_2\left(1-p_2\right)}{n_2}} Z_2 + \sqrt{\frac{p_1\left(1-p_1\right) p_2\left(1-p_2\right)}{n_1 n_2}} Z_1 Z_2$$

The last term becomes negligible as $n_1, n_2 \to$ large, leaving:

$$\hat{p}_1\hat{p}_2 \sim (approx)\ Normal\left[p_1 p_2,\ \frac{p_2^2 p_1(1-p_1)}{n_1} + \frac{p_1^2 p_2(1-p_2)}{n_2} \right]$$

I illustrate this approximation with a 3 level example. Let

$$p_1 = 0.04,\ p_2 = 0.05,\ p_3 = 0.06.$$

A histogram of the distribution of $\hat{p} = \hat{p}_1\hat{p}_2\hat{p}_3$, based on the complete expansion of the distribution (including the n_1n_2, n_1n_3, n_2n_3, $n_1n_2n_3$ terms) is constructed, and compared with a normal distribution with the same mean $(p_1p_2p_3)$, but with the variance it has when the higher order terms are neglected.

If the n_1, n_2, n_3 are "small", relative to the sizes they typically require in rare event simulation, the normal does <u>not</u> well approximate the actual distribution of $\hat{p}$. Letting $n_1 = n_2 = n_3 = 100$:

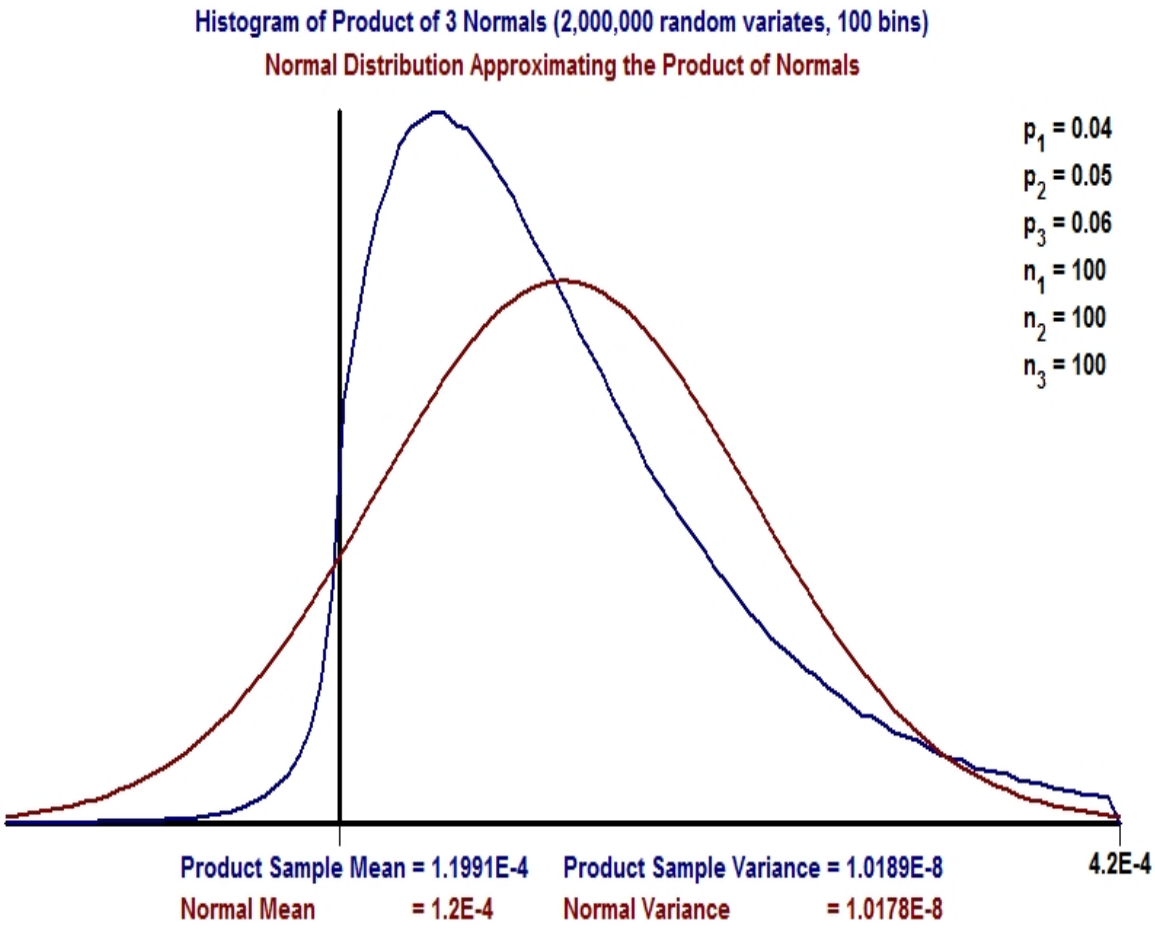

Figure 21. Normal Approximation: 100 Runs

Increasing each n_i to 10,000 generates a tight fit:

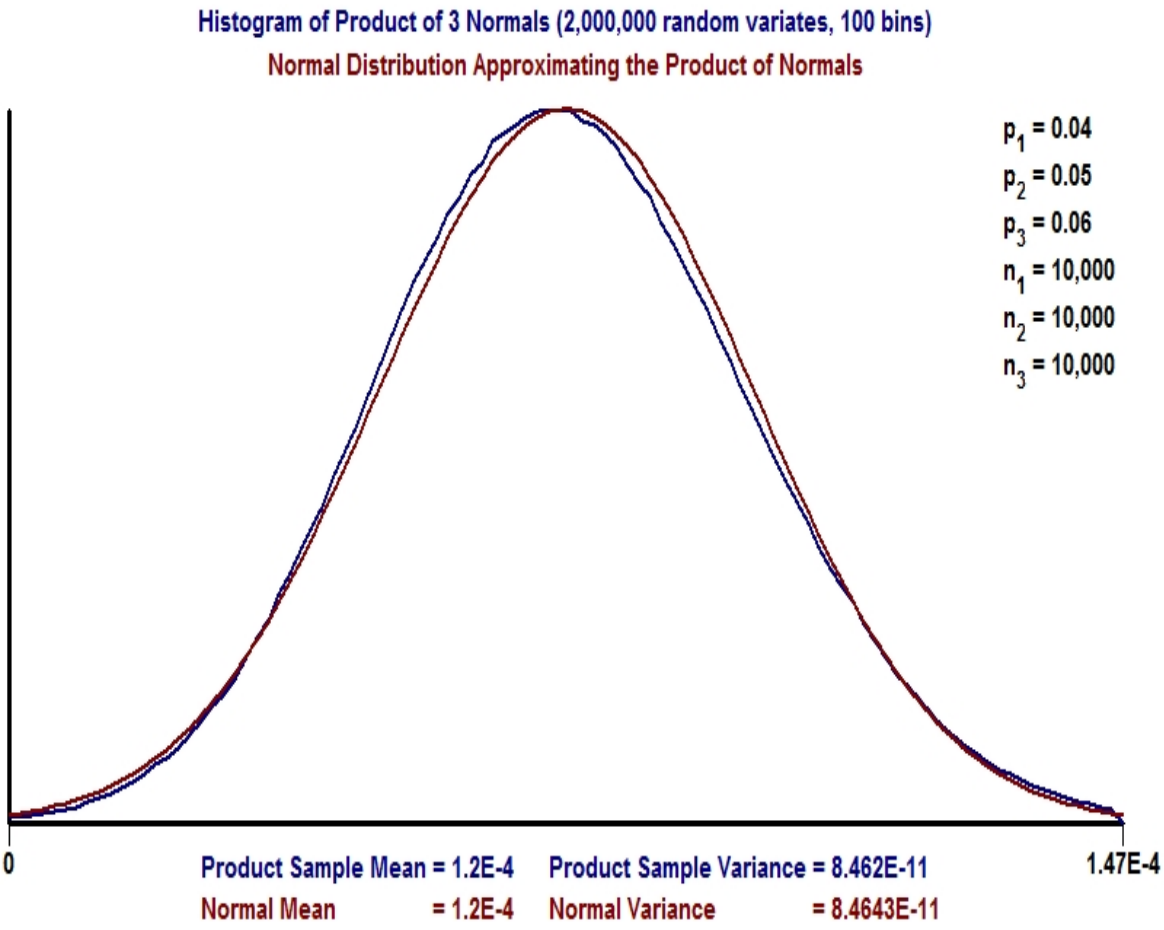

Figure 22. Normal Approximation: 10,000 Runs

In practical applications -- and certainly in my test model -- the runs at each level, over even modest budgets, will typically be several multiples of 10,000.

6.2 The Single Optimization Problem

$$\underset{n_{ij}}{Min} \sum_{i=2}^{D} P\left[\left(\hat{p}_1 - \hat{p}_i > 0\right)\right] \quad \text{s/t} \quad \sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij} n_{ij} = T\,,$$

The OSTRE assumption – that the probability of hitting L_j is the same for all runs in L_j -- is adopted. Under it the level estimators, the $\hat{p}_{ij}$, are independent. Each random variable $\hat{p}_1 - \hat{p}_i$ is (approximately) normally distributed. Thus:

$$E\left[\hat{p}_1 - \hat{p}_i\right] = p_1 - p_i \qquad Var\left[\hat{p}_1 - \hat{p}_i\right] = Var\left[\hat{p}_1\right] + Var\left[\hat{p}_i\right] = \sigma_1^2 + \sigma_i^2$$
$$\hat{p}_1 - \hat{p}_i \sim Normal\left[p_1 - p_i,\ \sigma_1^2 + \sigma_i^2\right]$$

For each *i*, the σ_i^2 are functions of all the p_{ij} and n_{ij} at each level *j*:

$$\sigma_i^2 = E\left[\hat{p}_i^2\right] - E\left[\hat{p}_i\right]^2 = E\left[\hat{p}_{i1}^2 \cdots \hat{p}_{iL}^2\right] - E\left[\hat{p}_{i1} \cdots \hat{p}_{iL}\right]^2$$

Under the OSTRE assumption the level estimators are independent, justifying:

$$\sigma_i^2 = E\left[\hat{p}_{i1}^2\right] \cdots E\left[\hat{p}_{iL}^2\right] - E\left[\hat{p}_{i1}\right]^2 \cdots E\left[\hat{p}_{iL}\right]^2$$

$$\sigma_i^2 = E\left[\hat{p}_{i1}^2\right] \cdots E\left[\hat{p}_{iL}^2\right] - p_{i1}^2 \cdots p_{iL}^2$$

For each design *i* and level *j*:

$$Var\left[\hat{p}_{ij}\right] = \frac{p_{ij}\left(1 - p_{ij}\right)}{n_{ij}} = E\left[\hat{p}_{ij}^2\right] - E\left[\hat{p}_{ij}\right]^2 = E\left[\hat{p}_{ij}^2\right] - p_{ij}^2$$

$$E\left[\hat{p}_{ij}^2\right] = p_{ij}^2 + \frac{p_{ij}\left(1 - p_{ij}\right)}{n_{ij}}$$

$$\sigma_i^2 = \prod_{j=1}^{L}\left[p_{ij}^2 + \frac{p_{ij}\left(1 - p_{ij}\right)}{n_{ij}}\right] - \prod_{j=1}^{L} p_{ij}^2$$

6.3 Simplification of the Variance

It will be useful – even necessary – to work with a modest simplification of this variance term. Useful because the implementation of the single optimization algorithm will be (somewhat) streamlined, at almost no cost in accuracy. Necessary because the ultimate goal – the development of an alternative, but equivalent, two-step algorithm (OCBA+OSTRE) – will require it.

This simplification follows the approach adopted in the derivation of OSTRE. The OSTRE simplification is justified by noting that, typically, $n_{ij} >> \left(1 - p_{ij}\right) / p_{ij}$. I seek an equivalent simplification in Single Optimization. The optimality equations of Single Optimization are considerably more complicated than those for OSTRE. They contain no obvious analogue to $n_{ij} >> \left(1 - p_{ij}\right) / p_{ij}$. But the OSTRE simplification can also be generated by truncating the expansion of the OSTRE variance. (See Appendix 4.) That same truncation can be applied to Single Optimization, obtaining:

$$\sigma_i^2 \approx p_i^2 \left[\frac{1 - p_{i1}}{p_{i1} n_{i1}} + \cdots + \frac{1 - p_{iL}}{p_{iL} n_{iL}} \right].$$

The derivation of single optimization is based on this truncation.

The standard variance (without the truncation) converges to the truncated variance when all n_{ij} terms become arbitrarily large. The notion of <u>all</u> the terms increasing is commonly captured by assuming they increase in a fixed ratio, in which case $n_{11} \to \infty \Rightarrow all\ n_{ij} \to \infty$ at the same rate. For a given design *i* let the n_{ij} increase in

some fixed ratio to n_{i1}:

$$n_{ij} = c_{ij} n_{i1}, \quad j > 1, \quad \text{for constants } c_{ij}.$$

With fixed ratios the standard variance can be expressed solely in terms of constants and n_{i1}. To ease notation let n denote n_{i1}, for any design *i*. By expanding the standard variance, under the assumption of fixed ratios, it can be expressed as:

$$\sigma^2(\text{standard}) = \frac{k_1}{n} + \left[\frac{k_2}{n^2} + \frac{k_3}{n^3} + \cdots + \frac{k_q}{n^q} \right]$$

where the k_i are constants and q is the last term in the sum of higher order terms.

$$\sigma^2(\text{standard}) = \frac{k_1}{n} + \textit{Higher Order Terms}$$
$$\sigma^2(\text{truncated}) = \frac{k_1}{n}$$

The *Higher Order Terms* can be made arbitrarily small by making *n* arbitrarily large, so $\sigma^2(\text{standard}) \to \sigma^2(\text{truncated})$ *as* $n \to \infty$, *or* $T \to \infty$, where *T* is the computing budget for the design in question.

Convergence is nice but we simulate in finite time. I examine the behavior of the standard and truncated variances over finite ranges of *n*. Define:

$e_n =$ Error after n runs = standard variance - truncated variance after n runs.
$=$ *Higher Order Terms* at *n*

$$\text{Relative Variance Error} = \frac{|e_n|}{\text{standard variance}}$$

The following graphs plot the relative error for design 1 of the test model, over n_{11} = 100 to 10,000, and from 10,000 to 100,000. The OSTRE ratios between n_{11} and runs at the other levels are imposed, so the variances can be calculated, and plotted, solely as functions of n_{11}. The values of the level probabilities are shown on the graph.

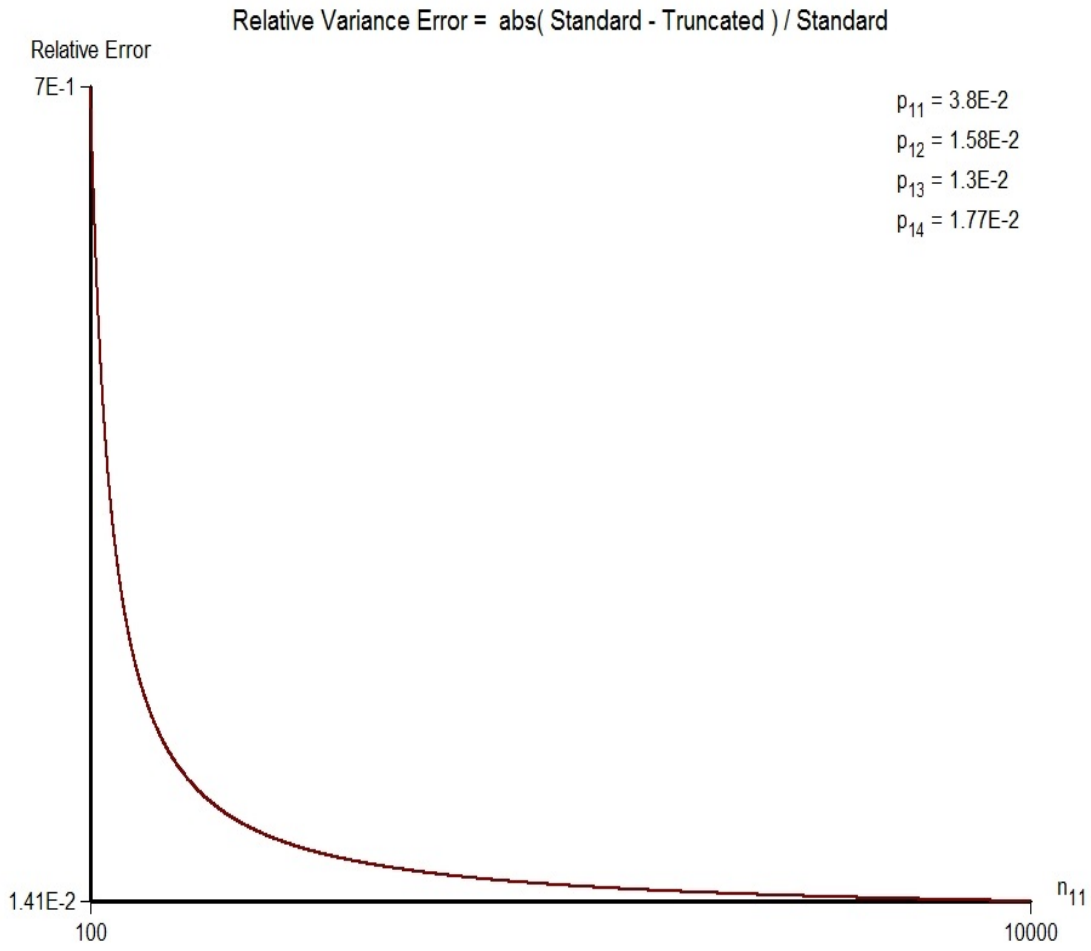

Figure 23. Relative Variance Error: 10,000 Runs

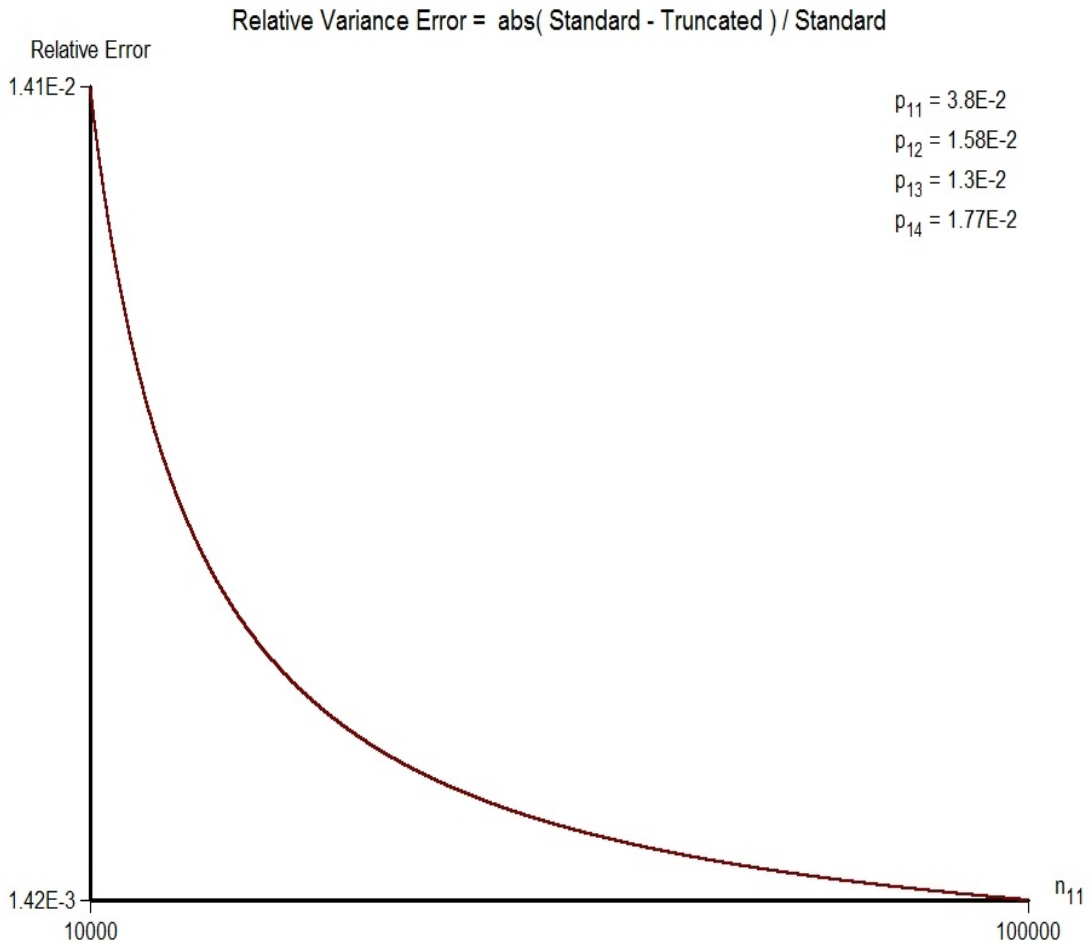

Figure 24. Relative Variance Error: 100,000 Runs

A relative error of about 0.0014 is attained at 100,000 runs in level 1 of design 1. If design 1 is given a computing budget of about 3 minutes, OSTRE would generate about 8.5 million runs in level 1. If the runs in the levels are fixed at the OSTRE ratios, the relative error would be about 1.7E-5. A relative error this small certainly justifies the use of the truncated variance, in both OSTRE and Single Optimization. (And, in the next chapter, in OCBA+OSTRE).

The rate of convergence, r, and the rate constant, C, are defined (if they exist) by

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^r} = C.$$

Over finite ranges of n both r and C may vary with n. Write a possible relationship over such ranges as:

$$e_{n+1} \approx C e_n^r$$
$$\log(e_{n+1}) \approx \log(C) + r \log(e_{n+1})$$

Values for r and C can be estimated empirically by plotting $\log(e_{n+1})$ against $\log(e_n)$. The log values are negative because the error terms are less than 1. Decreasing log values are associated with an increasing n_{11}:

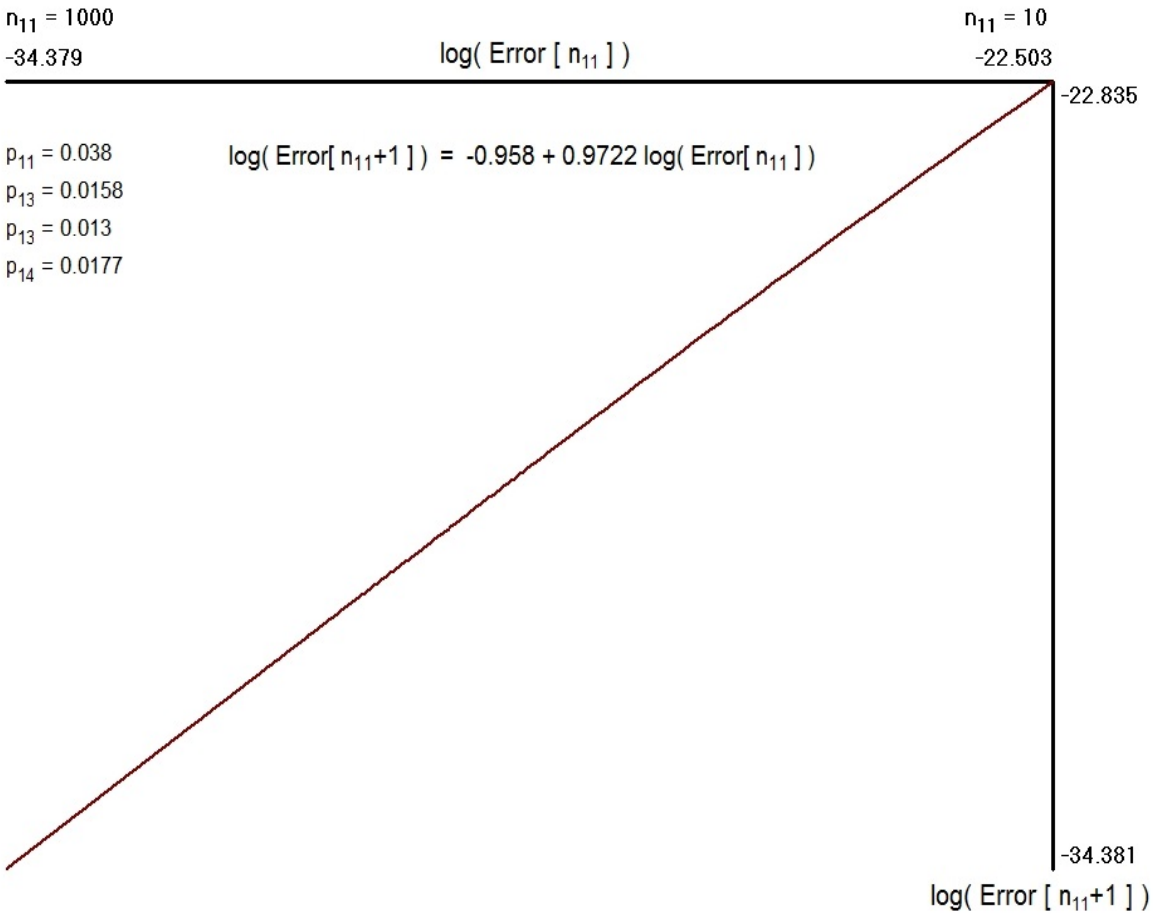

Figure 25. Log[Error]: 10 to 1000

Over this range the convergence rate, r, is close to 1 (linear convergence). The rate constant, C, is about 0.4. For much larger values of n_{11} the convergence rate is clearly 1, and C comes closer and closer to 1:

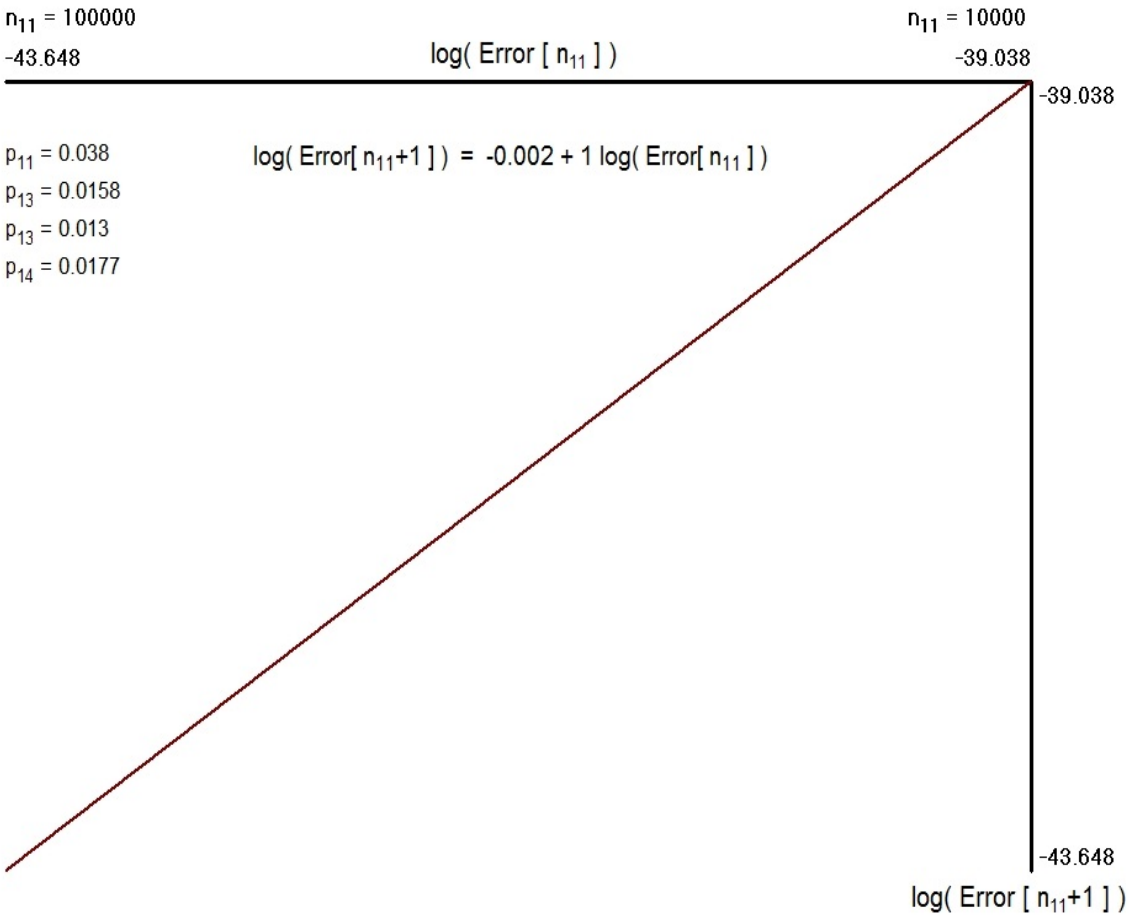

Figure 26. Log[Error]: 10,000 to 100,000

If $r = 1$ and $C = 1$ exactly additional runs would not reduce the error. We know that is never true because the error term converges to 0. But progress toward 0 slows as the number of runs increases. Nonetheless, as seen in the results on the relative error, the number of runs required to make the error term negligible is much smaller than the number of runs generated in the test model, and generally in typical applications, over modest computing budgets.

6.4 Optimality Equations

Under the normality assumption:

$$\hat{p}_1 - \hat{p}_i \sim N\left[p_1 - p_i ,\ \sigma_1^2 + \sigma_i^2 \right], \quad i = 2,...,D ,$$

$$P\left[\left(\hat{p}_1 - \hat{p}_i > 0 \right) \right] = P\left[\frac{\hat{p}_1 - \hat{p}_i - p_1 + p_i}{\sqrt{\sigma_1^2 + \sigma_i^2}} > \frac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}} \right]$$

$$= P\left[Z > \frac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}} \right], \quad Z \sim N\left[0,1\right]$$

Define functions ϕ_i, whose arguments are

$$\phi_i \equiv \phi_i\left(n_{11},...,n_{1L} , n_{i1},...,n_{iL} \right), \quad i = 2,...,D \quad j = 1,...,L ,$$

as $$\phi_i \equiv \frac{p_i - p_1}{\sqrt{\sigma_1^2\left(n_{11},...,n_{1L} \right) + \sigma_i^2\left(n_{i1},...,n_{iL} \right)}} \qquad i = 2,...,D$$

where $$\sigma_i^2 = p_i^2\left[\frac{1 - p_{i1}}{p_{i1} n_{i1}} + \cdots + \frac{1 - p_{iL}}{p_{iL} n_{iL}} \right], \quad i = 1,...,D$$

For notational simplicity these functions will be referenced without listing their arguments. It should be noted that they are indexed $i = 2,\ldots, D$, there being no ϕ_1, and that the runs in level 1, $n_{11},...,n_{1L}$, are arguments of every ϕ_i, whereas the runs in design i

are arguments solely of the corresponding ϕ_i. These properties follow from the designation of design 1 as the best design.

Thus we have:

$$P\left[\left(\hat{p}_1 - \hat{p}_i > 0\right)\right] = \frac{1}{2\pi}\int_{\phi_i}^{\infty} e^{-\frac{x^2}{2}} dx$$

and the optimization becomes

$$\underset{n_{ij}}{Min} \sum_{i=2}^{D} \int_{\phi_i}^{\infty} e^{-\frac{x^2}{2}} dx \qquad \text{s/t} \quad \sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij}\, n_{ij} = T\,.$$

The derivatives of the ϕ_i functions will be needed. They require the derivatives of the variances, which are:

$$\frac{\partial \sigma_i^2}{\partial n_{ij}} = \frac{-p_i^2\left(1-p_{ij}\right)}{p_{ij}\, n_{ij}^2}$$

The ϕ_i derivatives are then:

$$\frac{\partial \phi_i}{\partial n_{mj}} = \frac{\left(p_i - p_1\right) p_m^2 \left(1-p_{mj}\right)}{2\, p_{mj}\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}} n_{mj}^2} \qquad i = 2,...,D \quad j = 1,...L \quad m = 1 \; or \; i$$

The Lagrangian: $L\left(n_{1j}, n_{ij}\right) = \frac{1}{\sqrt{2\pi}} \sum_{i=2}^{D} \int_{\phi_i\left(n_{1j}, n_{ij}\right)}^{\infty} e^{-\frac{x^2}{2}} dx + \lambda\left[\sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij} n_{ij} - T\right]$

$$\frac{\partial L}{\partial n_{1j}} = -\frac{1}{\sqrt{2\pi}}\sum_{i=2}^{D} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{1j}} + \lambda b_{1j} = 0 \qquad j = 1,...,L$$

$$\lambda = \frac{1}{b_{1j}\sqrt{2\pi}}\sum_{i=2}^{D} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{1j}}$$

$$\frac{\partial L}{\partial n_{ij}} = -\frac{1}{\sqrt{2\pi}} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{ij}} + \lambda b_{ij} = 0 \qquad i = 2,...D, \quad j = 1,...,L$$

$$\lambda = \frac{1}{b_{ij}\sqrt{2\pi}} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{ij}}$$

$$\frac{\partial L}{\partial \lambda} = \sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij} n_{ij} - T = 0$$

Equating the $\lambda's$ yields $DL-1$ equations which, together with the budget constraint, constitute *DL* equations for solving *DL* variables. The system may be difficult to decipher in the general case of *D* designs, *L* levels. Appendix 7 contains a complete statement of the equations for the case of 4 designs, 3 levels. The general solution is:

$$\frac{1}{b_{1k}}\sum_{i=2}^{D} e^{\frac{-\phi_i^2}{2}}\frac{\partial\phi_i}{\partial n_{1k}} = \frac{1}{b_{1,k+1}}\sum_{i=2}^{D} e^{\frac{-\phi_i^2}{2}}\frac{\partial\phi_i}{\partial n_{1,k+1}} \qquad k=1,\ldots,L-1$$

$$\frac{1}{b_{1L}}\sum_{i=2}^{D} e^{\frac{-\phi_i^2}{2}}\frac{\partial\phi_i}{\partial n_{1L}} = \frac{1}{b_{21}} e^{\frac{-\phi_2^2}{2}}\frac{\partial\phi_2}{\partial n_{2,1}}$$

$$\frac{1}{b_{2k}} e^{\frac{-\phi_2^2}{2}}\frac{\partial\phi_2}{\partial n_{2k}} = \frac{1}{b_{2,k+1}} e^{\frac{-\phi_2^2}{2}}\frac{\partial\phi_2}{\partial n_{2,k+1}} \qquad k=1,\ldots,L-1$$

$$\frac{1}{b_{2L}} e^{\frac{-\phi_2^2}{2}}\frac{\partial\phi_2}{\partial n_{2L}} = \frac{1}{b_{31}} e^{\frac{-\phi_3^2}{2}}\frac{\partial\phi_3}{\partial n_{31}}$$

.

.

$$\frac{1}{b_{D-1,k}} e^{\frac{-\phi_{D-1}^2}{2}}\frac{\partial\phi_{D-1}}{\partial n_{D-1,k}} = \frac{1}{b_{D-1,k+1}} e^{\frac{-\phi_{D-1}^2}{2}}\frac{\partial\phi_{D-1}}{\partial n_{D-1,k+1}} \qquad k=1,\ldots,L-1$$

$$\frac{1}{b_{D-1,L}} e^{\frac{-\phi_{D-1}^2}{2}}\frac{\partial\phi_{D-1}}{\partial n_{D-1,L}} = \frac{1}{b_{D1}} e^{\frac{-\phi_D^2}{2}}\frac{\partial\phi_D}{\partial n_{D1}}$$

$$\frac{1}{b_{Dk}} e^{\frac{-\phi_D^2}{2}}\frac{\partial\phi_D}{\partial n_{Dk}} = \frac{1}{b_{D,k+1}} e^{\frac{-\phi_D^2}{2}}\frac{\partial\phi_D}{\partial n_{D,k+1}} \qquad k=1,\ldots,L-1$$

The term $1/\sqrt{2\pi}$ appears in each derivative of the Lagrangian, so it cancels out of these equations. Stated in the form above, some of the exponential terms could be cancelled from both sides of some equations. But this form is arbitrary. At the optimum every term, on the LHS or RHS of these equations, should equal every other term. The order in which they are equated to each other is arbitrary, and for some such comparisons the exponential terms cannot be cancelled. They are retained here for complete generality, and will be needed in the implementation of the algorithm.

6.5 Implementing Single Optimization

Single Optimization implementation is very similar to OCBA implementation, with the major exception that selecting the level, at each update, for the next allocation is more complicated. I present a fairly complete description of the algorithm.

Notation

T = total time for the algorithm (the budget constraint)
AccumulatedTime = time consumed by the algorithm at any point prior to termination
D = number of designs
L = number of levels (the same for each design).
$p[i]$ = probability of design *i*; corresponds to p_i in the theory.
$pL[..]$ = double array of level probabilities; $pL[i, j]$ = probability of design *i*, level *j*; corresponds to p_{ij} in the theory.
$nL[i, j]$ = double array of number of runs in design *i*, level *j*; corresponds to n_{ij} in the theory.
$hL[i, j]$ = double array of number of hits in design *i*, level *j*.
$timeL[i, j]$ = double array of time consumed doing simulations in design *i*, level *j*
$bL[i, j]$ = double array of average time-per-run, design *i*, level *j*; corresponds to b_{ij} in the theory.
ES = Array of Entrance States; a multiple array, indexed for design, level, number of states, and number of parameters required to characterize an entrance state. Test model example:

$ES[i, j, 3, 1]$ = 25 means that in design *i*, level *j*, the first parameter of the third entrance state 3 is 25.
$ES[i, j, 3, 2]$ = 5 means that in design *i*, level *j*, the second parameter of the third entrance state is 5.

T_0 = Stopping time for simulations at each level in the initialization phase.
H_0 = Required number of hits at each level in the initialization phase.

Initialization Phase

The initialization phase is equivalent to standard fixed effort splitting with equal allocation to all levels, but within a time constraint much less than the total computing budget. Initial estimates for the parameters must be generated. Allocate an equal amount of time to each level, across all designs, and conduct simple MC runs within each level. A stopping time is required, for each level: T_0. Or perhaps a required number of runs: H_0. (Or perhaps both). The optimization commences with the initial estimates, so they should be as accurate as possible. That suggests allocating a "large" T_0, or requiring a "large" H_0. But the optimization part of the algorithm also requires a "large" portion of the total time, in order for the allocations to approach their optima. This suggests limiting the initialization phase. It might be possible to establish an optimum split of the total time between the initialization and optimization phases, but that is not part of this research.

Call the initialization procedure:

$$\text{INITIALIZE}\left(T_0, H_0, D, L, pL[i,j], nL[i,j], hL[i,j], bL[i,j], timeL[i,j], ES[i,j,...]\right)$$

Constant inputs: T_0, H_0, D, L

Variable inputs: $pL[i,j]$, $nL[i,j]$, $hL[i,j]$, $bL[i,j]$, $timeL[i,j]$, ES

The operational part of the algorithm is the simulation of the stochastic process. That is unique to each simulation problem, so no description is presented here. The level

probability estimates are the numbers of hits obtained (within the time, or the runs, allocated to that level) divided by the number of runs:

$$pL[i, j] = \frac{h[i, j]}{n[i, j]}$$

Common notation denotes estimates with a "hat", such as $\hat{p}L$, but I write them plain, as they appear in the code.

Average time-per-run is estimated, for each set of runs within a level, by

$$b[i, j] = \frac{timeL[i, j]}{n[i, j]}$$

Every hit, at every level (except the final one), generates a "new" entrance state from which runs at the next level can be launched. (Each such entrance state is a new addition to the set of entrance states at that level, though it may be, and often is, a copy of an entrance state already in the set.) The entrance state data, for all designs and levels, are retained in the array ES. These entrance state sets will be expanded by new hits in the optimization phase.

Optimization Phase

The parameters of the optimality equations are, in the theoretical derivation, the true parameters (the true p_i, p_{ij}, b_{ij}). The algorithm must necessarily utilize estimates of those parameters. The initialization phase generates initial estimates. The estimates for the design probabilities are constructed from the level probability estimates:

$$p[i] = \prod_{j=1}^{L} pL[i, j]$$

The optimization phase consists of a series of simulation updates. At each update new estimates for all the parameters are calculated. Current values of the runs, for each design and level, are available. These are plugged into the optimality equations, which should be satisfied at the optima. But they will not, in general, be satisfied at any update. The purpose of each update is to nudge the equations toward satisfaction by allocating a new set of runs to one of the levels in a way that reduces these inequalities. Δ designates the number of runs in the new allocation. It is typically small, relative to the number of runs the overall simulation will eventually generate. The strategy is to try to allocate Δ, at each update, to the design and level which will make the largest contribution to reducing the inequalities. The method by which this allocation is chosen is a very important aspect of the algorithm.

In principle the optimization problem, at each update, could be solved by numerical techniques. The next allocation could then be made to the n_{ij} which differs most from its optimum. In practice, however, the optimization problem may be quite large and intractable. The problem would have to be solved at each update, and there will be many updates. A simpler approach, not so computationally demanding, is required.

It is not obvious, from inspection of the terms of the optimality equations, how the allocation should be made. Nor is there a single analytical approach to making the allocation, as there are several plausible measures of the inequality to be reduced. One might, for example, seek the allocation which most reduces the sum of all the inequalities, across all combinations of the terms. Or one could seek the allocation which

most reduces the largest inequality in the system of optimality equations. But the ordering of the terms in this system is arbitrary.

I adopt a heuristically appealing, non-arbitrary method which entails relatively modest overhead. For *D* designs, *L* levels, there will be *DL* terms in the optimality equations (ignoring the budget constraint). Designate them as $t_1, t_2, ..., t_{DL}$. Optimality requires $t_1 = t_2 = ... = t_{DL}$. At each update the value of each of these terms is calculated. Each of these terms converges asymptotically to 0 as the PCS converges to 1, which it will if all $n_{ij} \to \infty$ (see Appendix 5 for proof). A simple, seemingly plausible allocation rule would be: Select the largest term, calculate its new value under every possible Δ allocation (the allocation of Δ runs to every possible level) , and select the allocation which reduces it the most. If the largest term is continuously reduced all terms must, in the limit, be pushed to 0, and thus to equality.

In practice this is not the best strategy. For any finite budget the terms, at optimum, are not 0. Increasing a single n_{ij}, as is done at each allocation, can reduce the largest term, but its impact on other terms is uncertain.

In practice it is better to reduce the maximum difference, *MaxDiff* , among all terms. Select the maximum and minimum among them and calculate *MaxDiff* = $t_{\max} - t_{\min}$. Increase, by Δ, each n_{ij} that appears in either term, and calculate the resulting change in *MaxDiff*. Make the allocation to the n_{ij} that generates the largest reduction in *MaxDiff*.

In practice the decline in *MaxDiff* will not be strictly monotonic. Increasing a single n_{ij} may not decrease every optimality term, or decrease every inequality between them. It is possible that, after any update, the *MaxDiff* at the next update (which will, in general, contain different terms from the prior *MaxDiff*) is larger than the prior *MaxDiff*. At each update the (estimated) parameters comprising the terms are random, meaning each *MaxDiff* has some randomness, meaning a new *MaxDiff* could be larger than the prior *MaxDiff* even if the prior Δ is optimally allocated. And in practice there is the noxious problem of false leads. If, at any update, the algorithm generates a false lead, the optimality terms for the next update are structurally different from the last update. The expected continuity between the prior and the new *MaxDiff* could be impaired.

Despite some inevitable unevenness in the decline of *MaxDiff*, I show that, in practice, on my test model, it does decline rapidly, over relatively short time periods, thereby significantly reducing the inequalities in the optimality equations.

The following graphs show the decline of *MaxDiff*, for Single Optimization on my test model, over 8 designs, 4 levels. The budget time was 480 sec (1 minute for each design). At the first update, immediately after the initialization phase, *MaxDiff* was about 1.6E-3. After 3000 updates (20 observations, recorded every 150 updates) it was about 3E-7. An initial unevenness in the decline is evident:

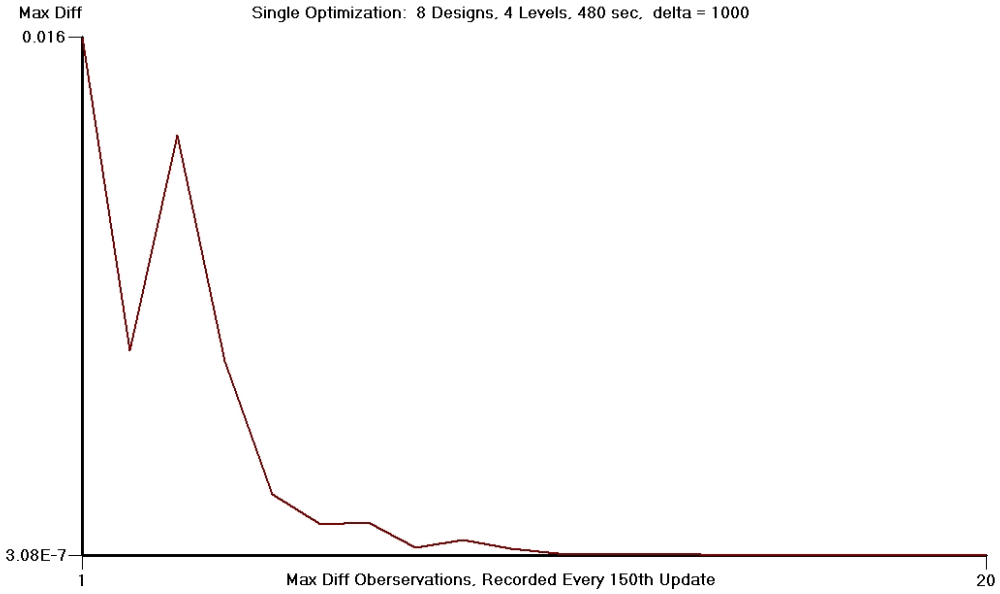

Figure 27. *MaxDiff*: 1 to 20 Updates

Extending the algorithm out to 15,000 updates takes *MaxDiff* down to about 7.5E-8:

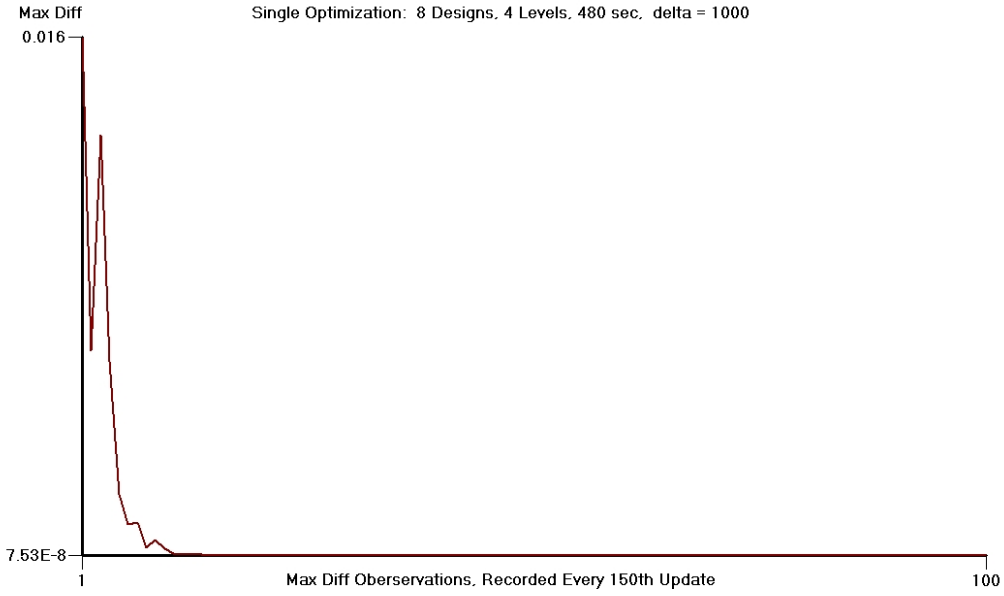

Fig 28. *MaxDiff*: 1 to 100 Updates

When the initialization phase concludes, the Accumulated Time stands at:

$$AccumulatedTime = DLT_0$$

Then the optimizing updates commence.

repeat
// constant inputs: D, L, Δ
// variable inputs: *AccumulatedTime*, *p*, *pL*, *nL*, *hL*, *timeL*, *bL*, *ES*
UPDATE$(D, L, \Delta,$ *AccumulatedTime*, *p*, *pL*, *nL*, *hL*, *timeL*, *bL*, *ES*$)$;
until $AccumulatedTime \geq T$;

When *AccumulatedTime* reaches *T*, the total budget, the algorithm stops, thereby automatically enforcing the budget constraint.

The UPDATE procedure can be rather involved. Its key steps are:

(1) Determine, by a strategy such as the one suggested above, which level, in which design, should get the next Δ allocation.

(2) Do Δ simulation runs in the selected level. Each run is launched from a randomly selected entrance state from the appropriate ES[*i*,*j*,…]. Each run that generates a hit on the next level also generates a new entrance state for storage in the next level of the appropriate ES[*i*,*j*,…], except for the final level.

(3) Update the relevant estimated parameters. These are estimated, at each update, for the entire simulation up to that point, meaning new hits, runs, and time are added to old hits, runs, time:

$$h[i,j] = h[i,j] + \text{ New Hits, if any;}$$
$$n[i,j] = n[i,j] + \Delta;$$
$$time[i,j] = time[i,j] + \text{ time consumed by this update}$$
$$b[i,j] = \frac{time[i,j]}{n[i,j]};$$
$$pL[i,j] = \frac{h[i,j]}{n[i,j]};$$
$$p[i] = \prod_{j=1}^{L} pL[i,j];$$

False Leads

As with OCBA, Single Optimization can suffer from false leads. Any algorithm optimizing allocations among designs that must designate the best design to play, in theory, a role structurally different from the other designs will suffer from false leads. This applies to OCBA, to Single Optimization, and it will apply to the algorithm of the next chapter, OCBA+OSTRE. It does not apply to Monte Carlo or standard splitting because they do not optimize anything. Nor does it apply to OSTRE, because OSTRE optimizes solely among levels within designs, not across designs.

False leads arise in Single Optimization because, in optimizing across all levels of all designs, the levels of the best design play a different role in Single Optimization theory. All levels of the best design appear as variables in every optimality term. Levels of other designs appear only in the optimality terms corresponding to their particular designs. If, at each update, the algorithm estimates a different design to be best (different from its prior estimate), it must code the update to replace the old estimated best with the new estimated best. (Depending on the coding scheme this could entail swapping all data

between the data arrays for the old and new best.) As with OCBA the strength of Single Optimization must lie in its ability to recover from false leads.

6.6 Computational Complexity

I offer a very basic complexity analysis of the Single Optimization algorithm, focusing solely on how computational effort scales with the variables that define problem size. For the Single Optimization algorithm problem size is measured by the number of designs *D*, and the number of levels per design, *L*. At issue is not the actual computational work done for a given *D* and *L*, but how that work increases as a function of *D* and *L*. Big *O* notation is used to capture this. Let *X* represent the computational effort required by the algorithm (in part or whole). Then:

$$X \sim O\left[f\left(D, L; c\right)\right]$$

means there is a constant c such that the function $f\left(D, L; c\right)$ is an upper bound on the computational effort (however measured) of *X*, for all *D* and *L*. There is no claim that it is a tight upper bound. The constant c can be any number for which the bound holds and the effort scales only with *D* and *L*. As the analysis expands to incorporate more parts of the algorithm c is understood to expand to some constant sufficient to enforce the bound, for any *D* and *L*.

This analysis is based on several assumptions and simplifications. Pure record keeping, including the maintenance, in memory, of the entrance state sets, is ignored. It is assumed that the computational effort required to set up a run – such as selecting the entrance state from which to launch it -- is the same for all levels. The work required to

do the run – the actual simulation of one sample path of the stochastic process – is also ignored. In most cases this "simulation work" consumes the bulk of the real computational time. But it depends on the type of process being simulated, which can vary considerably from case to case, and cannot be captured in this analysis.

The algorithm divides into three parts: the initialization phase, the optimality equations phase, the updating phase. Total computational effort is the sum of the effort required for these parts.

The initialization phase is straightforward, as there is no optimization during initialization, and no computation beyond record keeping. Thus:

$$\textit{Initialization} \sim O\left[c_1 DL\right]$$

Constructing the terms in the optimality equations requires considerable calculation. I do not count the actual computations – the flops, or floating point operations – since they do not scale with D or L. The standard term of the optimality equations takes the form:

$$e^{\frac{-\phi_i^2}{2}} \frac{\partial \phi_i}{\partial n_{mj}} \qquad m = 1 \text{ or } i, \quad i = 2,...,D, \quad j = 1,..,L$$

The actual calculation of this term requires several flops. But they are the same, for a given m and j. I consider only how this entire calculation scales with D and L. From the complete set of optimality equations it is clear that:

For $m = 1$, this term is calculated $(D\text{-}1)$ times for $j = 1,\ldots,L$: $(D\text{-}1)L$ calculations.

For $m = i$, it is calculated $i = 2,\ldots,D$, and, for $j = 1,\ldots,L$: $(D\text{-}1)L$ calculations.

In total the effort required is $2(D\text{-}1)L$. But the 2 doesn't scale, so:

$$Optimality\,Terms \sim O\left[c_2 (D-1) L\right]$$

The updating phase, employing the *MaxDiff* method for selecting the next allocation, requires (1) finding the maximum and minimum of the optimality terms, (2) re-computing them for every possible allocation, and (3) selecting that allocation which most reduces *MaxDiff*. There are *DL* optimality terms. Steps (1) and (3) require *DL* comparisons each, for an effort of 2*DL*. Step (2) requires re-computing the max and min terms for each of the possible allocations, of which there are *DL*, so it also requires an effort of 2*DL*. Total effort is thus:

$$Updating \sim O\left[c_3 DL\right]$$

Combining all the phases:

Single Optimization Effort = Initialization + Optimality Terms + Updating

Single Optimization Effort $\sim O\left[c_1 DL\right] + O\left[c_2 (D-1) L\right] + O\left[c_3 DL\right]$

6.7 Single Optimization vs Splitting

The first test pits Single Optimization against Splitting. Single Optimization should win. It optimizes across all levels of all designs. Splitting optimizes nothing. But Single Optimization is built on assumptions that, in practice, could be shaky. The results:

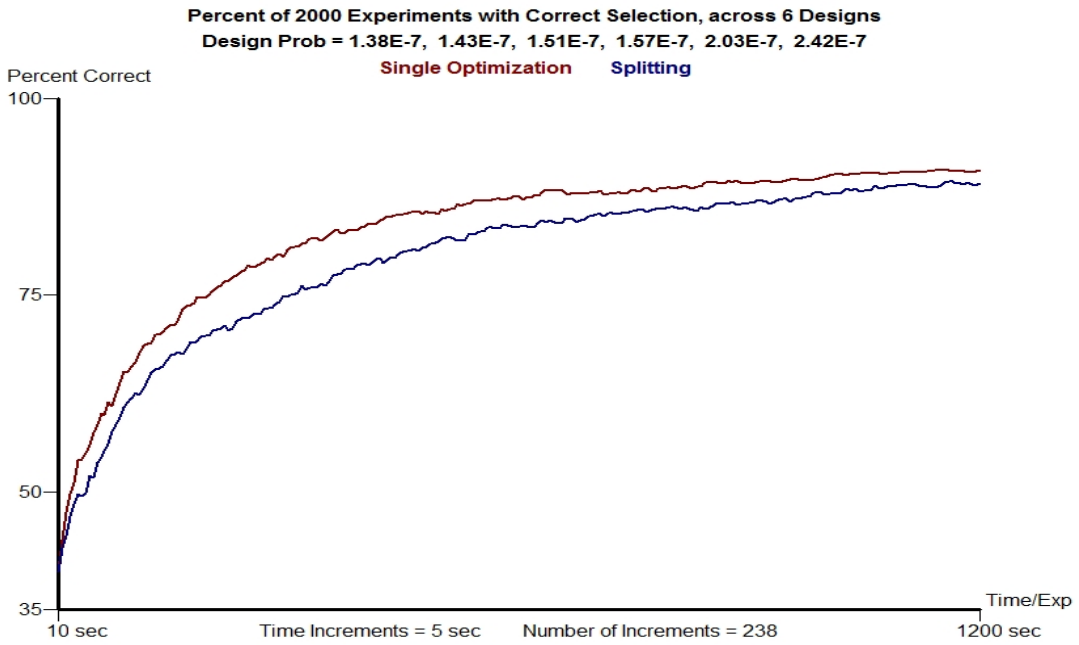

Figure 29. Single Optimization vs Splitting, 6 Designs

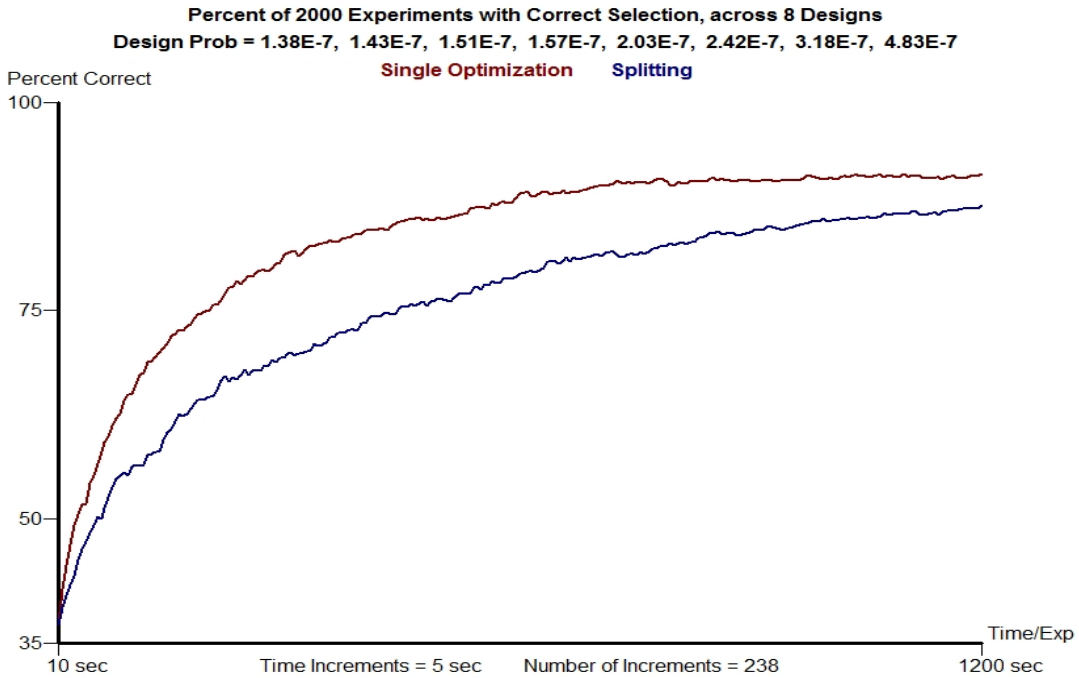

Figure 30. Single Optimization vs Splitting, 8 Designs

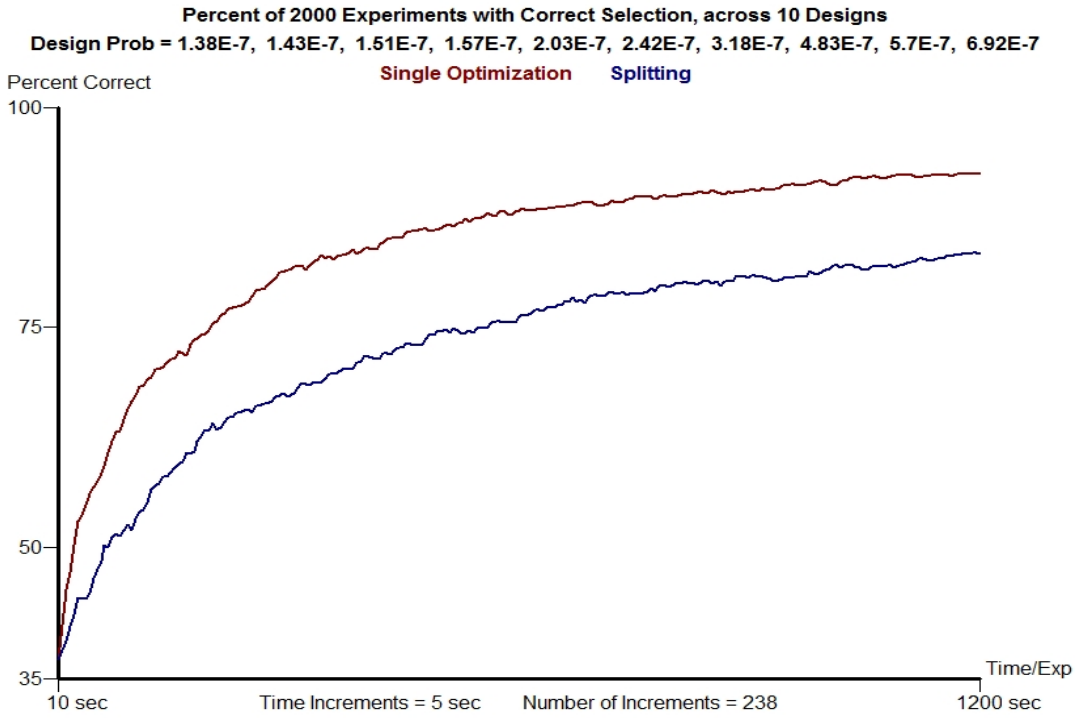

Figure 31. Single Optimization vs Splitting, 10 Designs

Single Optimization wins. For 6 designs it reaches 0.8 PCS after 295 sec., splitting after 500 sec. For 8 designs Single Optimization reaches 0.8 PCS at 285 sec. splitting at 675. For 10 designs those times are 295 and 810 sec.. Splitting's performance deteriorates, as expected, as the number of designs increases: 500, 675, 810 sec., for designs 6, 8, 10. Single Optimization should perform approximately the same as the number of designs increase. Its times for reaching 0.8 PCS are 295, 285, 295.

I compare allocations made by splitting and Single Optimization. The comparison is between allocations (of time, and the resulting number of runs) for level 1, of designs 1 through 10. Allocations to other levels follow a similar pattern. The data

come from the test in Figure 30 above. Note how Single Optimization devotes almost no time to designs 4-10, all of which it quickly deems uncompetitive.

Table 3. Splitting Allocations vs Single Optimization Allocations

	Splitting, Level 1			Single Optimization, Level 1	
Designs	Time(sec)	Runs		Time(sec)	Runs
1	30	8,285,279		25.8	7,115,442
2	30	8,267,280		80.3	22,136,790
3	30	8,240,639		22	6,032,476
4	30	8,222,640		2.3	629,045
5	30	8,083,439		0.3	79,964
6	30	7,992,719		0.25	66,814
7	30	7,871,040		0.25	65,593
8	30	7,650,719		0.25	64,002
9	30	7,586,399		0.25	63,646
10	30	7,486,560		0.27	66,278

6.8 Single Optimization vs OSTRE

Single Optimization's real competitor is not splitting, which optimizes nothing, but OSTRE, which optimizes across levels but not across designs. Single Optimization's potential advantage over OSTRE lies in the fact that it optimizes across all levels in all designs at a single stroke. The total effort it devotes to relatively uncompetitive designs (the effort across all the levels of those designs) will be (considerably) less than the effort

OSTRE devotes to uncompetitive designs, which is, by default, exactly the effort it devotes to competitive designs. Single Optimization will, therefore, devote more total effort to competitive designs than does OSTRE, giving Single Optimization an advantage where it most counts. Single Optimization and OSTRE share the same major assumption (the OSTRE assumption), so neither has a leg up on that count. OSTRE does have an advantage in implementation. Within each design it easily computes, at each update, the exact solution to the optimization problem, which facilitates the task of choosing where the next allocation should go. This advantage lies not just in speed of computation, which can be considerable when there are many designs, but also in accuracy, in the sense that, with an exact solution at hand, at each update, OSTRE's choice of where to send the next allocation of effort could be somewhat better than Single Optimization's choice. Better in the sense of moving the equality conditions toward equality more decisively. Single Optimization relies on the clumsier method of *MaxDiff*, which should work just as well asymptotically, but probably not as well over modest computing budgets.

Over a small number of relatively competitive designs there should be little difference between Single Optimization and OSTRE. But the advantage should shift to Single Optimization as the number of uncompetitive designs increases. This pattern holds, on the test model, for comparisons over 6, 8, and 10 designs:

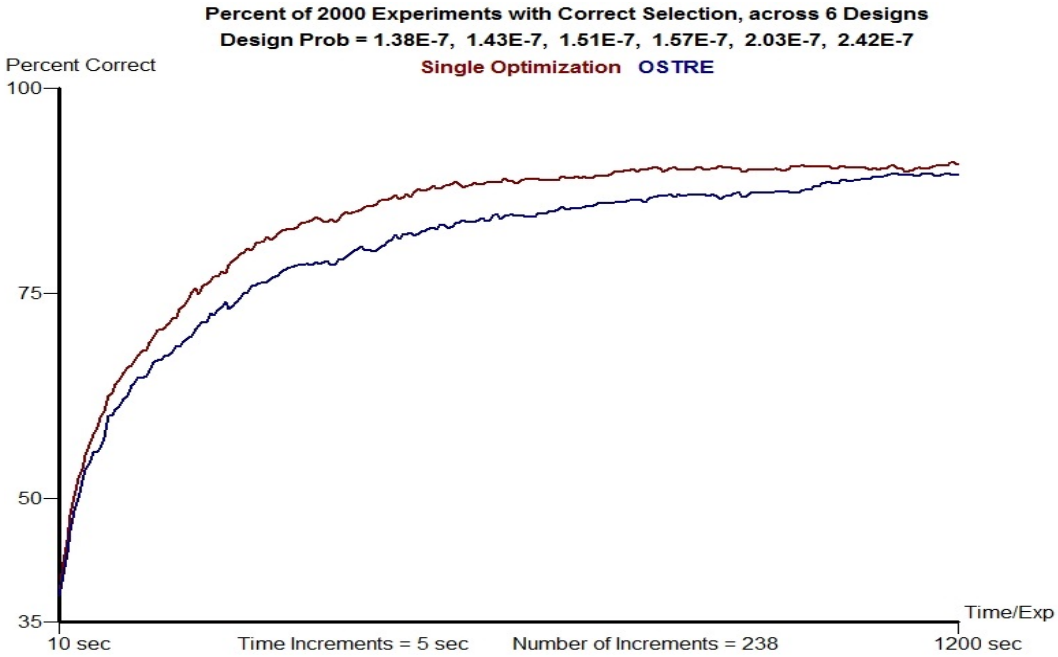

Figure 32. Single Optimization vs OSTRE, 6 Designs

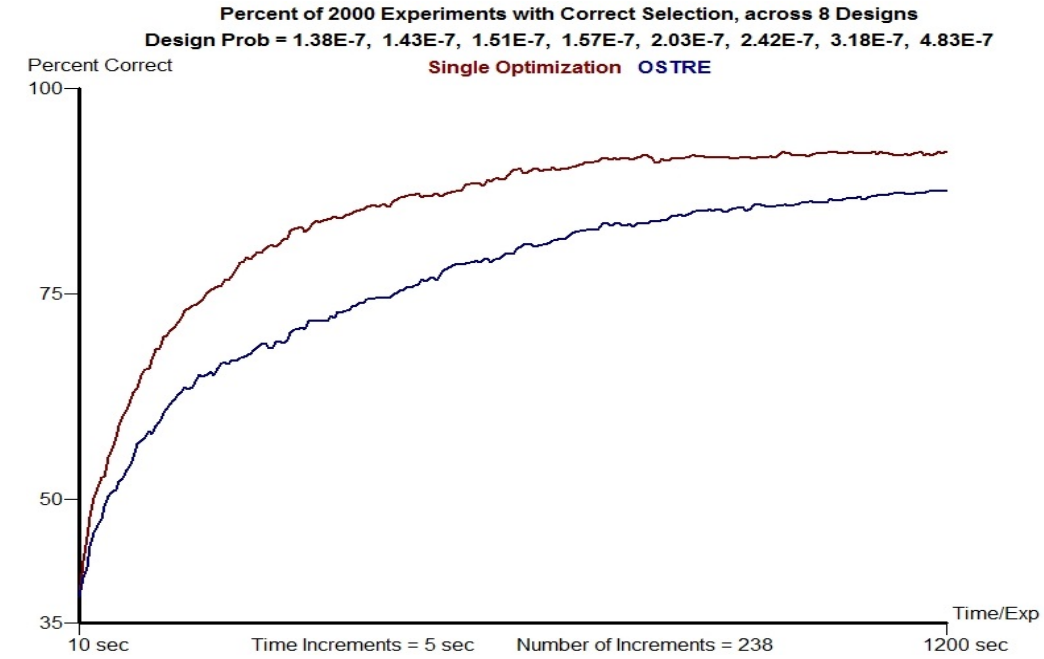

Figure 33. Single Optimization vs OSTRE, 8 Designs

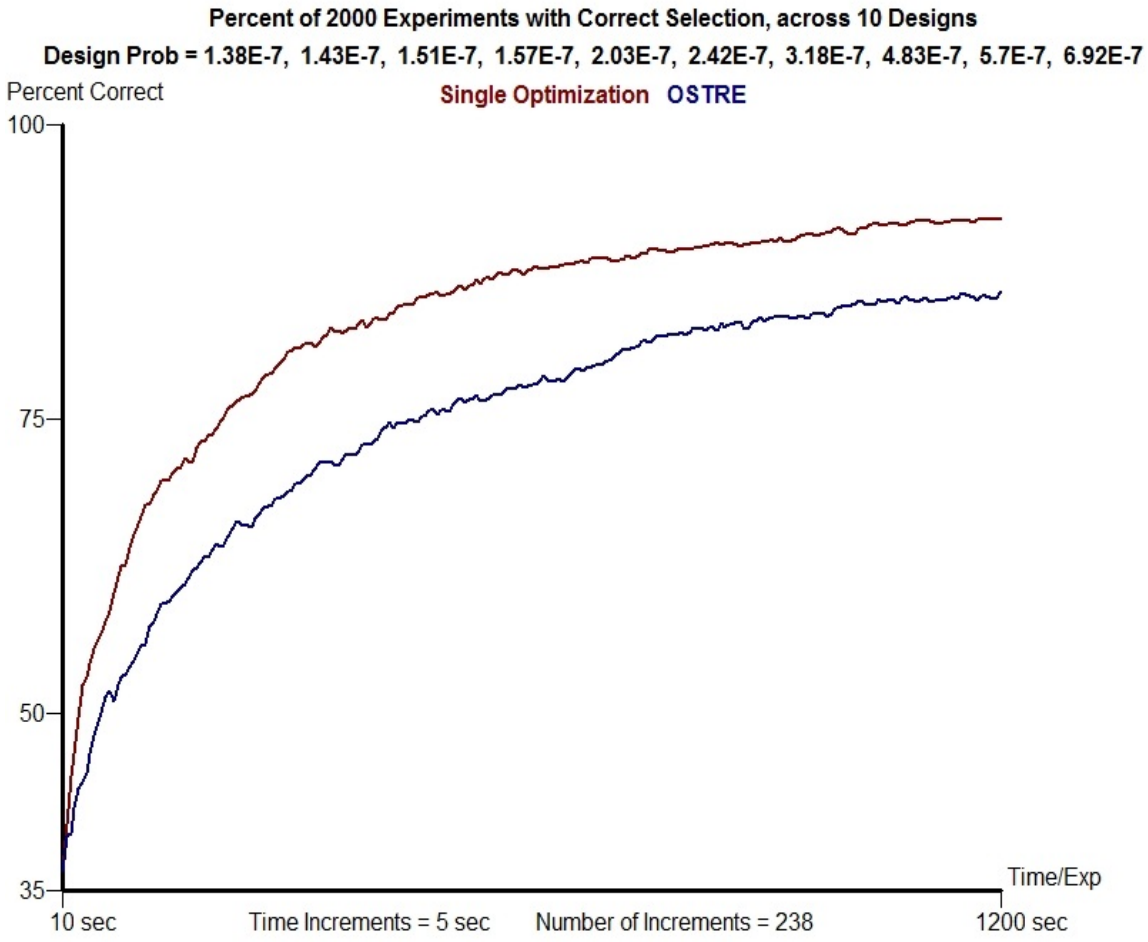

Figure 34. Single Optimzation vs OSTRE, 10 Designs

7. OCBA+OSTRE

Single Optimization is, theoretically, the best approach (subject to its assumptions and approximations) to maximizing the Probability of Correct Selection. But Single Optimization is unwieldy in implementation, especially as the number of designs and/or levels grows. The complexity of its optimality equations renders it unattractive.

This dissertation proposes, as an alternative to Single Optimization, a two-step algorithm, called OCBA+OSTRE. The goal is to simplify, somewhat, the implementing algorithm. The method is to break the problem into smaller bites. Break it into two parts, or steps, to be carried out sequentially.

The first step, the OCBA step, allocates runs (or time) among designs. It is similar to the original OCBA optimization of the allocation of runs among designs. The next step is to take the runs (or time) allocated to a design – the output of the OCBA step – and allocate them among the levels of that design. This is the OSTRE step, as it implements the OSTRE method for optimizing the allocation of effort among levels of a given design.

If OCBA+OSTRE were mathematically inferior to Single Optimization – in the sense that it fails, theoretically, to deliver as high a PCS as does Single Optimization – it could still be an attractive approach, given its promise of (somewhat) less complexity. In fact, however, OCBA+OSTRE is mathematically equivalent to Single Optimization. It

yields, theoretically, the same solution. Proof of this claim will be the subject of the next chapter. This chapter derives OCBA+OSTRE and discusses its implementation.

The OCBA part of the optimization problem is:

$$Min\ P\left[\hat{p}_1 > \hat{p}_i, ..., \hat{p}_1 > \hat{p}_D\right] \quad \text{s/t} \quad T = \sum_{i=1}^{D} b_i N_i \quad \text{for } D \text{ designs}$$

$T =$ total budget constraint

$b_i =$ the average time of a MC run in design i

$N_i =$ the allocation of runs to design i

Given an allocation, N_i^*, to design *i* (the solution to the OCBA optimization), the OSTRE part of the algorithm determines the allocation of that N_i^* to a particular level within that design. That optimization problem may be stated as

$$Min\ \sigma_i^2 \quad \text{s/t} \quad \sum_{j=1}^{L} b_{ij} n_{ij} = b_i N_i^* \quad \text{for each design } i;\ L = \text{levels}$$

with

σ_i^2 = the variance of the estimator of design i

b_{ij} = the average time of one run at level i, within design j

n_{ij} = the runs allocated to level j, within design i

N_i^* = the solution of the OCBA optimization problem, for design i

This statement of the problem poses a dilemma: b_i, the average time of a run in design *i*, appears in each optimization step. It links the two steps. But what is it? There are no single runs over the entire design, only runs at each level within the design. The N_i^* allocation of runs to design *i* is an alias: it does not designate the number of MC runs that will be allocated to all of design *i*, but rather the number of runs that will be allocated

to one of the levels of design *i*, under the OSTRE step. For the OCBA step to be legitimate the N_i^* must consistent with OCBA and with the subsequent OSTRE step. The b_i must be consistent with an actual MC run over the entire design and a run over just a single level within the design. The unpretentious b_i unites the OCBA and the OSTRE steps. It must do so in a way that allows each step to "do its own thing" – allocate designs or allocate to levels – with no contradiction between them.

In theory the b_i are "just" parameters, which we could assume are given. But in practice they are not given, in the sense that they cannot be directly estimated. The algorithm never does a single MC run across an entire design, so it can never estimate the b_i directly. But the algorithm will require estimates of the b_i. They can only emerge indirectly, from estimates the algorithm can make. The link between them must be provided by theory, in a way that makes everything consistent with everything!

The first step in implementation is the OCBA step, but theory must start with the OSTRE solution. We start by requiring that, within each design, the OSTRE optimality equations are satisfied (in theory). From that requirement we can back out the correct definition of the b_i, and harmoniously unite the OCBA and OSTRE steps.

The OSTRE optimality equations, in the approximated version attained by truncating the variance, are:

$$\frac{n_{i1}^2\, b_{i1}\, p_{i1}}{1-p_{i1}} = \cdots = \frac{n_{iL}^2\, b_{iL}\, p_{iL}}{1-p_{iL}} \qquad L = \text{levels within design } i$$

which can be expressed as:

$$\frac{n_{ij}}{n_{i1}} = \sqrt{\frac{b_{i1}\, p_{i1}\left(1-p_{ij}\right)}{b_{ij}\, p_{ij}\left(1-p_{i1}\right)}} \qquad j = 2,...,L$$

The key to OCBA+OSTRE is to define the notional N_i so that it reflects, or is consistent with, these OSTRE ratios. Define constants α_{ij} as the OSTRE ratios:

$$\alpha_{ij} \equiv \sqrt{\frac{b_{i1}\, p_{i1}\left(1-p_{ij}\right)}{b_{ij}\, p_{ij}\left(1-p_{i1}\right)}} = \frac{n_{ij}}{n_{i1}}$$

$$\alpha_{i1} \equiv 1$$

$$n_{ij} = \alpha_{ij}\, n_{i1}$$

The runs allocated to each design (the notional N_i) must equal the sum of the runs allocated to the levels of that design. So, under the OSTRE ratios:

$$N_i = n_{i1} + n_{i2} + ... + n_{iL} = \alpha_{i1}\, n_{i1} + \alpha_{i2}\, n_{i1} + ... + \alpha_{iL}\, n_{i1} = \left(\alpha_{i1} + ... + \alpha_{iL}\right) n_{i1}$$

Define: $\alpha_i \equiv \alpha_{i1} + ... + \alpha_{iL}$.

so: $N_i = \alpha_i\, n_{i1}$

Then: $n_{ij} = \alpha_{ij}\, n_{ij} = \dfrac{\alpha_{ij}\, N_i}{\alpha_i}$

These are the theoretical links between the notional OCBA allocations, the N_i, and the runs at the various levels, the n_{ij}. In effect they make the notional N_i for a design consistent with the optimal n_{ij} within the design.

The time allocated to each design equals the sum of the times allocated to its levels:

$$b_i N_i \;=\; b_{i1}\, n_{i1} \;+ b_{i2}\, n_{i2} \;+ \ldots. +\; b_{iL}\, n_{iL} \;=\; \frac{\alpha_{i1}\, b_{i1} N_i}{\alpha_i} \;+ \ldots. + \frac{\alpha_{iL}\, b_{iL} N_i}{\alpha_i} \;=\; \frac{N_i}{\alpha_i} \sum_{j=1}^{L} \alpha_{ij}\, b_{ij}$$

From this emerges the elusive b_i:

$$b_i \;=\; \frac{1}{\alpha_i} \sum_{j=1}^{L} \alpha_{ij}\, b_{ij}$$

The algorithm can estimate this b_i: it estimates the b_{ij}, since it does make runs at the various levels; the α_{ij} are defined by the optimal OSTRE relationships, which can be estimated, at each update, from the current estimates $\hat{p}_{ij}$ and $\hat{b}_{ij}$; α_i is a definition, from the α_{ij}.

7.1 The OCBA Step

$$Min\; P\left[\hat{p}_1 > \hat{p}_i, \ldots, \hat{p}_1 > \hat{p}_D\right] \quad \text{s/t} \quad T = \sum_{i=1}^{D} b_i N_i \;=\; \sum_{i=1}^{D} \left(\frac{1}{\alpha_i} \sum_{j=1}^{L} \alpha_{ij}\, b_{ij} \right) N_i$$

This is similar to the OCBA problem in Chen & Lee (2011), but different in two important features: in the b_i, as explained above, and in the variances of the design estimators.

Standard OCBA assumes the design estimators are simple Monte Carlo, normally distributed. The OCBA optimality equations are

$$\frac{1}{b_1} \sum_{i=2}^{D} e^{-\frac{\phi_i^2}{2}} \frac{\partial \phi_i}{\partial N_1} \;=\; \frac{1}{b_2} e^{-\frac{\phi_2^2}{2}} \frac{\partial \phi_2}{\partial N_2} \;=\; \ldots.. \;=\; \frac{1}{b_D} e^{-\frac{\phi_D^2}{2}} \frac{\partial \phi_D}{\partial N_D}$$

with $$\phi_i = \frac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}}$$

The derivation of the OCBA step in OCBA+OSTRE is formally identical to this, but differs in substance because the b_i and the design variances are different. The appropriate expression for b_i is developed above. In standard OCBA the design variances are

$$\sigma_i^2 = \frac{p_i(1-p_i)}{N_i}.$$

This is the variance of a scaled binomial, reflecting the fact that, in standard OCBA, the MC runs, covering the entire design, produce hits that are binomially distributed.

But OCBA+OSTRE will do no MC runs over the entire design. All its runs are in the OSTRE step, over levels. Its variance is the variance of the product of the level estimators:

$$\sigma_i^2 = Var\left[\prod_{j=1}^{L} \hat{p}_{ij}\right]$$

In Single Optimization the truncated version of this variance is employed. OCBA+OSTRE employs the same truncated version:

$$\sigma_i^2 = p_i^2\left[\frac{1-p_{i1}}{p_{i1}n_{i1}} + \cdots + \frac{1-p_{iL}}{p_{iL}n_{iL}}\right]$$

But this truncated variance is expressed in terms of the n_{ij}, whereas the OCBA step is based on the variance expressed in terms of the (notional) N_i. The transformation from one to the other is accomplished via the link between the n_{ij} and the N_i developed above:

$$n_{ij} = \alpha_{ij} n_{i1} = \frac{\alpha_{ij} N_i}{\alpha_i}$$

The design variance becomes:

$$\sigma_i^2(N_i) = p_i^2 \left[\frac{\alpha_i (1-p_{i1})}{p_{i1} \alpha_{i1} N_i} + \cdots + \frac{\alpha_i (1-p_{iL})}{p_{iL} \alpha_{iL} N_i} \right] = \frac{\alpha_i p_i^2}{N_i} \sum_{k=1}^{L} \frac{(1-p_{ik})}{p_{ik} \alpha_{ik}}.$$

Or, more succinctly:

$$\sigma_i^2(N_i) = \frac{c_i}{N_i}, \qquad c_i \equiv \alpha_i p_i^2 \sum_{k=1}^{L} \frac{(1-p_{ik})}{p_{ik} \alpha_{ik}}$$

with derivative:

$$\frac{\partial \sigma_i^2}{\partial N_i} = -\frac{c_i}{N_i^2}.$$

At this point it might be useful to reflect on what the theoretical link between the N_i and the n_{ij} tries to accomplish. The goal is to construct OCBA+OSTRE so that it is theoretically equivalent to Single Optimization, so that its solution is the same as Single Optimization solution. It would seem – a conjecture at this point! – that such equivalence would require that, at the optimum, the OSTRE ratios hold within each design. So OCBA+OSTRE imposes those ratios within each design. It must, then, construct its

notional N_i so that, when it calculates the design variances in the OCBA step, it is calculating the variances implied by the OSTRE ratios it imposes on the n_{ij}.

It is, at this point, only a conjecture that this construction of the relationships between the notional N_i and the n_{ij} will produce an OCBA+OSTRE that is equivalent to Single Optimization. The next chapter will prove this conjecture correct.

To complete the OCBA step, the optimality equations, built on the constructed relationships between the N_i and the n_{ij}, are:

$$\frac{1}{b_1}\sum_{i=2}^{D} e^{-\frac{\phi_i^2}{2}} \frac{\partial \phi_i}{\partial N_1} = \frac{1}{b_2} e^{-\frac{\phi_2^2}{2}} \frac{\partial \phi_2}{\partial N_2} = ... = \frac{1}{b_D} e^{-\frac{\phi_D^2}{2}} \frac{\partial \phi_D}{\partial N_D}$$

$$\phi_i\left(N_1, N_i\right) = \frac{p_i - p_1}{\sqrt{\sigma_1^2\left(N_1\right) + \sigma_i^2\left(N_i\right)}} \qquad i = 2,...,D$$

$$\frac{\partial \phi_i}{\partial N_m} = \frac{\left(p_i - p_1\right) c_m}{2\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}} N_m^2} \qquad i = 2,...,D \quad m = 1 \text{ or } i$$

$$\frac{1}{b_1}\sum_{i=2}^{D} \frac{e^{-\frac{\phi_i^2}{2}}\left(p_i - p_1\right) c_1}{2\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}} N_1^2} = \frac{e^{-\frac{\phi_2^2}{2}}\left(p_2 - p_1\right) c_2}{2b_2\left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}} N_2^2} = ... = \frac{e^{-\frac{\phi_D^2}{2}}\left(p_D - p_1\right) c_D}{2b_D\left(\sigma_1^2 + \sigma_D^2\right)^{\frac{3}{2}} N_D^2}$$

$$\frac{c_1}{b_1 N_1^2}\sum_{i=2}^{D} \frac{e^{-\frac{\phi_i^2}{2}}\left(p_i - p_1\right)}{\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}}} = \frac{e^{-\frac{\phi_2^2}{2}}\left(p_2 - p_1\right) c_2}{b_2\left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}} N_2^2} = ... = \frac{e^{-\frac{\phi_D^2}{2}}\left(p_D - p_1\right) c_D}{b_D\left(\sigma_1^2 + \sigma_D^2\right)^{\frac{3}{2}} N_D^2}$$

$$\frac{N_1}{N_2} = \left[\frac{b_2 c_1 \left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}} \sum_{i=2}^{D} \frac{e^{\frac{-\phi_i^2}{2}} \left(p_i - p_1\right)}{\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}}}}{b_1 e^{-\frac{\phi_2^2}{2}} \left(p_2 - p_1\right) c_2} \right]^{\frac{1}{2}}$$

$$\frac{N_2}{N_3} = \left[\frac{b_3 c_2 \left(p_2 - p_1\right)\left(\sigma_1^2 + \sigma_3^2\right)^{\frac{3}{2}}}{b_2 c_3 \left(p_3 - p_1\right)\left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}}} \right]^{\frac{1}{2}} e^{\frac{\phi_3^2 - \phi_2^2}{4}}$$

$$\frac{N_{D-1}}{N_D} = \left[\frac{b_D c_{D-1} \left(p_{D-1} - p_1\right)\left(\sigma_1^2 + \sigma_D^2\right)^{\frac{3}{2}}}{b_{D-1} c_D \left(p_D - p_1\right)\left(\sigma_1^2 + \sigma_{D-1}^2\right)^{\frac{3}{2}}} \right]^{\frac{1}{2}} e^{\frac{\phi_D^2 - \phi_{D-1}^2}{4}}$$

These ratios, with a budget constraint, generate the theoretical solutions N_i^*. In practice they guide, at each update, the selection of the design for the next allocation of runs.

7.2 The OSTRE Step

The OSTRE step could be stated as the OSTRE optimization problem, but that is an unnecessary formality. The solution to it has already been built into the N_i^*. The only thing required at the OSTRE step is to calculate the OSTRE solutions, from

$$n_{ij}^* = \frac{\alpha_{ij} N_i^*}{\alpha_i}.$$

7.3 OCBA+OSTRE Implementation

As with all the optimizing algorithms (OCBA, OSTRE, Single Optimization), OCBA+OSTRE must be implemented in stages, or updates. At each update estimates of all the parameters are substituted for the real parameters of the theory. At each update a (relatively small) number of new simulation runs is to be performed – at a particular level within a particular design. The design is first chosen, via the OCBA step, so as to minimize the actual inequalities among the OCBA optimality equalities. That determines the i in N_i. The number of new simulation runs is fixed, at some Δ. Thus, at each update, $N_i = \Delta$. These Δ runs are then performed at level *j* in design *i*, with *j* selected to minimize the actual inequalities within the theoretical OSTRE optimality equalities. Thus, $N_i = \Delta = n_{ij}$ at each update, the *i* chosen in the OCBA step, the *j* in the OSTRE step. The algorithm terminates when elapsed time reaches the budget constraint, *T*.

Initialization in OCBA+OSTRE is exactly the same as in Single Optimization. Arbitrary allocations are made to all the designs and levels. Absent a heuristic to guide these allocations, they are made equally: an equal amount of time (or runs) devoted to simulations at each level, within each design. That is equivalent to standard Splitting, for initialization.

On the OSTRE step the algorithm implements the update exactly as the updates in standard OSTRE are implemented. In the test model the same code is used (the same updating procedure is called). However, the OCBA step update differs from the standard OCBA update. The standard OCBA update, as employed in the test model for OCBA

simulations, uses the asymptotic simplification recommended by Chen & Lee (2011, pp. 46, 57.) It is difficult to discern how such a simplification could be imposed on the OCBA step update, in OCBA+OSTRE. Instead, the full optimality equations are employed. They do not permit an explicit solution for the N_i^*. Resort to numerical methods to attain a solution would be computationally expensive. Instead, the *MaxDiff* method, as employed in Single Optimization, is used to select the design most deserving of the next allocation.

The optimality terms at each OCBA step update contain b_i and c_i, constructed from α_i and α_{ij}. These parameters are defined in the OCBA step theory and can be estimated within the algorithm. They are used to determine the optimal N_i^* toward which the algorithm tries to drive the actual, non-optimal N_i. *MaxDiff* operates by comparing, at each update, the non-optimal N_i with the optimal N_i^*. But this poses a problem. In an OCBA step update there are no "real" N_i. The N_i allocated to a design become runs at a level within that design, not true MC runs over the whole design. It would be inconsistent to treat those "bogus" N_i, in the update, as true N_i.

What to do? This issue does not arise for the N_i^* because they are derived in theory. But the N_i to be compared with the theoretically optimal N_i^* must be computable, in practice, despite the lack of actual N_i.

The correct way to calculate the N_i, at each update, is found by examining the link between N_i and n_{ij} at the optimum. Recall that:

$$\alpha_{ij} \equiv \sqrt{\frac{b_{i1}\, p_{i1}\left(1-p_{ij}\right)}{b_{ij}\, p_{ij}\left(1-p_{i1}\right)}} = \frac{n_{ij}}{n_{i1}} \quad \text{(the optimum OSTRE ratio)}$$

$$\alpha_i \equiv \alpha_{i1} + \ldots + \alpha_{iL}$$

$$N_i = \alpha_i\, n_{i1}$$

At the optimum the link between N_i and n_{ij} flows from the optimum OSTRE ratio. In implementation neither N_i nor n_{ij} will be optimal at any finite update. The algorithm has actual n_{ij} at each update, but must calculate an appropriate notional N_i to be compared with the optimal N_i^*.

The solution is to define the α_{ij} differently, for implementing the algorithm. The α_{ij} impose the optimal OSTRE ratios. They might be better labeled as α_{ij}^*. The α_{ij} used in updates should, instead, be based on the actual, not the OSTRE optimal, ratios. Taking the actual n_{ij} at each update:

$$\alpha_{ij} = \frac{n_{ij}}{n_{i1}}$$

$$\alpha_i \equiv \alpha_{i1} + \ldots + \alpha_{iL}$$

$$N_i = \alpha_i\, n_{i1}$$

This change means, simply, that the notional N_i are calculated as though they had been proportionally divided among the level runs that actually occurred. The updates try to move the OCBA and the OSTRE optimality terms toward equality, which would ensure that the optimal OCBA ratios among the N_i and the optimal OSTRE ratios among the n_{ij} are attained.

7.4 Complexity Analysis

As with Single Optimization, the complexity analysis for OCBA+OSTRE focuses solely on how the computational effort scales with problem size, measured by the number of designs (*D*) and the number of levels per design (*L*). It is based on the same assumptions as those made for Single Optimization. It has the same initialization phase but differs from Single Optimization in that it has phases for calculating the optimality terms and updating the allocations at both the OCBA step and the OSTRE step.

The initialization phase is identical to that of Single Optimization:

$$Initialization \sim O\left[c_1 DL\right]$$

The optimality equations for the OCBA step are

$$\frac{1}{b_1}\sum_{i=2}^{D} e^{-\frac{\phi_i^2}{2}}\frac{\partial \phi_i}{\partial N_1} = \frac{1}{b_2}e^{-\frac{\phi_2^2}{2}}\frac{\partial \phi_2}{\partial N_2} = ... = \frac{1}{b_D}e^{-\frac{\phi_D^2}{2}}\frac{\partial \phi_D}{\partial N_D}$$

The terms $e^{-\frac{\phi_2^2}{2}}\frac{\partial \phi_i}{\partial N_1}$ must be calculated (*D*-1) times, then the terms $e^{-\frac{\phi_i^2}{2}}\frac{\partial \phi_2}{\partial N_i}$ are calculated (*D*-1) times. In total:

$$OCBA\ optimality \sim O\left[c_2\left(D-1\right)\right]$$

OCBA allocation employs the *MaxDiff* method but differs from the Single Optimization allocation because it is only spread over the designs, there being no levels at the OCBA step. Thus, whereas Single Optimization allocation was bounded by $O\left[c\left(D-1\right)L\right]$, the OCBA step allocation is bounded by

$$OCBA\ allocation\ \sim\ O\left[c_3\left(D-1\right)\right]$$

The OSTRE step optimality terms are calculated in only one design (the design selected in the OCBA step), so it scales solely with the number of levels per design, *L*:

$$OSTRE\ optimality\ \sim\ O\left[c_4\,L\right]$$

Likewise the OSTRE step allocations require only a set of calculations (solving for the optimal n_{ij}^{*} and comparing with the actual n_{ij}) which scales only with *L*:

$$OSTRE\ optimality\ \sim\ O\left[c_5\,L\right]$$

In total:

OCBA+OSTRE Effort = Initialization + OCBA optimality + OCBA allocation + OSTRE optimality + OSTRE allocation.

$$\text{OCBA+OSTRE Effort}\ \sim\ O\left[c_1 DL\right]+O\left[c_2\left(D-1\right)\right]+O\left[c_3\left(D-1\right)\right]+O\left[c_4\,L\right]+O\left[c_5\,L\right],$$

or, for some c:

$$\text{OCBA+OSTRE}\ \sim\ O\left[c_1 DL\right]\ +\ O\left[c\left(D-1\right)\right]\ +\ O\left[c\,L\right]$$

Compare this with the Single Optimization Effort. The initialization phases are identical in both methods. Assume a common c has been set, sufficient to bound all the other phases, in both methods. The Single Optimization effort may be expressed as:

$$\text{Single Optimization } \sim\ O\left[c_1 DL\right] \ +\ O\left[c\left(D-1\right)L\right] \ +\ O\left[cDL\right].$$

It is clear that OCBA+OSTRE is computationally more efficient than Single Optimization, in the sense that its required effort increases less than the required effort for Single Optimization, as *D* and *L* increase. For large *D* or *L* the difference can be considerable. As an approximation, replace the *D*-1 term in both expressions with *D*. There is, then, a direct comparison:

$$\text{Single Optimization Effort } \sim\ O\left[cDL\right]$$

$$\text{OCBA+OSTRE Effort } \sim\ O\left[c\left(D+L\right)\right]$$

The computational work required of Single Optimization increases (approximately) as the product of *D* and *L* increases, whereas that of OCBA+OSTRE increases only as their sum increases.

That difference can be substantial, rendering Single Optimization, for relatively large *D* and/or *L*, a very unattractive method. But OCBA+OSTRE becomes an attractive alternative only if it is, at least approximately, mathematically just as good as Single Optimization. Is it? Read the next chapter!

8. OCBA+OSTRE vs Single Optimization: Mathematical Equivalence

Theorem: Single Optimization and OCBA+OSTRE are, at the optimum, mathematically equivalent.

Let: $n_{ij}^{S} \equiv$ the Single Optimization optimum allocation of runs to design *i* , level j

$n_{ij}^{O} \equiv$ the OCBA+OSTRE optimum allocation of runs to design *i* , level *j*

Equivalence means that, given a common budget constraint:

$$n_{ij}^{S} = n_{ij}^{O} \quad all\ i, j$$

Given a common budget constraint, equivalence is proved by showing that two conditions are met:

(1) The ratios of runs allocated to different levels within a given design are, under the two methods, the same. Condition (1) is satisfied if, within each design *i*, for any levels *j* and *k*:

$$\frac{n_{ij}^{O}}{n_{ik}^{O}} = \frac{n_{ij}^{S}}{n_{ik}^{S}}$$

(2) The ratios of runs allocated to levels between different designs are, under the two methods, the same. Let *i, k* be different designs, *j, m* any levels within those

designs. Condition (2) means that:

$$\frac{n_{ij}^{O}}{n_{km}^{O}} = \frac{n_{ij}^{S}}{n_{km}^{S}}$$

Condition 2, alone, would suffice to prove equivalence. It is, however, useful to first prove Condition 1, and draw on it in the proof of Condition 2.

Claim: Condition (1) is satisfied.

Proof:

OCBA+OSTRE imposes the OSTRE optimality equations to link all levels within a design. The optimum ratios of runs between levels within a given design in OCBA+OSTRE are, by construction, the OSTRE ratios. Proving this claim thus requires showing that the Single Optimization optimality equations linking the levels within a design also generate OSTRE ratios.

The Single Optimization optimality equations differ, in form, between design 1 and all other designs, so the proof for design 1 is a bit different from the proof for the other designs.

8.1 OSTRE Holds Within Design 1 of Single Optimization

For design 1, levels *i* and *j*, the optimality equations have the form:

$$\frac{1}{b_{1i}} \sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \frac{\partial \phi_k}{\partial n_{1i}} = \frac{1}{b_{1j}} \sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \frac{\partial \phi_k}{\partial n_{1j}} \qquad i, j \in \{1,...,L\} \quad i \neq j$$

This may be rearranged as:

$$\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \left[\frac{1}{b_{1i}} \frac{\partial \phi_k}{\partial n_{1i}} - \frac{1}{b_{1j}} \frac{\partial \phi_k}{\partial n_{1j}} \right] = 0$$

Substituting the derivatives of the ϕ_k functions, the term in brackets becomes:

$$\frac{1}{b_{1i}}\frac{\partial \phi_k}{\partial n_{1i}} - \frac{1}{b_{1j}}\frac{\partial \phi_k}{\partial n_{1j}} = \frac{(p_k - p_1)\,p_k^2\,(1-p_{1i})}{2b_{1i}\,p_{1i}\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}} n_{1i}^2} - \frac{(p_k - p_1)\,p_k^2\,(1-p_{1j})}{2b_{1j}\,p_{1j}\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}} n_{1j}^2}$$

$$\frac{1}{b_{1i}}\frac{\partial \phi_k}{\partial n_{1i}} - \frac{1}{b_{1j}}\frac{\partial \phi_k}{\partial n_{1j}} = \left[\frac{(p_k - p_1)\,p_k^2}{2\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}\right]\left[\frac{1-p_{1i}}{b_{1i}\,p_{1i}\,n_{1i}^2} - \frac{1-p_{1j}}{b_{1j}\,p_{1j}\,n_{1j}^2}\right]$$

The design 1 optimality equations may be re-written as:

$$\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\left[\frac{(p_k - p_1)\,p_k^2}{2\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}\right]\left[\frac{1-p_{1i}}{b_{1i}\,p_{1i}\,n_{1i}^2} - \frac{1-p_{1j}}{b_{1j}\,p_{1j}\,n_{1j}^2}\right] = 0$$

$$\left[\frac{1-p_{1i}}{b_{1i}\,p_{1i}\,n_{1i}^2} - \frac{1-p_{1j}}{b_{1j}\,p_{1j}\,n_{1j}^2}\right]\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\left[\frac{(p_k - p_1)\,p_k^2}{2\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}\right] = 0$$

Each term $\dfrac{(p_k - p_1)\,p_k^2}{2\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}$ is positive, since the optimality equations were derived under the stipulation that design 1 is optimal: $p_k > p_1$, all $k > 1$. The optimality equations are thus satisfied iff, for each *i*, *j*:

$$\frac{1-p_{1i}}{b_{1i}\,p_{1i}\,n_{1i}^2} - \frac{1-p_{1j}}{b_{1j}\,p_{1j}\,n_{1j}^2} = 0$$

or

$$\frac{b_{1i}\,p_{1i}\,n_{1i}^2}{1-p_{1i}} = \frac{b_{1j}\,p_{1j}\,n_{1j}^2}{1-p_{1j}}.$$

This is precisely the OSTRE optimality equation linking levels *i* and *j*, within design 1. OSTRE ratios thus apply within design 1 of Single Optimization.

8.2 OSTRE Holds Within All Other Designs of Single Optimization

The Single Optimization optimality equations for designs 2 through D are simpler, as there is no summation of terms. For design $k > 1$, levels *i* and *j*:

$$\frac{1}{b_{ki}} e^{\frac{-\phi_k^2}{2}} \frac{\partial \phi_k}{\partial n_{ki}} = \frac{1}{b_{kj}} e^{\frac{-\phi_k^2}{2}} \frac{\partial \phi_k}{\partial n_{kj}}$$

$$\frac{1}{b_{ki}} \frac{\partial \phi_k}{\partial n_{ki}} = \frac{1}{b_{kj}} \frac{\partial \phi_k}{\partial n_{kj}}$$

$$\frac{\left(p_k - p_1\right) p_k^2 \left(1 - p_{ki}\right)}{2 b_{ki}\, p_{ki} \left(\sigma_1^2 + \sigma_k^2\right) n_{ki}^2} = \frac{\left(p_k - p_1\right) p_k^2 \left(1 - p_{kj}\right)}{2 b_{kj}\, p_{kj} \left(\sigma_1^2 + \sigma_k^2\right) n_{kj}^2}$$

$$\frac{\left(1 - p_{ki}\right)}{b_{ki}\, p_{ki}\, n_{ki}^2} = \frac{\left(1 - p_{kj}\right)}{b_{kj}\, p_{kj}\, n_{kj}^2}$$

$$\frac{b_{ki}\, p_{ki}\, n_{ki}^2}{1 - p_{ki}} = \frac{b_{kj}\, p_{kj}\, n_{kj}^2}{1 - p_{kj}}$$

This is the OSTRE optimality equation linking levels *i* and *j*, within design *k*, $k = 2,\ldots D$.

The equations linking any two levels within a design are, in both methods, equivalent to the OSTRE optimality equations. Condition 1 is satisfied.

Claim: Condition 2 is satisfied.

Proof: It will be shown that

$$\frac{n_{ij}^{S}}{n_{km}^{S}} = \frac{n_{ij}^{O}}{n_{km}^{O}}$$

by deriving these ratios from their respective optimality equations. As in Condition (1), the proof is somewhat different for the links (or ratios) between design 1 and other designs and the links between any of the designs other than 1.

8.3 Cross design links, Designs Other Than 1

This will be proven by deriving the ratios from their respective optimality equations. For $i,k>1,\ \ j,m\in\{1,...,L\}$

Single Optimization: $$\frac{1}{b_{ij}}e^{\frac{-\phi_i^2}{2}}\frac{\partial\phi_i}{\partial n_{ij}} = \frac{1}{b_{km}}e^{\frac{-\phi_k^2}{2}}\frac{\partial\phi_k}{\partial n_{km}}$$

OCBA+OSTRE: $$\frac{1}{b_i}e^{\frac{-\phi_k^2}{2}}\frac{\partial\phi_i}{\partial N_i} = \frac{1}{b_k}e^{\frac{-\phi_k^2}{2}}\frac{\partial\phi_k}{\partial N_k}$$

Complete notation for the n_{ij}, n_{km} would be $n_{ij}^{S},\ n_{km}^{S}$ for Single Optimization, n_{ij}^{O}, n_{km}^{O} for OCBA+OSTRE, but the subscripts are suppressed when it is clear from the context which set of runs is meant.

$$\frac{1}{b_{ij}}e^{\frac{-\phi_i^2}{2}}\frac{\partial\phi_i}{\partial n_{ij}} = \frac{1}{b_{km}}e^{\frac{-\phi_k^2}{2}}\frac{\partial\phi_k}{\partial n_{km}}\ .$$

Expanding:

$$\frac{1}{b_{ij}} e^{\frac{-\phi_i^2}{2}} \frac{(p_i - p_1) p_i^2 (1 - p_{ij})}{2 p_{ij} (\sigma_1^2 + \sigma_i^2)^{\frac{3}{2}} n_{ij}^2} = \frac{1}{b_{km}} e^{\frac{-\phi_k^2}{2}} \frac{(p_k - p_1) p_k^2 (1 - p_{km})}{2 p_{km} (\sigma_1^2 + \sigma_k^2)^{\frac{3}{2}} n_{km}^2}$$

$$\frac{b_{ij} p_{ij} (\sigma_1^2 + \sigma_i^2)^{\frac{3}{2}} n_{ij}^2}{(p_i - p_1) p_i^2 (1 - p_{ij})} e^{\frac{\phi_i^2}{2}} = \frac{b_{km} p_{km} (\sigma_1^2 + \sigma_k^2)^{\frac{3}{2}} n_{km}^2}{(p_k - p_1) p_k^2 (1 - p_{km})} e^{\frac{\phi_k^2}{2}}$$

$$\frac{n_{ij}^2}{n_{km}^2} = \frac{(p_i - p_1) p_i^2 (1 - p_{ij}) b_{km} p_{km} (\sigma_1^2 + \sigma_k^2)^{\frac{3}{2}}}{(p_k - p_1) p_k^2 (1 - p_{km}) b_{ij} p_{ij} (\sigma_1^2 + \sigma_i^2)^{\frac{3}{2}}} e^{\frac{\phi_k^2 - \phi_i^2}{2}}$$

Cross Design Links in OCBA+OSTRE, All Designs Other Than 1

The ratio of n_{ij} to n_{km} in OCBA+OSTRE is derived from the ratio of N_i to N_k.

The derivation requires these relationships (introduced above):

$$\alpha_{ij} \equiv \sqrt{\frac{b_{i1} p_{i1} (1 - p_{ij})}{b_{ij} p_{ij} (1 - p_{i1})}} = \frac{n_{ij}}{n_{i1}}$$ (The OSTRE relationship between n_{ij} and n_{i1})

$$n_{ij} = \alpha_{ij} n_{i1} \quad (\alpha_{i1} = 1)$$

$$N_i = n_{i1} + ... + n_{iL} = \alpha_{i1} n_{i1} + \alpha_{i2} n_{i1} + ... + \alpha_{iL} n_{i1} = (\alpha_{i1} + ... + \alpha_{iL}) n_{i1}$$

$$\alpha_i \equiv \alpha_{i1} + ... + \alpha_{iL} \qquad N_i = \alpha_i n_{i1}$$

$$b_i = \frac{1}{\alpha_i} \sum_{n=1}^{L} \alpha_{in} b_{in}$$

$$c_i = \alpha_i p_i^2 \sum_{n=1}^{L} \frac{1 - p_{in}}{\alpha_{in} p_{in}}$$, derived from the truncated variance $\sigma_i^2 (N_i) = \frac{c_i}{N_i}$

The OCBA+OSTRE optimality equations:

$$\frac{1}{b_i} e^{\frac{-\phi_i^2}{2}} \frac{\partial \phi_i}{\partial N_i} = \frac{1}{b_k} e^{\frac{-\phi_k^2}{2}} \frac{\partial \phi_k}{\partial N_k}$$

$$e^{\frac{-\phi_i^2}{2}} \frac{(p_i - p_1)c_i}{2\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}} N_i^2} = \frac{b_i}{b_k} e^{\frac{-\phi_k^2}{2}} \frac{(p_k - p_1)c_k}{2\left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}} N_k^2}$$

$$\frac{\left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}} N_i^2}{(p_i - p_1)c_i} e^{\frac{\phi_i^2}{2}} = \frac{b_k \left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}} N_k^2}{b_i (p_k - p_1)c_k} e^{\frac{\phi_k^2}{2}}$$

$$\frac{N_i^2}{N_k^2} = \frac{(p_i - p_1)c_i b_k \left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}}}{(p_k - p_1)c_k b_i \left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}}} e^{\frac{\phi_k^2 - \phi_i^2}{2}}$$

Substituting $N_i = \alpha_i n_{i1}$:

$$\frac{\alpha_i^2 n_{ij}^2}{\alpha_k^2 n_{km}^2} = \frac{(p_i - p_1)c_i b_k \left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}}}{(p_k - p_1)c_k b_i \left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}}} e^{\frac{\phi_k^2 - \phi_i^2}{2}}$$

$$\frac{n_{ij}^2}{n_{km}^2} = \frac{\alpha_k^2 (p_i - p_1)c_i b_k \left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}}}{\alpha_i^2 (p_k - p_1)c_k b_i \left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}}} e^{\frac{\phi_k^2 - \phi_i^2}{2}}$$

The ratios of n_{ij} to n_{km} in OCBA+OSTRE and Single Optimization are the same iff:

OCBA+OSTRE Ratio = Single Optimization Ratio

$$\frac{\alpha_k^2\left(p_i-p_1\right)c_i\,b_k\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}{\alpha_i^2\left(p_k-p_1\right)c_k\,b_i\left(\sigma_1^2+\sigma_i^2\right)^{\frac{3}{2}}}e^{\frac{\phi_k^2-\phi_i^2}{2}}=\frac{\left(p_i-p_1\right)p_i^2\left(1-p_{ij}\right)b_{km}p_{km}\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}{\left(p_k-p_1\right)p_k^2\left(1-p_{km}\right)b_{ij}p_{ij}\left(\sigma_1^2+\sigma_i^2\right)^{\frac{3}{2}}}e^{\frac{\phi_k^2-\phi_i^2}{2}}$$

After cancellations:

$$\frac{\alpha_k^2\,c_i\,b_k}{\alpha_i^2\;c_k\,b_i}\;=\;\frac{p_i^2\left(1-p_{ij}\right)b_{km}p_{km}}{p_k^2\left(1-p_{km}\right)b_{ij}p_{ij}}$$

Is this true?

Expand $\frac{\alpha_k^2\,c_i\,b_k}{\alpha_i^2\;c_k\,b_i}$ in terms of $p_i,\ p_k,\ p_{ij},\ p_{km},\ b_i,\ b_k,\ b_{ij},\ b_{km}$:

Expressions for $c_i,\ c_k$, from substituting $\alpha_{ij}\;=\;\sqrt{\frac{b_{i1}\,p_{i1}\left(1-p_{ij}\right)}{b_{ij}\;p_{ij}\left(1-p_{i1}\right)}}$:

$$c_i=\alpha_i p_i^2\sum_{j=1}^{L}\frac{1-p_{ij}}{\alpha_{ij}p_{ij}}=\alpha_i p_i^2\sum_{j=1}^{L}\frac{1-p_{ij}}{p_{ij}\sqrt{\frac{b_{i1}p_{i1}\left(1-p_{ij}\right)}{b_{ij}p_{ij}\left(1-p_{i1}\right)}}}=\alpha_i p_i^2\sqrt{\frac{1-p_{i1}}{b_{i1}p_{i1}}}\sum_{j=1}^{L}\sqrt{\frac{b_{ij}\left(1-p_{ij}\right)}{p_{ij}}}$$

The expansion of c_k is the same, indexed at *k* instead of *i*:

$$c_k\;=\;\alpha_k p_k^2\sqrt{\frac{1-p_{k1}}{b_{k1}p_{k1}}}\sum_{j=1}^{L}\sqrt{\frac{b_{kj}\left(1-p_{kj}\right)}{p_{kj}}}$$

Now we have:

$$\frac{\alpha_k^2 c_i b_k}{\alpha_i^2 c_k b_i} = \frac{\alpha_k^2 \alpha_i p_i^2 \sqrt{\frac{1-p_{i1}}{b_{i1} p_{i1}}} \left(\sum_{j=1}^{L} \sqrt{\frac{b_{ij}\left(1-p_{ij}\right)}{p_{ij}}} \right) b_k}{\alpha_i^2 \alpha_k p_k^2 \sqrt{\frac{1-p_{k1}}{b_{k1} p_{k1}}} \left(\sum_{j=1}^{L} \sqrt{\frac{b_{kj}\left(1-p_{kj}\right)}{p_{kj}}} \right) b_i}$$

After substituting $b_i = \frac{1}{\alpha_i} \sum_{n=1}^{L} \alpha_{in} b_{in}, \quad b_k = \frac{1}{\alpha_k} \sum_{n=1}^{L} \alpha_{kn} b_{kn}$ and cancelling:

$$\frac{\alpha_k^2 c_i b_k}{\alpha_i^2 c_k b_i} = \frac{p_i^2 \sqrt{\frac{1-p_{i1}}{b_{i1} p_{i1}}} \left(\sum_{n=1}^{L} \sqrt{\frac{b_{in}\left(1-p_{in}\right)}{p_{in}}} \right) \sum_{n=1}^{L} \alpha_{kn} b_{kn}}{p_k^2 \sqrt{\frac{1-p_{k1}}{b_{k1} p_{k1}}} \left(\sum_{n=1}^{L} \sqrt{\frac{b_{kn}\left(1-p_{kn}\right)}{p_{kn}}} \right) \sum_{n=1}^{L} \alpha_{in} b_{in}}$$

Expand the $\sum_{n=1}^{L} \alpha_{in} b_{in}$, $\sum_{n=1}^{L} \alpha_{kn} b_{kn}$ terms and substitute into $\frac{\alpha_k^2 c_i b_k}{\alpha_i^2 c_k b_i}$:

$$\sum_{n=1}^{L} \alpha_{in} b_{in} = \sqrt{\frac{b_{i1} p_{i1}}{1-p_{i1}}} \sum_{n=1}^{L} b_{in} \sqrt{\frac{1-p_{in}}{b_{in} p_{in}}} = \sqrt{\frac{b_{i1} p_{i1}}{1-p_{i1}}} \sum_{n=1}^{L} \sqrt{\frac{b_{in}\left(1-p_{in}\right)}{p_{in}}}$$

$$\sum_{n=1}^{L} \alpha_{kn} b_{kn} = \sqrt{\frac{b_{k1} p_{k1}}{1-p_{k1}}} \sum_{n=1}^{L} \sqrt{\frac{b_{kn}\left(1-p_{kn}\right)}{p_{kn}}}$$

$$\frac{\alpha_k^2 c_i b_k}{\alpha_i^2 c_k b_i} = \frac{p_i^2 \sqrt{\frac{1-p_{i1}}{b_{i1} p_{i1}}} \left(\sum_{n=1}^{L} \sqrt{\frac{b_{in}\left(1-p_{in}\right)}{p_{in}}} \right) \sqrt{\frac{b_{k1} p_{k1}}{1-p_{k1}}} \sum_{n=1}^{L} \sqrt{\frac{b_{kn}\left(1-p_{kn}\right)}{p_{kn}}}}{p_k^2 \sqrt{\frac{1-p_{k1}}{b_{k1} p_{k1}}} \left(\sum_{n=1}^{L} \sqrt{\frac{b_{kn}\left(1-p_{kn}\right)}{p_{kn}}} \right) \sqrt{\frac{b_{i1} p_{i1}}{1-p_{i1}}} \sum_{n=1}^{L} \sqrt{\frac{b_{in}\left(1-p_{in}\right)}{p_{in}}}}$$

Cancellations simplify this to:

$$\frac{\alpha_k^2\, c_i\, b_k}{\alpha_i^2\; c_k\, b_i} \;=\; \frac{p_i^2\sqrt{\dfrac{1-p_{i1}}{b_{i1}p_{i1}}}\;\sqrt{\dfrac{b_{k1}p_{k1}}{1-p_{k1}}}}{p_k^2\sqrt{\dfrac{1-p_{k1}}{b_{k1}p_{k1}}}\;\sqrt{\dfrac{b_{i1}p_{i1}}{1-p_{i1}}}} \;=\; \frac{p_i^2\,\sqrt{1-p_{i1}}\,\sqrt{b_{k1}p_{k1}}\,\sqrt{b_{k1}p_{k1}}\,\sqrt{1-p_{i1}}}{p_k^2\;\sqrt{b_{i1}p_{i1}}\;\sqrt{1-p_{k1}}\;\sqrt{1-p_{k1}}\;\sqrt{b_{i1}p_{i1}}}$$

$$\frac{\alpha_k^2\, c_i\, b_k}{\alpha_i^2\; c_k\, b_i} \;=\; \frac{p_i^2\left(1-p_{i1}\right) b_{k1}\, p_{k1}}{p_k^2\left(1-p_{k1}\right) b_{i1}\, p_{i1}} \;=\; \text{Single Optimization Ratio} \qquad \textit{QED}$$

Note that the OCBA+OSTRE ratio $\frac{\alpha_k^2\, c_i\, b_k}{\alpha_i^2\; c_k\, b_i}$ is valid for any designs *i* and *k*, including design 1. It is developed, above, to establish cross design equivalence for all designs other than 1, but the expansion of $\frac{\alpha_k^2\, c_i\, b_k}{\alpha_i^2\; c_k\, b_i}$ depends only on the definitions of α, c, b, which hold for any design. This fact will be very useful in the following:

8.4 Cross Design Equivalence, Design 1 With Any Other Design

The previous proof establishes cross design equivalence for any two designs other than design 1. The proof of cross design equivalence when one of the designs is design 1 requires a slightly different approach, as the structure of the optimality equations for design 1 differs from the structure for all other designs.

The OSTRE ratios within any design *i* permit all the optimum number of runs at any level, n_{ij}, to be expresses in terms of the optimum number of a single level. That is, there exist constants k_{ij} (which can be explicitly recovered from the optimality

equations), such that

$$n_{ij} = k_{ij}\, n_{i1}\ , \quad \text{all } j > 1$$

Given this fact, together with the previously established cross design equivalence between any levels in any designs other than design 1, cross design equivalence between any level in design 1 and any level in any other design can be proven by showing that it holds for any specific level in design 1 and any specific level in any other design. I show this equivalence for levels 1 in design 1 and 2. That is, I show that, between Single Optimization and OCBA+OSTRE:

$$\frac{n_{11}^{S}}{n_{21}^{S}} = \frac{n_{11}^{O}}{n_{21}^{O}}$$

Single Optimization

The ratio between n_{11} and n_{21} is derived from the optimality equation:

$$\frac{1}{b_{11}} \sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \frac{\partial \phi_k}{\partial n_{11}} = \frac{1}{b_{21}} e^{\frac{-\phi_2^2}{2}} \frac{\partial \phi_2}{\partial n_{21}}$$

$$\frac{1}{b_{11}} \sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \frac{(p_k - p_1)\, p_1^2 (1 - p_{11})}{2\, p_{11} \left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}} n_{11}^2} = \frac{1}{b_{21}} e^{\frac{-\phi_2^2}{2}} \frac{(p_2 - p_1)\, p_2^2 (1 - p_{21})}{2\, p_{21} \left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}} n_{21}^2}$$

$$\frac{p_1^2(1-p_{11})}{2b_{11}\,p_{11}\,n_{11}^2}\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\frac{(p_k-p_1)}{\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}} \;=\; \frac{1}{b_{21}}e^{\frac{-\phi_2^2}{2}}\frac{(p_2-p_1)\,p_2^2(1-p_{21})}{2\,p_{21}\left(\sigma_1^2+\sigma_2^2\right)^{\frac{3}{2}}\,n_{21}^2}$$

$$\frac{n_{11}^2}{n_{21}^2} \;=\; \frac{b_{21}\,p_1^2(1-p_{11})\,p_{21}\left(\sigma_1^2+\sigma_2^2\right)^{\frac{3}{2}}\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\frac{(p_k-p_1)}{\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}}}{b_{11}p_{11}(p_2-p_1)\,p_2^2(1-p_{21})e^{\frac{-\phi_2^2}{2}}}$$

<u>OCBA+OSTRE</u>

The ratio between n_{11} and n_{21} is derived from the optimality equation for N_1 and N_2

$$\frac{1}{b_1}\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\frac{\partial\phi_k}{\partial N_1} \;=\; \frac{1}{b_2}e^{\frac{-\phi_2^2}{2}}\frac{\partial\phi_2}{\partial N_2}$$

$$\frac{1}{b_1}\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\frac{(p_k-p_1)c_1}{2\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}N_1^2} \;=\; \frac{1}{b_2}e^{\frac{-\phi_2^2}{2}}\frac{(p_2-p_1)c_2}{2\left(\sigma_1^2+\sigma_2^2\right)^{\frac{3}{2}}N_2^2}$$

$$\frac{c_1}{2b_1N_1^2}\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\frac{(p_k-p_1)}{\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}} \;=\; \frac{1}{2b_2N_2^2}e^{\frac{-\phi_2^2}{2}}\frac{(p_2-p_1)c_2}{\left(\sigma_1^2+\sigma_2^2\right)^{\frac{3}{2}}}$$

$$b_2N_2^2c_1\left(\sigma_1^2+\sigma_2^2\right)^{\frac{3}{2}}\sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}}\frac{(p_k-p_1)}{\left(\sigma_1^2+\sigma_k^2\right)^{\frac{3}{2}}} \;=\; b_1N_1^2(p_2-p_1)c_2\,e^{\frac{-\phi_2^2}{2}}$$

$$\frac{N_1^2}{N_2^2} = \frac{b_2 c_1 \left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}} \sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \frac{\left(p_k - p_1\right)}{\left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}}}}{b_1 \left(p_2 - p_1\right) c_2 e^{\frac{-\phi_2^2}{2}}}$$

Substitute: $N_1 = \alpha_1 n_{11}, \quad N_2 = \alpha_2 n_{21}$

$$\frac{n_{11}^2}{n_{21}^2} = \frac{\alpha_2^2 b_2 c_1 \left(\sigma_1^2 + \sigma_2^2\right)^{\frac{3}{2}} \sum_{k=2}^{D} e^{\frac{-\phi_k^2}{2}} \frac{\left(p_k - p_1\right)}{\left(\sigma_1^2 + \sigma_k^2\right)^{\frac{3}{2}}}}{\alpha_1^2 b_1 \left(p_2 - p_1\right) c_2 e^{\frac{-\phi_2^2}{2}}}$$

8.5 Equivalence

The optimal ratios of n_{11} and n_{21} in Single Optimization and OCBA+OSTRE are thus the same iff, after cancellations:

$$\frac{b_{21} p_1^2 \left(1 - p_{11}\right) p_{21}}{b_{11} p_{11} p_2^2 \left(1 - p_{21}\right)} = \frac{\alpha_2^2 b_2 c_1}{\alpha_1^2 b_1 c_2}$$

This is exactly the equality of ratios shown, above, to hold between levels of different designs, for arbitrary designs *i* and *k*. Set *i* = 1, *k* = 2, and the equality

$$\frac{n_{11}^S}{n_{21}^S} = \frac{n_{11}^O}{n_{21}^O}$$

is established, implying equality between design 1, any level *j*, and any design *k*, level *m*:

$$\frac{n_{1j}^S}{n_{km}^S} = \frac{n_{1j}^O}{n_{km}^O}.$$

QED

9. OCBA+OSTRE vs Single Optimization: Implementation

Mathematical equivalence is one thing. Equivalence in practice is another. No one should be surprised if twin in theory is twain in practice. The implementations of Single Optimization and OCBA+OSTRE differ in ways that could produce some divergence in practice, beyond randomness.

How do OCBA+OSTRE and Single Optimization differ on the test model. I compare them for 6, 8, and 10 designs:

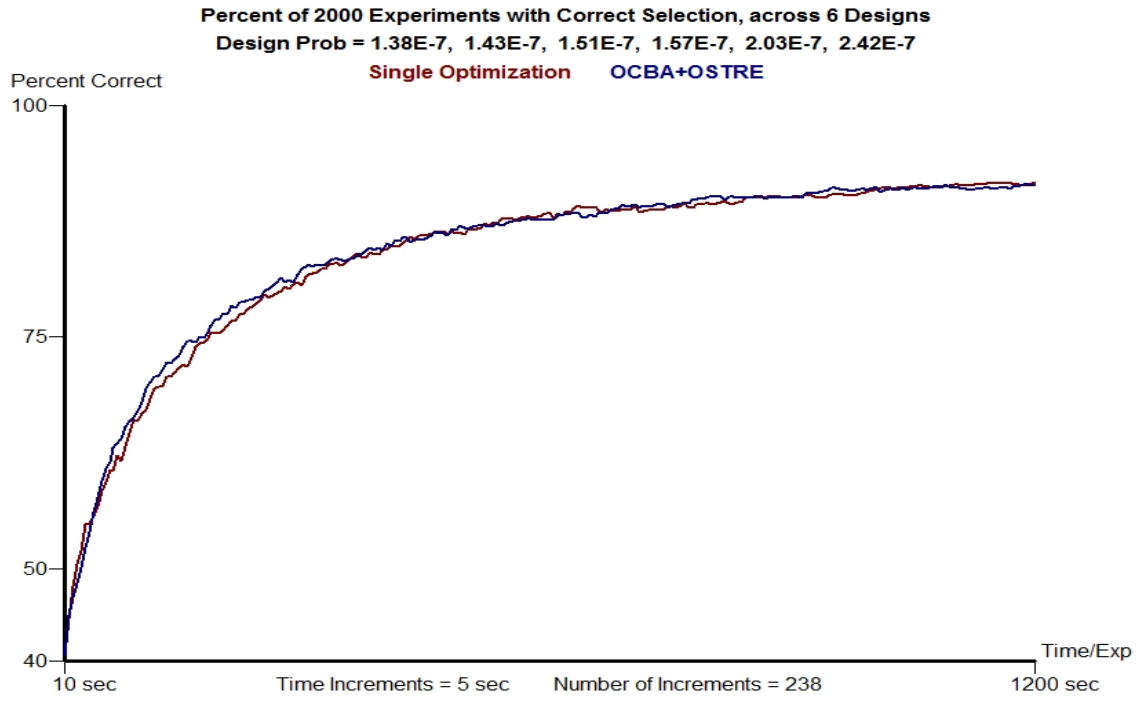

Figure 35. Single Optimization vs OCBA+OSTRE, 6 Designs

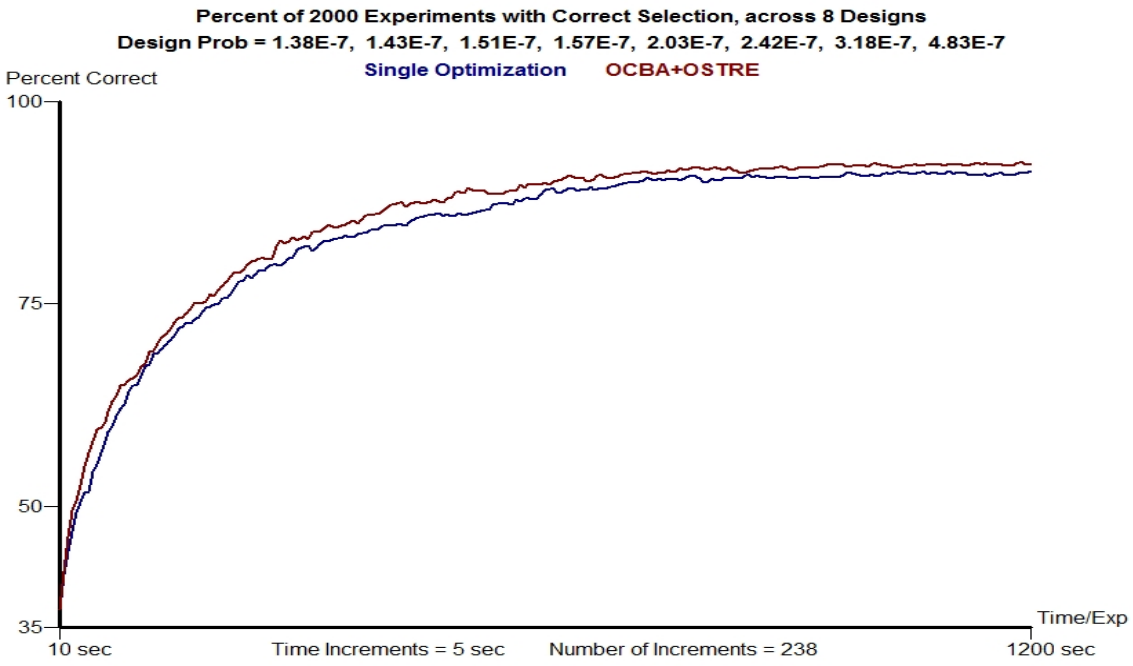

Figure 36. Single Optimization vs OCBA+OSTRE, 8 Designs

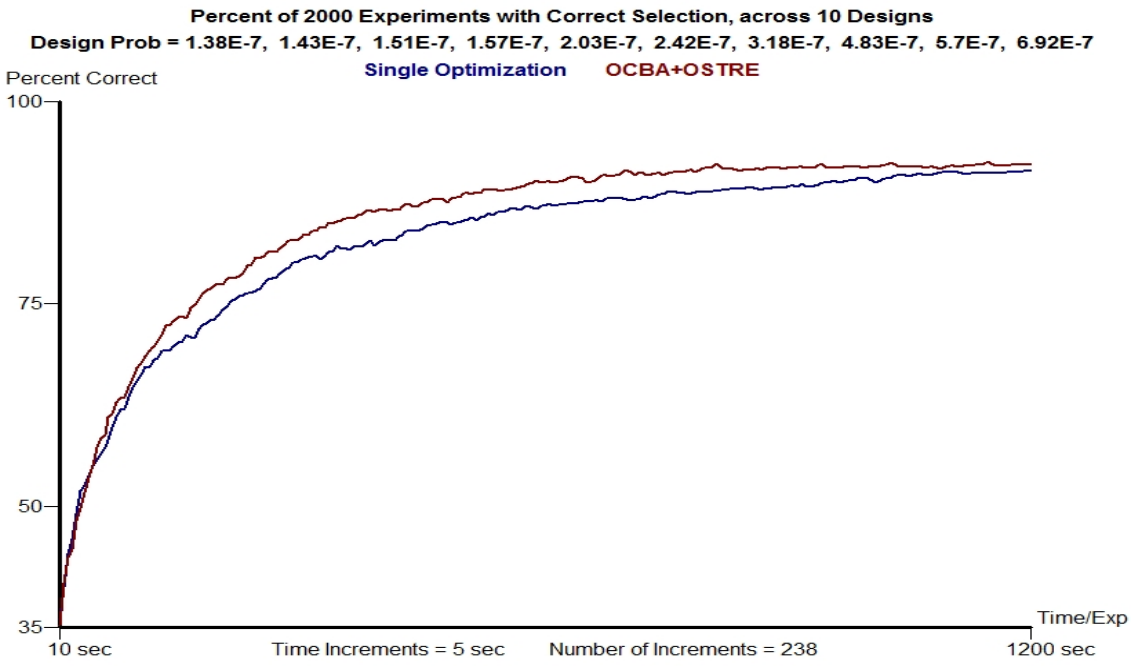

Figure 37. Single Optimization vs OCBA+OSTRE, 10 Designs

Over 6 designs there is no wiggle room between Single Optimization and OCBA+OSTRE. But some begins to appear, slightly, over 8 designs, and a bit more clearly over 10 designs. Why?

The most likely reason – a conjecture – lies in the implementation of the OSTRE step updates. The update phase of Single Optimization uses the *MaxDiff* method for selecting the next allocation. That is computationally expensive, as it must be applied to all levels of all designs. But, on the test model the computational overhead is not counted against the computational budget (only the pseudo simulation time counts) so that difference should make no difference in performance. *MaxDiff* is a heuristic, utilized to avoid having to find the optimal solution, by numerical methods, at each update.

Updating the OCBA step of OCBA+OSTRE also utilizes the *MaxDiff* method, but needs to apply it only across the designs, not, as with Single Optimization, across all levels of all designs. Updating in the OSTRE step does not require *MaxDiff*, or any similar method. It solves for the optimum solution directly (and cheaply). It is, to be sure, an approximate optimum, because it is based on the truncated version of the variance, but the truncation error is negligible for fairly large n_{ij}.

The conjecture is that the OSTRE step allocation, based on an explicit solution of the optimum at each update, is more efficient than *MaxDiff* at moving the algorithm closer to the optimum. This advantage should give OCBA+OSTRE an edge over Single Optimization in implementation, despite their mathematical equivalence in theory. This edge is borne out in tests on the test model.

10. Conclusion

This dissertation has reviewed four current simulation methods that could be used to select the best design from a set of designs, discussed their strengths and weaknesses when applied to the estimation of rare event probabilities, and tested them on a practical test model. Their strengths and weaknesses are relative to the computing budget constraint. All can work as well as desired, given a sufficiently large budget. Comparisons between them are based on the same, fairly tight, constraint. Their chief characteristics are:

Monte Carlo

- Equal allocation of computing effort among designs.
- Simple Monte Carlo estimation (hits/runs) for each design.
- Unbiased estimator, but unacceptably inefficient when applied to rare events. Would require a massively larger budget than other methods to achieve comparable results.

OCBA (Optimal Computing Budget Allocation)

- Optimizes allocation of computing effort among designs.
- Effective when selecting the best design in context of non-rare probabilities.

- Better than Monte Carlo when selecting best design in context of rare event probabilities, due to optimization of effort across designs, but still very inefficient, compared to other methods, due to reliance on simple Monte Carlo (hits/runs) for estimating each design probability.

Splitting

- No optimization of computing effort within or across designs. Assumes equal allocation within and across designs.
- Despite no optimization, can be very effective in estimating rare event probabilities.
- Its efficiency in estimating rare event probabilities is so much greater than that of OCBA (or Monte Carlo) that it is clearly superior to OCBA in selecting the best designs, when design probabilities are rare.

OSTRE (Optimal Splitting Technique for Rare Event Simulation)

- Optimizes allocation of effort among the levels of a design, given a budget constraint for that design.
- No optimization across designs; assumes equal allocation to designs.
- Should be better than Splitting in terms of estimating individual design probabilities, which should give it an edge on Splitting in terms of selecting the best design.

- Performance depends on the appropriateness of a key assumption: that the probability of a run hitting the next level is independent of the state from which the run launched.
- On the test model used in this research that assumption is rather weak.
- OSTRE nonetheless outperforms Splitting (though not by much) in selecting the best design, suggesting it may be quite robust even when its key assumption is not well supported.

The contribution of this dissertation is the development, in theory and implementation, of two new techniques for selecting the best design when all design probabilities are rare. They are:

Single Optimization

- Optimizes the allocation of effort simultaneously across all levels of all designs.
- Implementation requires a heuristic method for allocating the next increment of effort at each update.
- Computational overhead is substantial, and grows rapidly as number of designs and levels increases. Scales with (grows proportionately to) the product of number of designs and the number of levels per design.
- Outperforms the four techniques reviewed above

OCBA+OSTRE

- The major contribution of this research: a novel technique for combining OCBA and OSTRE in a two-step procedure that, like Single Optimization, optimizes across designs (the OCBA step) and, within each design, across levels (the OSTRE step).
- Mathematically equivalent to Single Optimization: attains, in theory, the same optimum allocations to all levels of all designs.
- More tractable in implementation: computational overhead somewhat less, and scales with the sum of number of levels and number of designs (whereas Single Optimization scales with their product).
- On the test model used in this research, OCBA+OSTRE is indistinguishable from Single Optimization across 6 designs, but is somewhat better across 8 and 10 designs. This suggests it enjoys an advantage in implementation, as designs increase, probably due to the fact that its allocation of effort in the OSTRE step is based on exact solutions, not on the heuristic method employed by Single Optimization.

The following graph displays the comparative performance of all the methods reviewed, and developed, in this research:

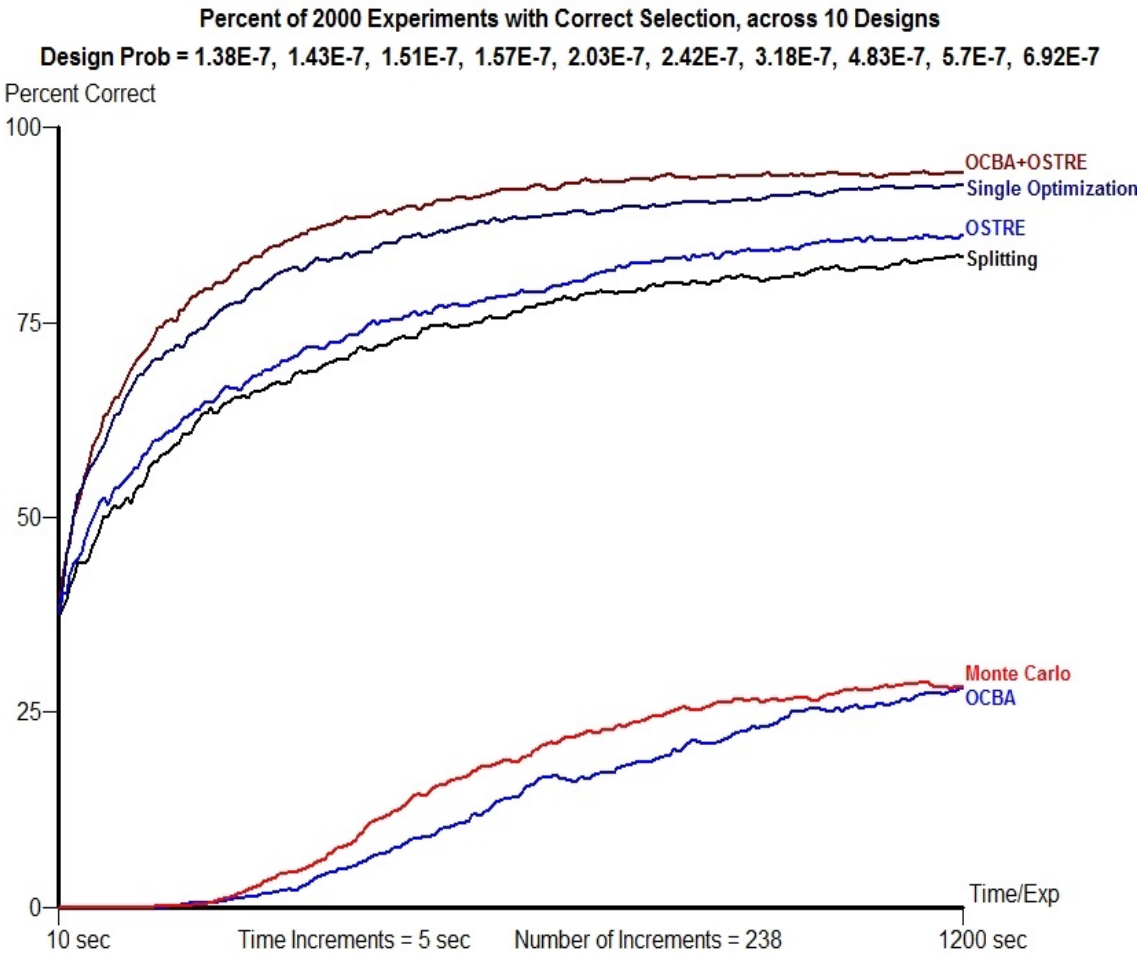

Figure 38. All Methods, 10 Designs

Neither Monte Carlo nor OCBA are competitive methods. Between them, MC seems to perform better, but that is due to the relatively small budget constraint. (See Figure 5, p 19, for an illustration of OCBA's superiority over MC over a longer budget.)

For a closer look at the four competitive designs:

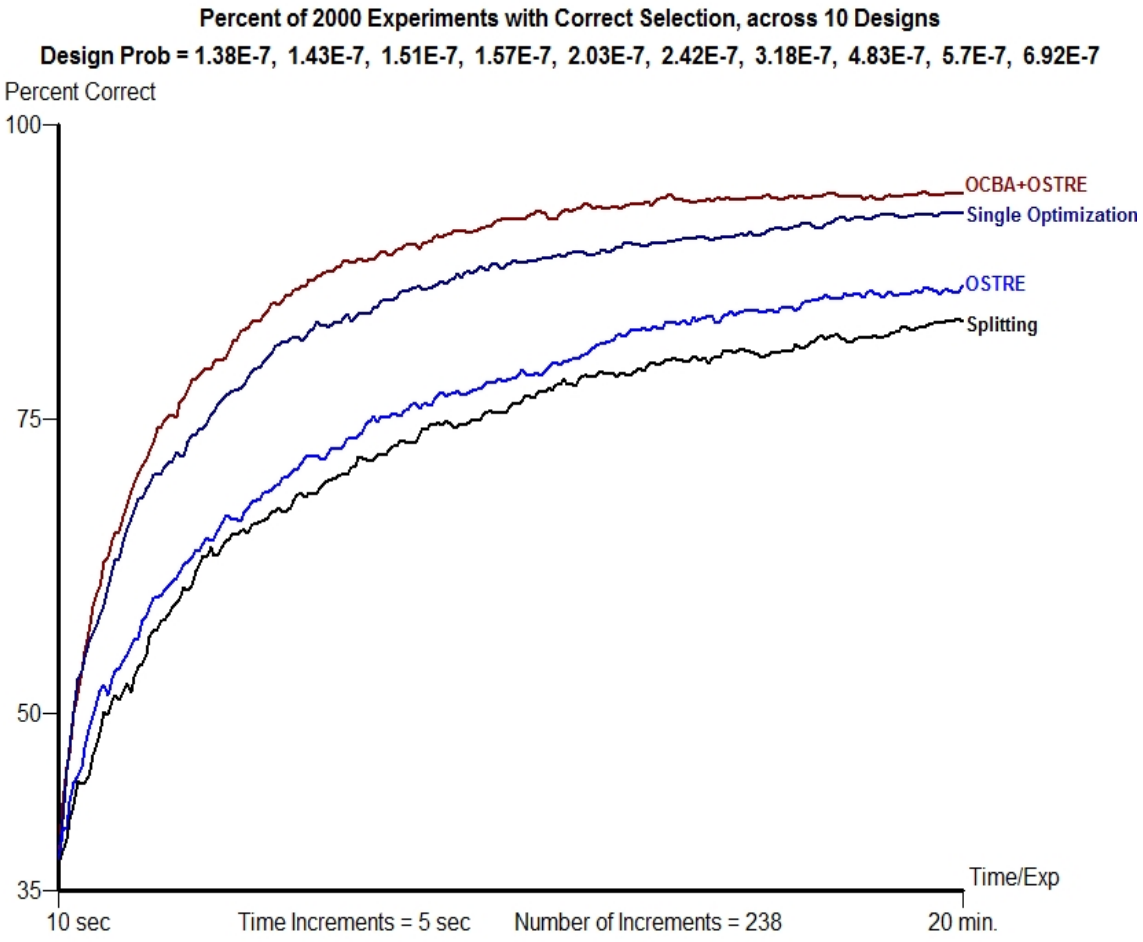

Figure 39. OCBA+OSTRE vs Single Optimization vs OSTRE vs Splitting

Prior to this research Splitting and OSTRE were state-of-the-art approaches to rare event estimation, even across several designs, within the framework of "fixed effort" splitting. Fixed splitting (see sec. 4.2, p. 54), not considered in this research, could offer alternative approaches. Single Optimization and OCBA+OSTRE – the contributions of

this dissertation – have moved the state-of-the-art for fixed effort splitting, applied to the selection of the best design in the context of rare events, a few notches higher.

Appendix 1: The Exact Probabilities of the Test Model

Because the test model, a two server tandem queue, is an absorbing Markov chain all the design and level probabilities can be calculated analytically. (See Winston, 2004, p. 945, for details.) In addition, the probability of hitting L_j from every possible starting state in L_{j-1} (the q_{jk} , for all *j*, *k*) can be calculated, analytically, along with the average number of transitions required to do so. These analytical solutions make it possible to do pseudo simulation on the test model.

The random variables driving the process -- the time to the next arrival, the times to completion of service in the servers – are exponentially distributed, with rates λ, μ_1, μ_2. For all designs λ = 1. μ_1 and/or μ_2 differ from design to design. p_{ik} is the level probability for design *i*, level *k*; p_i is the design probability for design *i*.

Table 4. Design Parameters

Design:	1	2	3	4	5	6	7	8	9	10
μ_1:	1.4	1.398	1.395	1.393	1.392	1.395	1.385	1.383	1.38	1.37
μ_2:	1.4	1.4	1.4	1.4	1.4	1.39	1.39	1.385	1.381	1.384

Table 5. Design Probabilities

Design	p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_i
1	0.037959229	0.015750363	0.013004592	0.017747597	1.37988924E-7
2	0.038218298	0.015911667	0.013130667	0.017905504	1.42975098E-7
3	0.038612548	0.01616228	0.01333082	0.018161203	1.51088892E-7
4	0.038879197	0.016335291	0.013471841	0.018344581	1.56956356E-7
5	0.04090109	0.017613131	0.014442516	0.019525255	2.03147276E-7
6	0.042311688	0.018538121	0.0151515415	0.020394392	2.42378247E-7
7	0.044401756	0.020007779	0.0163441024	0.021930997	3.18433297E-7
8	0.04.814084	0.022620291	0.0182909784	0.0242427018	4.8286958E-07
9	0.049776117	0.023786267	0.0191251617	0.0251902433	5.704069E-07
10	0.051506333	0.02512962	0.0201790462	0.0264993546	6.9212172E-07

Appendix 2: OCBA Literature

Chen et al. (1997) and Chen et al. (2000, 2010) developed the Optimal Computing Budge Allocation (OCBA) to (approximately) maximize the PCS of choosing the correct design, where "correct" means the design with the minimum mean (of some metric of interest).

OCBA was extended by Fu et al. to incorporate correlated sampling. Non-normal distributions were brought in by Glynn and Juneia (2004) and Fu et al. (2004). Other refinements include the introduction of regression models (Brantley et al., 2008); multiple objective functions (Lee at al. 2004, Chew et al. 2009, Lee et al., 2010); replacing the PCS with expected opportunity cost (Chick and Wu, 2005; He, Chick, and Chen 2007); and minimizing variances in lieu of maximizing the PCS (Trailovic and Pao, 2004). Chen et al. (2008) extend OCBA to the selection of an optimal subset in lieu of the single best design. Pujowidianto et al. (2009), and Yan et al. (2010) focus on selecting the best design under stochastic constraints or design complexity.

Appendix 3: Splitting Literature

A comprehensive survey of splitting in found in L'Ecuyer *et al*. (2009). Glasserman *et al*. (1998) analyses the role of the importance function in splitting, showing that a bad choice can yield poor results. Asymptotic optimality for the importance function can be obtained under conditions derived by Dean and Dupuis (2009).

Garvels (2000) provides a thorough analysis of fixed splitting. The RESTART method, presented in Villen-Altamirano (1994, 2006, 2007), is a prominent implementation of fixed splitting, containing methods for killing off unpromising simulation runs. Techniques to enhance the effectiveness of fixed splitting are developed by Lagnoux-Renaudie (2008).

Fixed effort splitting is addressed by L'Ecuyer *et al*. (2006). Glasserman *et* al. (1996) analyze bias and efficiency in simple implementations of splitting, and Glasserman *et al*. (1998) derive necessary conditions for splitting, in general, to by asymptotically optimal. Glasserman *et* al. (1999) apply a branching-process analysis to derive "the optimal degree of splitting at each threshold as the rarity of the event increases". Garvels and Kroese (1998) derive optimal allocations under fixed effort splitting, under the assumption equal costs for all simulation runs. Shortle, Chen *et al*. generalize these results, including variable costs.

Appendix 4: Truncation of the Variance in OSTRE

Claim: *The approximation of the OSTRE design variance justified by the assumption that* $n_j >> (1-p_j)/p_j$ *is equivalent to truncating that variance at its higher order terms.*

Proof:

$\hat{p}$ = design estimator; p_j = level probability for L_j; b_j = average time of a run in L_j.

For *L* levels, the design variance can be expressed as:

$$Var[\hat{p}] = \left[p_1^2 + \frac{p_1(1-p_1)}{n_1}\right] \cdots \left[p_L^2 + \frac{p_L(1-p_L)}{n_L}\right] - p_1^2 \cdots p_L^2$$

Define: $a_j \equiv \frac{b_j(1-p_j)}{p_j}$, $s_j \equiv b_j n_j$

Express the variance in these terms:

$$Var[\hat{p}] = p_1^2\left(1+\frac{a_1}{s_1}\right) \cdots p_L^2\left(1+\frac{a_L}{s_L}\right) - p_1^2 \cdots p_L^2$$

$$\text{var}[\hat{p}] = \left(p_1^2 \cdots p_L^2\right)\left(1+\frac{a_1}{s_1}\right) \cdots \left(1+\frac{a_L}{s_L}\right) - p_1^2 \cdots p_L^2$$

$$Var\left[\hat{p}\right] = \left(p_1^2 \cdots p_m^2\right)\left[1+\frac{a_1}{s_1}+\cdots+\frac{a_L}{s_L} + \sum\left(\text{higher order terms}\right)\right] - p_1^2 \cdots p_L^2$$

The higher order terms are of the form $\frac{a_i}{s_i}\cdots\frac{a_j}{s_j}$, involving products of the form

$\frac{1}{n_i \cdots n_j}$, and may be regarded as negligible, compared to the first order terms.

Discarding them:

$$\text{var}\left[\hat{p}\right] \approx \left(p_1^2 \cdots p_L^2\right)\left(\frac{a_1}{s_1}+\cdots+\frac{a_L}{s_L}\right) = p^2\left(\frac{a_1}{s_1}+\cdots+\frac{a_L}{s_L}\right)$$

The minimization problem is:

$$\underset{s_j}{Min}\, Var\left[\hat{p}\right] \quad \text{s/t} \quad \sum_j b_j n_j = \sum_j s_j = T$$

The Lagrangian is

$$F = p^2\left(\frac{a_1}{s_1}+\cdots+\frac{a_L}{s_L}\right) + \lambda\left(\sum_j s_j - T\right)$$

$$\frac{\partial F}{\partial s_j} = \frac{-p^2 a_j}{s_j^2} + \lambda = 0 \qquad \text{which implies:}$$

$$\frac{a_1}{s_1^2} = \cdots = \frac{a_L}{s_L^2}, \qquad \text{or:}$$

$$\frac{n_1^2 b_1 p_1}{1-p_1} = \cdots = \frac{n_L^2 b_L p_L}{1-p_L},$$

which are the OSTRE optimality equations under the $n_j >> \left(1-p_j\right)/p_j$ justification.

Appendix 5: PCS = 1 Requires Infinite Runs at All Levels

The optimization problem is constrained minimization:

$$\underset{n_{ij}}{Min} \sum_{i=2}^{D} \int_{\phi_i\left(n_{1j},n_{ij}\right)}^{\infty} e^{-\frac{x^2}{2}} dx \quad \text{s/t} \quad \sum_{j=1}^{L}\sum_{i=1}^{D} b_{ij}\, n_{ij} = T$$

The PCS is maximized at 1 iff the objective function

$$\sum_{i=2}^{D} \int_{\phi_i\left(n_{1j},n_{ij}\right)}^{\infty} e^{-\frac{x^2}{2}} dx = 0.$$

This is attained iff all $\phi_i\left(n_{1j}, n_{ij}\right) \to \infty$.

For each i, $\quad \phi_i = \dfrac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}} \to \infty$ iff $\sigma_1 \to 0$ *and* $\sigma_i \to 0$

For each i (including 1), the truncated variance used in Single Optimization is:

$$\sigma_i^2 = p_i^2 \left[\frac{1-p_{i1}}{p_{i1}\, n_{i1}} + \cdots + \frac{1-p_{iL}}{p_{iL}\, n_{iL}} \right], \qquad L = \text{number of levels}$$

Clearly, $\sigma_i \to 0$ iff $n_{ij} \to \infty$ *for all* j.

Thus, $\quad \sum_{i=2}^{D} \int_{\phi_i\left(n_{1j},n_{ij}\right)}^{\infty} e^{-\frac{x^2}{2}} dx \to 0 \;\Rightarrow\; all\ n_{ij} \to \infty \;\Rightarrow\; T \to \infty$

Appendix 6: Convergence of Higher Order Terms in the Design Estimators

The expansion of the product of the level estimator may be expressed as:

$$\prod_{j=1}^{L} \hat{p}_{ij} \;=\; \prod_{j=1}^{L} p_{ij} \;+\; \sum_{j=1}^{L} c_{ij} \sqrt{\frac{p_{ij}\left(1-p_{ij}\right)}{n_{ij}}}\, Z_{ij} \;+\; \sum_{m} \frac{k_m X_m}{\sqrt{n_{i j_1} \cdots n_{i j_q}}}$$

where:

c_{ij}, k_m = constants

X_m = the product of two or more Z_{ij} from $\{Z_1, ..., Z_L\}$

$n_{i j_1} \cdots n_{i j_q}$ = product of q n_{ij} terms from $\{n_{i1}, ..., n_{iL}\}$, $q \geq 2$

The last term, $\sum_{m} \frac{k_m X_m}{\sqrt{n_{i j_1} \cdots n_{i j_q}}}$, converges to 0, in probability, as <u>all</u> the n_{ij} increase without limit. The random variables in this summation are of the same form: a random variable X_m scaled by a non-random multiple, $\frac{k_m}{\sqrt{n_{i j_1} \cdots n_{i j_q}}}$. To show that the sum of these random variables converges to 0 it suffices to show that any given $\frac{k_m X_m}{\sqrt{n_{i j_1} \cdots n_{i j_q}}}$ so converges. (If each of a sum of random variables converges to 0, so does their sum.)

The argument depends on <u>all</u> the n_{ij} increasing as *T*, the computing budget, increases. The goal is to maximize the PCS. It is assumed that any increase in *T* is devoted to that

purpose. That is, given an increase in *T*, all the n_{ij} adjust to their new optimum values, those that maximize the PCS subject to the new *T*. It should be obvious that driving the PCS to certainty ($PCS \to 1$) forces all the n_{ij} to increase without limit. Increasing any n_{ij} will always improve the PCS, so the PCS cannot be 1 if any n_{ij} is finite. (See Appendix 5 for a proof confirming the obvious.) Thus:

$$PCS \to 1 \;\Rightarrow\; all\ n_{ij} \to \infty \;\Rightarrow\; T \to \infty\ .$$

All that is needed to prove convergence, in this case, is the requirement that, as *T* increases, all the n_{ij} change to their new optimum values. Then $T \to \infty \Rightarrow all\ n_{ij} \to \infty$, and we have, for any term $\dfrac{k_m X_m}{\sqrt{n_{i j_1} \cdots n_{i j_q}}}$, convergence in probability. For any $\varepsilon > 0$:

$$\lim_{T \to \infty} P\left[-\varepsilon < \frac{k_m X_m}{\sqrt{n_{i j_1} \cdots n_{i j_q}}} < \varepsilon \right] =$$

$$\lim_{all\ n_{ij} \to \infty} P\left[\frac{-\varepsilon \sqrt{n_{i j_1} \cdots n_{i j_q}}}{k_m} < X_m < \frac{\varepsilon \sqrt{n_{i j_1} \cdots n_{i j_q}}}{k_m} \right] =$$

$$P\left[-\infty < X < \infty \right] = 1$$

Appendix 7: Single Optimization Optimality Equations for 4 Designs, 3 Levels

$$\frac{1}{b_{11}}\sum_{i=2}^{4} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{11}} = \frac{1}{b_{12}}\sum_{i=2}^{4} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{12}} = \frac{1}{b_{13}}\sum_{i=2}^{4} e^{\frac{-\phi_i^2}{2}}\frac{\partial \phi_i}{\partial n_{13}} =$$

$$\frac{1}{b_{21}} e^{\frac{-\phi_2^2}{2}}\frac{\partial \phi_2}{\partial n_{21}} = \frac{1}{b_{22}} e^{\frac{-\phi_2^2}{2}}\frac{\partial \phi_2}{\partial n_{22}} = \frac{1}{b_{23}} e^{\frac{-\phi_2^2}{2}}\frac{\partial \phi_2}{\partial n_{23}} =$$

$$\frac{1}{b_{31}} e^{\frac{-\phi_3^2}{2}}\frac{\partial \phi_3}{\partial n_{31}} = \frac{1}{b_{32}} e^{\frac{-\phi_3^2}{2}}\frac{\partial \phi_3}{\partial n_{32}} = \frac{1}{b_{33}} e^{\frac{-\phi_3^2}{2}}\frac{\partial \phi_3}{\partial n_{33}} =$$

$$\frac{1}{b_{41}} e^{\frac{-\phi_4^2}{2}}\frac{\partial \phi_4}{\partial n_{41}} = \frac{1}{b_{42}} e^{\frac{-\phi_4^2}{2}}\frac{\partial \phi_4}{\partial n_{42}} = \frac{1}{b_{43}} e^{\frac{-\phi_4^2}{2}}\frac{\partial \phi_4}{\partial n_{43}}$$

$$\sum_{j=1}^{3}\sum_{i=1}^{4} b_{ij}\, n_{ij} = T$$

Appendix 8: In Single Optimization Each Optimality Term Converges to 0

In Single Optimization each optimality term $\to$ *0 as* $PCS \to 1$:

Appendix 6 showed that $PCS \to 1 \Rightarrow$ *all* $n_{ij} \to \infty \Rightarrow T \to \infty$. I extend that to:

$PCS \to 1 \Rightarrow$ *all* $n_{ij} \to \infty \Rightarrow$ *Each Optimality Term* $\to 0$:

Ignoring the constants $1/b_{ij}$ each term in the optimality conditions (see p….) takes the form (or is the sum of such forms):

$$e^{\frac{-\phi_i^2}{2}} \frac{\partial \phi_i}{\partial n_{m\,j}} \qquad i = 2,...,D \quad m = 1 \text{ or } i \quad j = 1,...,L$$

Consider the first term, $e^{\frac{-\phi_i^2}{2}}$. The function ϕ_i is defined as

$$\phi_i = \frac{p_i - p_1}{\sqrt{\sigma_1^2 + \sigma_i^2}}, \quad i = 2,...,D$$

with

$$\sigma_i^2 = p_i^2 \left[\frac{1 - p_{i\,1}}{p_{i\,1} n_{i\,1}} + \cdots + \frac{1 - p_{i\,L}}{p_{i\,L} n_{i\,L}} \right], \quad i = 1,...,D.$$

Clearly:

$$All\ n_{ij} \to \infty \Rightarrow \sigma_i^2 \to 0 \Rightarrow \phi_i \to \infty \Rightarrow e^{\frac{-\phi_i^2}{2}} \to 0\ .$$

Now consider the second term:

$$\frac{\partial \phi_i}{\partial n_{mj}} = \frac{\left(p_i - p_1\right) p_m^2 \left(1 - p_{mj}\right)}{2\, p_{mj} \left(\sigma_1^2 + \sigma_i^2\right)^{\frac{3}{2}} n_{mj}^2} \qquad i = 2,...,D \quad j = 1,...L \quad m = 1 \text{ or } i$$

Also for this term:

$$All\ n_{ij} \to \infty \ \Rightarrow \ \frac{\partial \phi_i}{\partial n_{mj}} \to 0$$

Thus:

$$PCS \to 1 \Rightarrow \ all\ n_{ij} \to \infty \ \Rightarrow \ e^{\frac{-\phi_i^2}{2}} \frac{\partial \phi_i}{\partial n_{m\,j}} \to 0$$

References

Booth, Thomas. (2009) Particle transport application, in Rubino, G., Tuffin, B. (eds), *Rare Event Simulation Using Monte Carlo Methods*, Wiley, Chichester, UK, pp 215-242.

Brantley, M. W., L. H. Lee, C. H. Chen, and A. Chen, "Optimal Sampling in Design of Experiment for Simulation-based Stochastic Optimization," *Proceedings of 2008 IEEE Conference on Automation Science and Engineering*, pp. 388-393, Washington, DC, August 2008.

Bucklew, James Antonio. (2004) *Introduction to Rare Event Simulation,* Springer-Verlag, New York, NY.

Chen, H. C., C. H. Chen, L. Dai, and E. Yücesan. (1997). New development of optimal computing budget allocation for discrete event simulation. In *Proceedings of the 1997 Winter Simulation Conference*, 334–341. Piscataway, NJ: IEEE.

Chen, C.H., He, D., and Yucesan, E. (2003) Better-Than-Optimal Simulation Run Allocation?, in *Proceedings of the 2003 Winter Simulation Conference*, IEEE Press, Piscataway, NJ.

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

Chen, C. H., D. He, M. C. Fu, and L. H. Lee. 2008. Efficient simulation budget allocation for selecting an optimal subset. *INFORMS Journal on Computing*. 20 (4): 579-595.

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

Chew, E. P., Lee, L.H., Teng, S.Y., and Koh, C.H. 2009. Differentiated service inventory optimization using nested partitions and MOCBA. *Computers and Operations*

Research 36(5): 1703-1710.

Chick, S. E., and Y.-Z. Wu. 2005. Selection procedures with frequentist expected opportunity cost. *Operations Research* 53 (5): 867–878.

Crain, B., Chen, C.H. and Shortle, J.F. (2011) Combining Simulation Allocation and Optimal Splitting for Rare-Event Simulation Optimization, in *Proceedings of the 2011 Winter Simulation Conference*, IEEE Press, Piscataway, NJ.

Fu, M. C., J. Q. Hu, C. H. Chen, and X. Xiong. 2004. Optimal computing budget allocation under correlated sampling. In *Proceedings of the 2004 Winter Simulation Conference*, 595–603. Piscataway, NJ: IEEE.

Garvels, M. (2000) The splitting method in rare event simulation. Ph.D. thesis, University of Twente, The Netherlands.

Garvels, Marnix J.J. and Kroese, Dirk K. (1998) A Comparison of RESTART Implementations, in *Proceedings of the 1998 Winter Simulation Conference*, IEEE Press, Piscataway, NJ, pp 601-608.

Ghahramani, Saeed. (2005) *Fundamentals of Probability with Stochastic Processes*, third edition. Pearson Prentice Hall, Upper Saddle River, NJ.

Glasserman, P., Heidelberger, P. Shahabuddin, P. and Zajic, T. (1996) Splitting for Rare Event Simulation: Analysis of Simple Casesilevel splitting, in *Proceedings of the 1996 Winter Simulation Conference*, IEEE Press, Piscataway, NJ, pp 302-308.

Glasserman, P., Heidelberger, P. Shahabuddin, P. and Zajic, T. (1998) A large deviations perspective on the efficiency of multilevel splitting. *IEEE Transactions on Automatic Control*, **43**, 1666-1679.

Glasserman, P., Heidelberger, P. Shahabuddin, P. and Zajic, T. (1999) Multilevel Splitting for Estimating Rare Event Probabilities. *Operations Research*. Vol. 47, No. 4, 585-600.

Glynn, P.W. (1994) Efficiency improvement technique. *Annals of Operations Research*, 53, 175-197.

Glynn, P., and S. Juneja (2004). A large deviations perspective on ordinal optimization, in *Proceedings of the 2004 Winter Simulation Conference*, 577–585. Piscataway, NJ: IEEE.

He, D., S. E. Chick, and C. H. Chen. 2007. The opportunity cost and OCBA selection procedures in ordinal optimization. *IEEE Transactions on Systems, Man, and Cybernetics–Part C* 37:951–961.

Heidelberger, P. (1993) Fast simulation of rare events in queueing and reliability models, in Donatiello, L. and Nelson, R. (eds.) *Performance Evaluation of Computer and Communication Systems*, Springer Verlag, Berlin, pp. 165-202.

Hogg, R.V., McKean, J.W. and Craig, A.T. (2005) *Introduction to Mathematical Statistics*, sixth edition. Pearson Prentice Hall, Upper Saddle River, NJ.

Lagnoux-Renaudie, A. (2008) Effective branching splitting method under cost constraint. *Stochastic Processes and Their Applications*. **118**, 1820-1851.

L'Ecuyer, P., Demers, V. and Tuffin, B. (2006) Splitting for rare-event simulation, in *Proceedings of the 2006 Winter Simulation Conference*, IEEE Press, Piscataway, NJ, pp 137-148.

L'Ecuyer, P., Le Gland, F., Lezaud, P., and Tuffin, B. (2009) Splitting techniques, in Rubino, G., Tuffin, B. (eds), *Rare Event Simulation Using Monte Carlo Methods*, Wiley, Chichester, UK, pp 39-62.

Lee, L. H., E. P. Chew, S. Y. Teng, and D. Goldsman. 2004. Optimal computing budget allocation for multi-objective simulation models. In *Proceedings of the 2004 Winter Simulation Conference*, 586–594. Piscataway, NJ: IEEE.

Lee, L. H., Chew, E. P., Teng, S. Y., and Goldsman, D. 2010. Finding the non-dominated Pareto set for multi-objective simulation models. *IIE Transactions* 42:656-674.

Hammersley J.M. and Handscomb, D.C. *Monte Carlo Methods*. Methuen, London, 1964.

Kahn, H. and Harris, T.E. *Estimation of Particle Transmission by Random Sampling*. National Bureau of Standards Applied Mathematics Series, 1951.

Pujowidianto, N. A., L. H. Lee, C. H. Chen, C. M. Yep, "Optimal Computing Budget Allocation For Constrained Optimization," *Proceedings of 2009 Winter Simulation Conference*, pp. 584-589, Austin, TX, December 2009.

Rubino, G. and Triffin, B. (eds) (2009) *Rare Event Simulation Using Monte Carlo Methods*, Wiley, Chichester, U.K.

Shortle, John F., L'Ecuyer, P. (2010) Introduction to Rare-Event Simulation, in James J. Cochran (ed.), *Wiley Encyclopedia of Operations Research and Management Science*, John Wiley & Sons.

Shortle, John F., Chen, Chun-Hung, *et al*. (2012) Optimal splitting for rare-event simulation. *IIE Transactions*, **44**, 352-367.

Trailovic, L., and L. Y. Pao. 2004. Computing budget allocation for efficient ranking and selection of variances with application to target tracking algorithms. *IEEE Transactions on Automatic Control* 49: 58–67.

Villen-Altamirano, M. and Villen-Altamirano, J. (1994) RESTART: A straightforward method for fast simulation of rare events, in *Proceedings of the 1994 Winter Simulation Conference*, IEEE Press, Piscataway, NJ, pp. 282-289.

Villen-Altamirano, M. and Villen-Altamirano, J. (2006) On the efficiency of RESTART for multidimensional state systems. *ACM Transactions on Modeling and Computer Simulation*, **16**, 251-279.

Villen-Altamirano, Jose. (2007) Rare event RESTART simulation of two-stage networks. *European Journal of Operational Research*, **179**, 148-159.

Yan, S., E. Zhou, and C. H. Chen, "Efficient Simulation Budget Allocation for Selecting the Best Set of Simplest Good Enough Designs," *Proceedings of 2010 Winter Simulation Conference*, pp. 1152-1159, Baltimore, MD, December 2010.

Winston, Wayne L. (2004) Operations Research Applications ad Algorithms, fourth edition. Brooks/Cole, Belmont, CA.

Curriculum Vitae

Ben W. Crain worked for many years in the House of Representatives of the U.S. Congress, earned several Masters Degrees from Johns Hopkins University, and finally earned a PhD from George Mason University, upon the completion of which he was unemployed and broke.