Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

DYNAMIC NEIGHBORHOOD BOUNDING FOR MONTE CARLO SIMULATION

Jason S. Glazier

Department of Computer Science
Columbia University
New York, NY 10027, USA

Salvatore J. Stolfo

Department of Computer Science
Columbia University
New York, NY 10027, USA

ABSTRACT

A general variance reduction technique is presented that allows the incorporation of 'expert knowledge' to speed up some Monte Carlo Simulations. A user supplied bounding function is introduced that is used to reduce the number of sampled vector evaluations, thus potentially leading to a speed up of a Monte Carlo Simulation. We call the technique Dynamic Neighborhood Bounding. A detailed case study is provided to demonstrate the effectiveness of the approach.

1 INTRODUCTION

Monte Carlo Simulation (MCS) is an important numerical analysis tool in many areas of research, and is a popular application of parallel computing. In this paper, we introduce a variance reduction technique (VRT) that is applicable in a wide range of MCS problems.

The MCS problem can be stated as follows. Given a function $f(.)$ and a probability distribution Ω, generate randomly distributed vectors $v \in \Omega$, and execute function evaluations $f(v)$. The vectors generated are considered to be statistically representative of real life scenarios. The $f(v)$ calculation is referred to as a *primary estimate calculation (PEC).* After the scenarios and evaluations are completed, statistical tests are performed to estimate means, variances, n^{th} percentile elements, and confidence levels. These calculations using the PECs are referred to as a *secondary estimate calculation (SEC).* We define a *candidate vector,* as a vector $v \in \Omega$ that has been generated, but has not yet been evaluated (i.e. a candidate for evaluation).

We propose a VRT called Dynamic Neighborhood Bounding (DNB). A VRT attempts to reduce the variance of the output of a MCS. By reducing the output variance, either fewer samples may be generated and evaluated to achieve the same final accuracy, or a higher level of accuracy can be achieved with the same number of samples. There are many surveys on VRTs Cheng (1986), Nelson (1985) ,Nelson (1987), Wilson (1984a), and Wilson (1984b). The most widely known VRTs are control variates, stratified sampling, common random numbers, and importance sampling. The similarities and differences between DNB and both control variates and importance sampling is discussed in section 3. We will explain how the other VRTs can be used *in conjunction with* DNB.

DNB reduces the variance by effectively 'increasing' the number of samples, *without* increasing the number of function evaluations. In standard VRTs, all candidate vectors, once generated are evaluated. DNB may decide not to evaluate a candidate vector. The function evaluation of all *unevaluated* candidate vectors are *estimated,* and are used to increase the sample count. Every time we estimate an unevaluated candidate vector, the cost of one function evaluation is saved, potentially speeding up the overall MCS. Thus, in problems where the function evaluations are very expensive, a substantial savings in overall execution time is possible. However, the overhead of applying DNB is considerable, and must be amortized by a reduction in the number of function evaluations.

We distinguish between two types of *experts,* the *domain expert* seeks a solution of some hard problem by MCS. The other, the *simulation expert* , is knowledgeable about MCS and computational complexity, but may know little about the domain application. DNB relies on a weak form of domain knowledge to provide a speed up. Using a domain expert's knowledge to speed up MCSs is difficult in general and is usually done in an ad hoc fashion by clever programmers. Few generic mechanisms exist. The difficulty is that one must intimately understand both the domain of interest as well as sophisticated knowledge of computational complexity in order to make specific time saving suggestions and formulate heuristics. What is needed is a general framework to guide a process whereby the computer expert can ask the domain expert for *specific* knowledge to produce a faster MCS.

DNB provides a framework for such a precise querying of experts with the notion of a *bounding function.* The bounding function is provided by an expert in the domain field, and is the only piece of expert knowledge needed to implement DNB. Given a vector v_1 where $f(v_1)$ is *known,* and a vector v_2 where $f(v_2)$ is *unknown,* the bounding function $B(f(v_1), v_1, v_2)$ bounds the absolute value of the difference, $f(v_2) - f(v_1)$:

$$| f(v_2) - f(v_1) | \leq B(f(v_1), v_1, v_2)$$

The bounding function should not be confused with Euclidean distance. The bounding function may be any function able to overestimate the absolute deviation of $f(v_2)$ from $f(v_1)$, for some arbitrary prior sample v_1. For our case study presented in section 5, a non-Euclidean domain specific B() is used to demonstrate the principle concepts. It is also important that B() be relatively un-biased.

DNB saves all evaluated vectors in a set, P. When a candidate vector is generated and is ready to be evaluated, we first check if it is in a neighborhood of some $p \in P$, where $f(p)$ is known. $f(p)$ is known because it was previously evaluated in a prior iteration of the algorithm. If the candidate vector is, we do not evaluate it. Although we did not evaluate it, by knowing that it is in a close neighborhood of p, we can prove a reduction in variance. If the cost of calculating $f(.)$ is high and the cost of calculating whether a candidate vector is a close neighbor of p is cheap relative to its cost of $f(.)$, we have a potential speed up, if enough candidate vector evaluations are saved.

We define an ϵ-neighborhood for some $p \in P$ to be:

$$N_{\epsilon,B}(p) = \{v \in \Omega \mid B(f(p), p, v) \leq \epsilon\}$$

The vector v is in the neighborhood of p iff $v \in N_{\epsilon,B}(p)$. Notice that this is *not* $d(p, v) \leq \epsilon$ for some distance metric d. We discuss the effect of ϵ on error later.

When a candidate vector is not evaluated because it is a close neighbor of some previously evaluated point, it is called a *hit.* This represents a savings of one function evaluation at the cost of a neighborhood check on the set of all evaluated vectors. We refer to the number of hits divided by the total number of candidate vectors as the *hit ratio.*

The set of all neighborhoods, for all vectors in P is defined as:

$$N_{\epsilon,B}(P) = \{v \in \Omega \mid \exists p \in P,\ s.t.\ v \in N_{\epsilon,B}(p)\}$$

A neighborhood defines a multi-dimensional volume. If the vectors were uniformly random, then the proportion of volume taken up by neighborhoods for large epsilon should be roughly equivalent to the hit ratio:

$$h_{\epsilon,B} \approx \frac{| N_{\epsilon,B}(P) |}{| \Omega |}$$

where '||' denotes a set measure. Note that the problem of calculating the mean, with uniformly distributed candidate vectors is the problem of calculating the multi-dimensional integral of $f(.)$. For this case $B(.)$ can be chosen to be based on the derivative.

The remainder of the paper is organized as follows. In section 2 we present the DNB algorithm which sets the stage for our subsequent discussions. Section 3 compares DNB to other well known and similar VRTs, control variates and importance sampling. Section 4 discusses the class of MCS problems to which DNB may be applied. Section 5 is a case study where DNB was applied to a financial simulation to determine pre-settlement risk. Section 6 concludes the paper with a discussion of future work.

2 THE ALGORITHM

The MCS algorithm with DNB is as follows.

```
1)   Input: $f(.), B(.), \epsilon, \Omega$
2)   initialize $P$ and $Q = \emptyset$
3)   while ( more scenarios are needed )
4)       generate new scenario $v \in \Omega$
5)       find one $(p, f(p)) \in P$ s.t. $v \in N_{\epsilon,B}(P)$
6)           $\tilde{f}(v) = f(p)$
7)           add $(v, \tilde{f}(v))$ to $Q$
8)       else
9)           evaluate $f(v)$
10)          add $(v, f(v))$ to $P$
11)  end while
12)  Upon completion, we next compute a number
         of statistics as follows:
```

$$n = | P | \quad \text{and} \quad m = | Q |$$

$$\text{mean: } \bar{f}_* = \frac{1}{n+m}\left[\sum_{p \in P} f(p) + \sum_{v \in Q} \tilde{f}(v)\right]$$

After the initialization in line 2, the first operation of the main loop is to generate a new scenario in line 4. A new scenario will be referred to as a candidate vector. For each candidate vector v, the algorithm searches the set P for a neighborhood that contains v in line 5. If a neighborhood $N_{\epsilon,B}(p)$ is found in line 5, then the evaluated point is saved in the set Q in line 7 as a pair with its corresponding $\tilde{f}(v)$ value, calculated and previously stored by a prior iteration of the algorithm at step 10. If v belongs to no neighborhood of P, then $f(v)$ must be evaluated in line 9.

A candidate vector v will either become an evaluated vector in P, or an unevaluated vector stored with its corresponding $\tilde{f}$ value in Q. After enough scenarios have been evaluated, we run statistics on the output.

In the algorithm above we allowed v to be associated with at most one neighborhood $N_{\epsilon,B}(P)$. As we will see later in the case study, it is significantly more accurate to let v fall into many neighborhoods, i.e. we shall allow multiple hits, and then let our estimate of $f(v)$ be the average function value over all the neighborhoods. In order to allow multiple hits, we replace lines 5 and 6 in the algorithm with:

```
5)      find $R = \{f(p) \mid (p, f(p)) \in P \text{ s.t. } v \in N_{\epsilon,B}(P)\}$
6)           $\tilde{f}(v) =$ average of set $R$
```

Note, a variety of indexing techniques can substantially reduce the time complexity of line 5 for multiple hits. Some of the issues critical to the algorithm's success are:

- How should ϵ be set?
- What is the confidence interval?
- What is the error induced by DNB?
- What is the increase in efficiency?
- What algorithms are efficient for locating neighborhoods?

We address these issues in the following sections.

3 COMPARISON TO OTHER VRTs

It is important to contrast DNB to two other important VRTs, importance sampling and control variates. Like DNB, both of these two techniques provide a means of exploiting domain knowledge in a MCS. Control variates provides a speed up if the expert can produce an integrable function $g()$ that approximates $f()$. It is not essential for $g()$ to be a good numeric approximation of $f()$, as long as $g()$ approximates the volatility of $f()$. By knowing $g()$ we then compute $f(v) - g(v)$, and add the known expectation of $g()$. The closer $g()$ approximates $f()$, the more $f(v) - g(v)$ will tend to zero. The $B()$ used in DNB differs from $g()$ because $B()$ is an overestimate of the maximum absolute difference of the $f()$ values of any two candidate vectors, whereas $g()$ is an estimate of $f()$.

Importance sampling assumes that the expert can tell us the regions in the probability space that will give us highly volatile $f()$. The candidate vector selection algorithm can then be biased to disproportionately sample vectors from these regions. Of course the output will need to be normalized to adjust for this bias. DNB does not require the MCS designer to have knowledge of the areas of greatest volatility. DNB does not explicitly bias the distribution of candidate vectors. DNB can be used even when little is know about the areas of high volatility or the specifics of the distribution. However, the $B()$ in DNB can be similar to importance sampling in the following way. If an area is known to be volatile, and it is embodied in $B()$, small neighborhoods will be created, forcing DNB to evaluate candidate vectors more frequently in these volatile areas. In the less volatile areas, if this knowledge is embodied in $B()$, large neighborhoods will be created, and less evaluations will be performed in these less volatile areas.

The similarities between the three techniques is that they all provide a precise framework to query an expert. The main difference is the type of information we seek from the expert. All three of the VRTs force the implementer to do some experimentation to determine if the variance will be reduced, otherwise, if applied improperly the variance can be increased. Also, the variance reduction must be substantial enough to still provide an overall application speed up due to the increase in overhead of the additional variance reduction code.

Notice that the other VRTs can be used *in conjunction with* DNB. Importance sampling's only effect is biasing of the candidate selection algorithm. Control variates simply subtracts a value for each vector. Neither of these strategies interferes with the application of DNB.

4 PROPERTIES OF PROBLEMS

Three characteristics distinguish the class of problems to which this technique should be applied.

1. Calculating $f(.)$ is expensive.
2. Random scenario generation is inexpensive.
3. An expert in the field can provide a relatively unbiased bounding function $B(.)$ that can be computed inexpensively. $B(.)$ need not be a tight bound, although, the tighter this bound, usually the bigger the savings.

We conjecture that domain experts can give a, crude estimate of the bounding formula $B(.)$ for many problems. $B(.)$ may be quite un-tight, but if it exists, it should be used to reduce the output variance if every function evaluation is sufficiently costly. This is where DNB's savings comes from. The bounding formula $B(.)$ is central to the technique. If a domain expert cannot provide a bounding formula, DNB cannot be applied. If $B(.)$ is biased the output may achieve a variance reduction, but the estimator will no longer be accurate.

If the function is Lipschitz continuous, or differentiable, then $B(.)$ is always bounded by some constant times the Euclidean distance. This is not always the best $B(.)$ to use. Often there is a 'tighter' bound, or a cheaper bound. For instance, in the case study detailed in the next section, we found both a significantly cheaper function and tighter bound then computing the Euclidean distance.

5 CASE STUDY

We chose pre-settlement risk (PSR), Hull (1989), as our case study. Evaluating PSR is an important financial problem that must be considered by every financial investment firm.

In this problem, a firm holds a portfolio of unsettled contracts. We are concerned with the risk of the counterparty going into bankruptcy and defaulting prior to settlement. If the counterparty defaults, we must go to the marketplace to replace the contracts, and will suffer a financial loss on the contracts that have a positive value.

The risk of a customer defaulting at time t in the future can be *estimated* using a Monte Carlo method to evaluate PSR. The method for determining PSR would be to simulate the market variables changing over time and calculate the effect on portfolio value. So, in our example, v is the vector of market variables and $f()$ is the present value of the portfolio at time t in the future. Actually $f()$ is the maximum of the present value or 0 since we can never have a negative risk, at best we can have a risk of 0. In order to assess risk over time, a separate simulation for every time slice t must be performed. A Company could then choose the maximum 97% confidence interval of any of the time slices, as an acceptable assessment of risk. Based on this assessment of the portfolio risk and the probability that the customer will default, a credit limit for the customer can be set.

For the purposes of demonstrating the effectiveness of DNB, we will only be looking at one time slice of risk, one month in the future. Instead of dealing with a portfolio of many different financial instruments, we will only consider portfolios of only forward foreign exchange contracts. A full system would consider all time slices, and all types of foreign exchange and interest rate securities including options, bonds, swaps, futures, and so on.

5.1 Calculations

There are four stages of the calculation: calculating the initial market at t_0 ('todays' rates), random number generation, generating forward forward FX rates, and calculating the present value of the portfolio marked to the simulated rates. A forward forward rate would be a forward rate beginning some time in the future. An example of a forward forward contract would be a one year forward rate where the contract starts in 1 month from the contract date. The derived rates are stored for the remainder of the simulation.

The initial market at t_0 is a combination of quoted rates and derived rates. The spot FX and money market interest rates for each of the countries are quoted rates. Spot rates are stored as vectors. All foreign exchange rates are stored quoted relative to US dollars (USD). All interest rates and derived zero coupon discount factors are stored as matrices where currencies are the rows and the tenors are the columns. Common tenors would be 1, 2, 3, 6, 9, 12, 24, 36, 60, and 120 month rates. From the money market rates we derive zero coupon discount factors. Using the zero coupon discount factors with the spot rates, we derive the forward FX rates. A one month forward FX rate is what the market believes that FX spot rate will be in one month. Since this is the best estimate we have, the simulated forward FX rates will be centered around this initial estimate of the forward FX rates. Similarly, the simulated money market rates will be centered around the 1 month forward interest rates.

The other inputs to our program will be a cross correlation matrix of currencies. Instead of using a three dimensional cross correlation matrix for the interest rates, we make the simplifying assumption that the volatility of the interest rates is 16% for any currency and tenor, and that there are no interest rate correlations between different countries.

One random scenario is calculated by first generating a vector of multivariate normal random numbers to be used in the simulation of forward FX spot rates. Generating a vector of independent normal random numbers with 16% volatility to be used in the simulation of forward zero discount coupon factors. Simulating the one month forward FX spot rates, which are centered around the initial estimate of the one month forward FX rates. Simulating a random change in the money market rates. From these, derive a forward zero discount coupon factor matrix. Use the forward FX spot rates and the forward zero discount coupon factor matrix to derive the matrix of forward forward FX for various tenors rates. The forward forward FX rates (FFFXR) is a matrix of forward rates for every tenor starting at time t. *The FFFXR matrix is the candidate vector v.*

Calculating $f()$ is a multistage calculation. First we mark the contracts to the simulated forward forward FX rates. Since we are considering the risk of default in a month, we need to only consider contracts that have maturities greater than or equal to one month. Because the counterparty will not honor contracts with positive value to the firm, our maximum loss (maximum risk) is the sum of all *positive* present value (pv) over the set of all contracts.

Our $f()$ calculation is the calculation of maximum loss. Notice that $f()$ can be arbitrarily expensive to calculate since there can be an arbitrarily large number of contracts. A large customer could have over 1000 forward FX contracts in their portfolio. Also note that if more security types are added, since we have done a full simulation of the interest rate environment, random scenario generation would not take more time, only the cost of $f()$ would increase. Adding more security types would only further exacerbate the computational demand thus creating a better environment for DNB.

5.2 Applying DNB to PSR

So far we have described v and $f()$. The only additional information needed to implement DNB is to give a bounding function $B()$. The bounding function must be calculated quickly. For this specific problem, interest rates shifts have much less dramatic effect on the *pv* of the portfolio than a shift in a forward FX rates. The FFFXRs are a function of forward zero discount coupon factors and forward FX spot rates. We condensed the maximum interest rate shift into a constant. By analysing the calculations, we found that the forward FX rates could never be more than 1.9 times the absolute difference of the contracted rate and the forward FX spot rates. The constant 1.9 is specific to the formulas in this simulation and is not generalizable to other interest rate models, or if new types of securities are added. With this major simplification, only the forward FX spot rate vector needs to be saved. Remember $B()$ is an overestimate of the difference, it is not a tight bound. This saves a lot of time over storing the entire FFFXR matrix.

Recall control variates and importance sampling VRTs. For this problem control variates would require the MCS designer to construct an inexpensive function $g()$ that approximates $f()$. For importance sampling, we would need to know the areas that would give use the most volatile $f()$. It would seem reasonable to use the extreme rate change scenarios to guide the importance sampling. This works for our simple test case of forward FX contracts, but does not work in general for all securities. Sometimes the maximum loss occurs when the rates remain constant!

5.3 Indexing data points

For PSR simulation the stored vectors were indexed based on the sum of the absolute distance between each element of the forward spot vector and the initial spot vector. Points in the same bin are not necessarily in the same ϵ-neighborhood. Two forward spot vectors can deviate by the same distance, but may deviate in different ways.

We initially decided to break the stored points into 200 bins. For epsilon values of up to 40MM (MM is an abbreviation for 1,000,000, so 40MM is 40 million), at most 15 of the 200 bins needed to be checked. From the indexed value, we inspected the 7 neighboring bins on either side of the indexed value. This reduced the number of points that needed to be checked by a factor of about 8.

5.4 Experimental Results

In order to analyze the variance of the output of a Monte Carlo simulation, for a given epsilon, we must perform hundreds of complete simulations, where each complete simulation consists of thousands of simulations.

In this problem, we are interested in the standard deviation of the average, 84.1345% element, and 97.7250% elements. We will refer to these as our *three metrics.* Standard metrics give us a way to intelligently compare different alternatives. These metrics were chosen because if the output distribution were normal (Gaussian), these would be the average, first and second standard deviation elements. Also, it is important to make the standard deviation magnitude independent, so we divide by the mean, and express it as a percentage. We felt that this was a better method of displaying the information than in numbers with magnitude. The mean used both in the standard deviation calculation and the division to make the standard deviation magnitude independent is the 'local' sample mean. In addition to the standard metrics we needed standard rates and a standard portfolio. We choose rates that we had stored from over one year ago and used a composite portfolio of 115 contracts totalling approximately 500MM, using 15 currencies, that closely mimicked an actual portfolio.

There are general trends to notice that will be illustrated in figures 1-6, and table 1. We conjecture that these trends are universal across many DNB applications, although much experimentation must be done to make this conclusion firm.

Figure 1 displays the number of candidate vectors in a simulation verses the sample standard deviation for our three metrics. Figure 1 does not use DNB. For example, if we perform five hundred simulations of four thousand candidate vectors each, the standard deviation of the 500 averages is 0.43%, the standard deviation of the 500 84% elements is 0.64%, and the standard deviation of the 500 97% elements is 0.90%.

There are a few trends to notice on figure 1. First, notice that the standard deviation of the 97% element is always greater than the standard deviation of the 84%-element, and the standard deviation of the 84%-element is always greater than the standard deviation of the average element. The second thing to notice is that the sample standard deviation decreases as the number of candidate vectors increases.

Figure 2 is the number of candidate vectors vs. the hit ratio for various epsilon. As the number of candidate vectors increases, the hit ratio increases and approaches 100%, and the hit ratio first increases quickly then increases slowly.

Figure 3 is the most important figure of our study. For concreteness, the data points of figure 3 are detailed in table 1. The sample variance with and without using DNB is compared for our three metrics. We make the assumption that the time to do a neighborhood check is negligible compared to the time to do an $f()$ calculation. With this assumption we ignore the number of candidate vectors and only compare the number of evaluated vectors necessary to achieve a given variance.

For example, for an epsilon of 35MM, the standard deviation of the 97%-element is 0.840% (although having epsilon greater than 5% of the portfolios total value may seem high, remember that $B()$ can be a *large overestimate,* the true error). This was computed for 5000 candidate vectors with a hit ratio of 3450/5000, or 69%. So, only 1550 points were evaluated out of the 5000. Assuming that the cost of a neighborhood check is trivial compared to an $f()$ calculation, it is instructive to compare the standard deviation of the 97% element of 1550 evaluations without DNB. This the essence of figure 3. The bar charts at the bottom of the figure displays our three metrics with DNB for the various epsilon, and the line chart displays corresponding three metrics *without* DNB for an equivalent number of evaluations determined by equating the hit ratio when DNB is used.

Figure 4 is a derivative of figure 3 (and table 1). Figure 4 measures the percentage improvement of using DNB.

When evaluating a choice of epsilon, there are two criteria: is the variance reduced, and is the estimator still accurate? It is irrelevant if we can reduce the variance of the output, but we are no longer predicting the correct value. Figure 5 shows the explosion of error of the estimator when epsilon is chosen too large. By running 500 simulations of 10,000 simulations without using DNB, we know the 'correct' values of our three metrics. We can use these correct values and see how far the absolute difference of the correct value and our estimator value deviate, at various epsilon. Looking at figure 5, we can see that the estimators are very accurate up to an epsilon of about 50MM. After 50MM, the standard deviation of the 84% and 97% elements start to vary drastically.

All the previous figures assumed multiple hits. Recall that multiple hits refers to allowing a candidate vector to 'hit' with multiple vectors in P. Figure 6 is the same as figure 5, but for single hits (a candidate vector may 'hit' with at most one vector in P). Notice in figure 6 that the estimator almost immediately becomes too inaccurate. We believe that this is due in part to the indexing structure specific to our implementation. It is possible that a different index structure would have produced better results for figure 6.

The most important thing to notice when looking at the charts is that the estimator of the mean is accurate for moderate epsilon and at those epsilon, DNB delivers large variance reductions.

6 CONCLUSION

The proposed DNB technique is not without problems:

- Experimentation must be done to efficiently use DNB. There is no general a priori automatic procedure for determining the number of candidate vectors and evaluation, how to index the data points, the 'tightness' of the bounding function $B()$, and the proper setting of ϵ. We are currently working on a software tool to aid the testing of a DNB implementation, to provide a user with an experimental means of exploiting the use of DNB.
- DNB increases the complexity of the MCS program. However, our experience teaches that it is not a major programming effort to add DNB to an existing MCS application. Additional software tools are being developed to make the process easier.
- If the time to calculate $f()$ is not large compared to the time to perform a neighborhood check and random scenario generation, the overhead of DNB may actually slow down the overall performance of the program.

However, if DNB is applied appropriately in a well engineered simulation then:

- DNB can provide a substantial output variance reduction, and can reduce overall execution time.
- DNB provides a general mechanism for domain expert - simulation expert interaction.

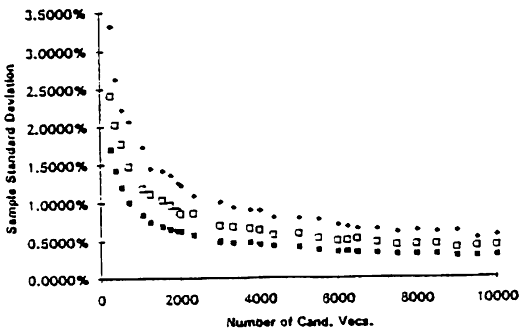

Figure 1: Number of Cand. Vecs. vs. Sample Deviation

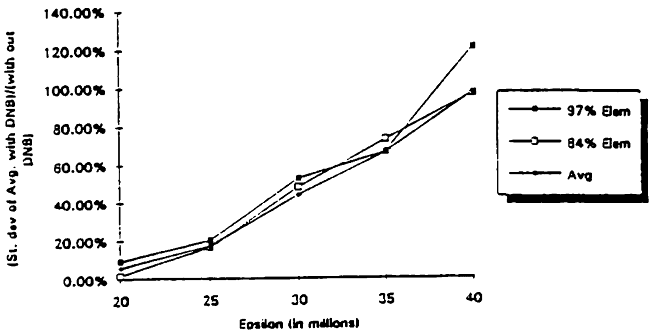

Figure 4: Percentage Improvement using DNB

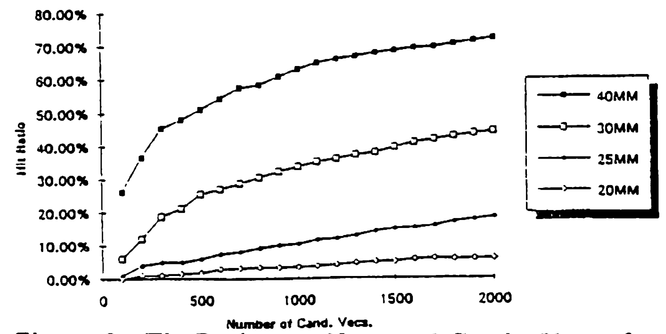

Figure 2: Hit Ratio vs. Num. of Cand. Vecs. for Various Epsilon

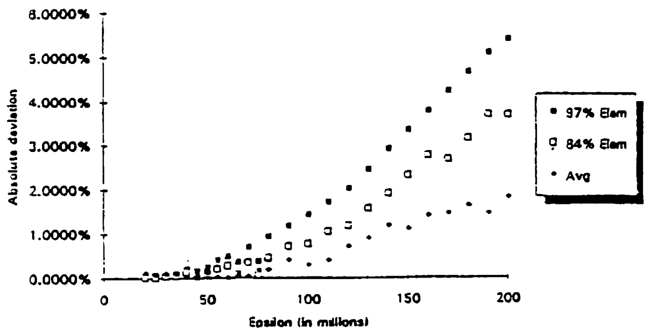

Figure 5: Estimator Accuracy with Multiple Hits

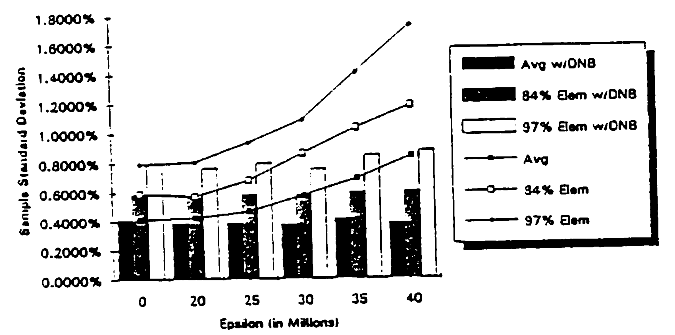

Figure 3: Sample Deviation with and without DNB

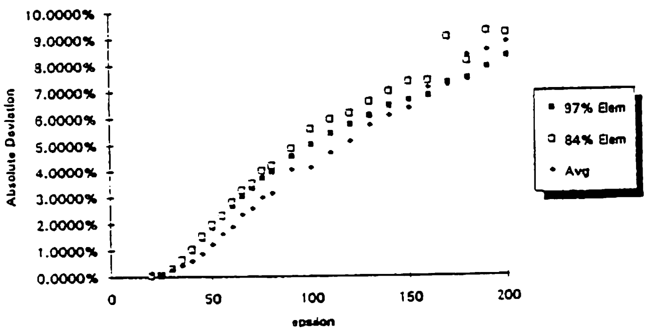

Figure 6: Estimator Accuracy with Single Hits

Table 1: Sample Deviation with and without DNB

	Num	DNB w/5000 Candidate Vecs.			Without DNB		
Epsilon	Evals.	Avg.	84%-Elem	97%-Elem	Avg.	84%-Elem	97%-Elem
0	5000	0.4101%	0.5899%	0.7930%	0.4101%	0.5899%	0.7930%
20	4350	0.3867%	0.5635%	0.7625%	0.4224%	0.5728%	0.8060%
25	3330	0.3883%	0.5843%	0.7970%	0.4667%	0.6812%	0.9354%
30	2330	0.3778%	0.5807%	0.7552%	0.5765%	0.8597%	1.0869%
35	1550	0.4123%	0.5955%	0.8448%	0.6875%	1.0316%	1.4078%
40	1050	0.3824%	0.6008%	0.8761%	0.8419%	1.1812%	1.7270%

DNB is being implemented in a general and easy to use software system to allow an engineer or analyst a way of experimenting with this technique.

ACKNOWLEDGMENTS

We thank Terrance Boult, Michael Elhadad, Paul Glasserman, and Spassimir Paskov for many useful discussions that improved the ideas presented in this paper. This work has been supported in part by the New York State Science and Technology Foundation through the Center for Advanced Technology under contract NYSSTFCU01207901.

REFERENCES

Russell C.H. Cheng, *Variance Reduction Methods,* Proceedings of the 1986 Winter Simulation Conference

John C. Hull, *Options, Futures, and other Derivative Securities,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989

Barry L. Nelson, *A Decomposition Approach To Variance Reduction,* Proceedings of the 1985 Winter Simulation Conference

Barry L. Nelson, *Variance Reduction for Simulation Practitioners,* Proceedings of the 1987 Winter Simulation Conference

James R. Willson, *Variance Reduction Techniques for Digital Simulation,* American Journal of Mathematical and Management Sciences, Vol. 4, Nos. 3 & 4, p277-312, 1984

James R. Willson, *Variance Reduction in Simulation,* Proceedings of the 1984 Winter Simulation Conference

AUTHOR BIOGRAPHIES

JASON S. GLAZIER is a doctoral candidate at the Department of Computer Science, Columbia University. He graduated in 1989 with a B.S. in CS from Johns Hopkins University, and in 1991 with a M.S. in CS from Columbia University. His research interests include computationally complex financial applications with emphasis on allocation problems, optimizations, parallel processing, and simulations. In 1992 he founded Complex Computing Company, Inc. which actively develops products for the financial industry.

SALVATORE J. STOLFO is an associate professor of computer science at Columbia University. His main fields of interest are artificial intelligence, knowledge-based systems, and parallel processing. He has published extensively in those fields. He was the principal architect of the 1023-processor DADO2 parallel computer and ACE, one of the first widely deployed expert systems. He consults for large companies in the information systems business. Stolfo received his B.S. in computational mathematics from Brooklyn College in 1974, and his Ph.D. in CS from Courant Institute, New York University, in 1979.