Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups

CATHERINE MCGEOCH

Department of Mathematics and Computer Science, Amherst College, Amherst, Massachusetts 01002

Although experimental studies have been widely applied to the investigation of algorithm performance, very little attention has been given to experimental method in this area. This is unfortunate, since much can be done to improve the quality of the data obtained; often, much improvement may be needed for the data to be useful. This paper gives a tutorial discussion of two aspects of good experimental technique: the use of *variance reduction techniques* and *simulation speedups* in algorithm studies.

In an illustrative study, application of variance reduction techniques produces a decrease in variance by a factor 1000 in one case, giving a dramatic improvement in the precision of experimental results. Furthermore, the complexity of the simulation program is improved from $\Theta(mn/H_n)$ to $\Theta(m + n \log n)$ (where m is typically much larger than n), giving a much faster simulation program and therefore more data per unit of computation time. The general application of variance reduction techniques is also discussed for a variety of algorithm problem domains.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*random number generation*; I.6 [**Computing Methodologies**]: Simulation and Modeling; I.6.3 [**Simulation and Modeling**]: Applications; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*discrete*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Experimental analysis of algorithms, move-to-front rule, self-organizing sequential search, simulation, statistical analysis of algorithms, transpose rule, variance reduction techniques

INTRODUCTION

Of central importance in computer science is the search for efficient algorithms to solve problems. When analyzing these algorithms we are usually concerned with finding bounds on functions relating input size to some combinatorial (machine-independent) measure of performance. For example, we might claim that algorithm A requires no more than $O(n^2)$ comparisons to sort a list of n items (this "big-oh" notation is defined later in this section). If the claim holds for all possible lists of size n, then we have a *worst-case* bound. Alternatively, we might establish a probability distribution over the set of possible inputs of size n and show that

CONTENTS

the expected number of comparisons is no more than $O(n \log n)$: this is an *average-case* bound on the algorithm.

Such analytical bounds provide a classification system that has been fairly reliable in predicting algorithm performance in practice; no matter what computer architecture or programming language is used, an $O(\log n)$ algorithm will be considerably faster than an $O(n^2)$ algorithm if n is large enough, and the difference will increase as n grows. For example, Bentley [1986, Chapter 7] describes an $O(n)$ algorithm implemented on a Radio Shack TRS-80 that runs faster than an $O(n^3)$ algorithm implemented on a Cray-I supercomputer (for $n > 2500$).

However, there are some limitations to this analytical approach. Worst-case bounds can be overly pessimistic in predicting performance in practice. On the other hand, average-case analyses can be very difficult to obtain for even the simplest of probabilistic models. As a general rule, average-case bounds using realistic models of input are beyond the limits of our current analytical methods.

When analytical methods fail, computational experiments may be used to study the average-case performance of algorithms using randomly generated problem instances. The process seems straightforward enough: implement the algorithm, generate the instances, and use appropriate statistical methods to analyze the results. But there is no guarantee of success. Experience shows that it can be very difficult to obtain reliable estimations of functional forms from experimental data. Experimental studies can produce inconclusive or even misleading results. For example, one series of experimental studies of bin-packing algorithms [Bentley et al. 1983; Csirik and Johnson 1991; McGeoch 1986a] produced conjectures that contradicted those of earlier experimental studies [Johnson 1973; Ong et al. 1984]. In another case, an extensive study of random insertion and deletion in binary trees gave results that contradicted those of a previous smaller study [Eppinger 1983; Knott 1975].

Certainly the *right* experiment can provide powerful insights and even stimulate new theorems. But in general how can we design the right experiment, one that will give correct, precise, and unambiguous results about algorithm performance?

This paper discusses two factors in the development of simulation studies that can influence the quality of simulation results. One is the observed *variance*, or "spread," in the data. A common goal of experimental studies is to estimate the mean of some quantity by averaging over several random trials. The reliability of this estimate depends on the variance in the results. For example, Figure 1 shows the results of two experiments described in Section 2. The left panel presents results of one experiment from a straightforward implementation of two algorithms. Each column of data shows 50 observations of a random variate D that measures the performance difference between the two algorithms for ten parameter settings (m = 100, 200, ... 1000). The large crosses mark column means; the horizontal reference line is located at zero. The right panel shows

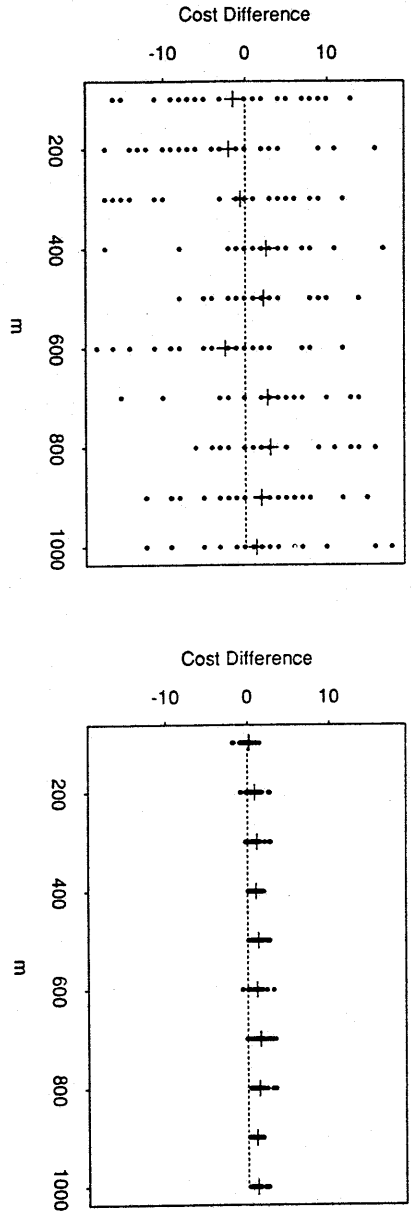

Figure 1. Results of two experiments.

the result of another experiment, after application of several *variance reduction techniques* (VRTs) to the simulation programs. Note that both graphs represent the same quantity D and that the expectations of the means in corresponding columns are provably identical. The right panel gives a more reliable view of the means because the variance in each column is small compared to the change in means between columns. It is known analytically that the mean is negative for small values of m and positive for large m, but the crossover point is not known; consider which data set would be more useful for estimating the location of the crossover point.

Variance in the data is an important factor in any simulation problem. Pawlikowski [1990] has remarked that "despite the fact that various VRTs have been extensively studied theoretically since the beginning of digital simulation, most of them have found only limited practical application." However, as is evidenced by the above example and by several more in this paper, algorithm problems are rich in opportunities for exploiting variance reduction techniques. This may be because algorithms are generally simpler and more rigorously defined than more traditional simulation problems and because experimental studies of algorithms usually begin with some partial theoretical understanding.

A second factor is *simulation efficiency* —how quickly results are returned by the simulation program. Program efficiency influences such choices as largest problem size, the number of trials, and the number and range of parameters considered. Also, efficient simulation can reduce variance by allowing more experiments per unit of time, since variance is inversely proportional to the number of independent sample points taken. Conversely, a reduction in variance can provide a simulation speedup when fewer trials are needed to obtain the desired accuracy, even though a single trial may take longer. As is the case with variance reduction techniques, algorithmic problems appear to present especially strong opportunities for exploiting *simulation speedups*, which can dramatically reduce the time required to generate the data.

Section 1 of this paper introduces a problem concerning two algorithms and their known theoretical analyses. Section 2 gives a tutorial discussion of a series of experiments in which variance reduction techniques are applied to simulation programs for the two algorithms;

also the end of the section describes several simulation speedups. Section 3 presents some variance reduction techniques that were not applied in the example study but which might be of general use. Section 4 surveys experimental studies from the algorithm's literature and discusses variance reduction techniques and speedups for general algorithm problems.

This paper assumes some familiarity with algorithms and their analysis at about the level of an undergraduate course. In particular, an algorithm A is said to have $O(f(n))$ running time if there exists a constant c such that the total number of operations A performs is asymptotically never *more* than $cf(n)$ when the input is of size n. The algorithm has $\Omega(f(n))$ running time if the number of operations is never *less* than $af(n)$ for some constant a. An algorithm has $\Theta(f(n))$ running time if both the upper bound $O(f(n))$ and the lower bound $\Omega(f(n))$ hold.

The only statistical background required is familiarity with elementary terms such as *mean*, *variance*, *covariance*, *estimator*, and *distribution function*, which can be found in any introductory statistics textbook.

This is by no means a complete or rigorous treatment of variance reduction techniques; the intention is to provide only a tutorial introduction to the subject. For more thorough and more technical presentations see Bratley et al. [1983], Cheng [1986], Hammersley and Handscomb [1964], Kleijnen [1974], Law and Kelton [1982], Nelson [1987], and Wilson [1984]. Some of the rich lore of experimental methodology for simulation studies can be found in textbooks [Bratley et al. 1983; Law and Kelton 1982; Payne 1982].

1. THE SELF-ORGANIZING SEARCH PROBLEM

This section describes two algorithms that are to be compared experimentally. Suppose we have a *search list* containing some ordering of n items that are named 1 through n. A sequence of m *requests* for items is presented, where a request for item i causes a search for that item starting from the front of the list.

A *sequential search rule* tries to maintain the list so that frequently accessed items are near the front, thereby reducing the cost of linear searches. The *optimal list order* is the one in which the most frequently accessed item is first in the list, the second-most frequently accessed item is second, and so forth. However, the search rule does not know the request frequencies in advance; therefore, the rule is allowed to modify its list after each request in order to approximate the optimal order. The *Move-to-Front* rule moves the requested item to the front of the list. The *Transpose* rule moves the requested item ahead by one position in the list. (No rearrangement occurs if the requested item is already at the front of the list.) Although several other rules have been proposed [Bitner 1976, 1979; Gonnet et al. 1982], these two have received the most attention.

The goal is to determine which search rule gives the lower average cost of searching for requested items. A common theoretical model assumes that the request sequence is formed by drawing item names randomly and independently according to some distribution $P = \{p_1, p_2, \ldots, p_n\}$. That is, the probability that "3" is the next item requested is given by p_3. Without loss of generality let $p_i \geq p_{i+1}$; therefore the optimal list has the order $1, 2, \ldots n$.

Suppose the list is in some ordering π. The *request cost* of item i is $\pi(i)$, its position in the list. The *expected list cost* for an ordering π and distribution P is found by summing over all items: $L(\pi, P) = \sum_{i=1}^{n} p_i \cdot \pi(i)$. Let $\Pr(\pi, m)$ be the probability that the list is in order π just before the mth request (the list is assumed to be in some random order initially). The *expected request cost* for a search rule is found by summing over all possible list orderings, that is, by $\sum_{\pi} \Pr(\pi, m) \cdot L(\pi, P)$. Expected request costs for Move-to-Front and Transpose

are denoted by $\mu(n, m)$ and $\tau(n, m)$. Note that the expected request cost for the optimal list is the same for any m and is equal to $\sum_{i=1}^{n} ip_i$. This gives a lower bound on the expected request cost for any search rule.

Probabilistic analyses have considered sequential search as a Markov process, where the *asymptotic expected request cost* is derived from the steady-state probabilities of the search list orderings for the rules as m goes to infinity. We denote asymptotic expected request costs by $\mu(n)$ and $\tau(n)$.

Gonnet et al. [1982] give closed forms for $\mu(n)$ for several specific probability distributions P. For example, Zipf's Distribution is defined by

$$p_i = \frac{1}{iH_n} \quad \text{where} \quad H_n = \sum_{j=1}^{n} \frac{1}{j}.$$

H_n is the nth Harmonic number and grows as $\Theta(\log n)$. For this distribution $\mu(n) \approx 1.386n/H_n$, whereas the optimal list has cost n/H_n.

Bitner [1979] analyzed Move-to-Front for arbitrary P and obtained an exact formula for expected request cost:

$$\mu(n, m) = \mu(n) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{(p_i - p_j)^2}{2(p_i + p_j)} \cdot (1 - p_i - p_j)^m$$

where the asymptotic cost is given by

$$\mu(n) = \frac{1}{2} + \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{p_i p_j}{p_i + p_j}.$$

Rivest [1976] obtained a formula for the asymptotic cost of the Transpose rule:

$$\tau(n) = \Pr(I_n)^{-1} \sum_{\pi} \prod_{i=1}^{n} p_i^{i - \pi(i)} \sum_{j=1}^{n} p_j \pi(j)$$

where

π = an ordering of the search list

$\pi(i)$ = the position of item i in π

$Pr(I_n)$ = the probability of the optimal ordering occurring initially

Rivest also showed that $\tau(n) \leq \mu(n)$ for any P and that the inequality is strict for all but certain trivial probability distributions. On the other hand, Bitner [1979] showed that while Transpose has better asymptotic cost, Move-to-Front converges to its asymptote more quickly.

It is worth noting at this point that the probabilistic model is only one of several approaches to algorithm analysis; any single analytical model will give an incomplete picture of an algorithms's performance. Under alternative models of analysis that do not assume independence in the sequence of requests, Move-to-Front can have lower asymptotic cost than Transpose (see Bentley and McGeoch [1985] and Sleater and Tarjan [1985]).

The probabilistic analyses that do exist are of limited use in comparing the two search rules. First, Rivest's formula for $\tau(n)$ involves summing over all $n!$ orderings of the list. Therefore, $\tau(n)$ cannot be directly computed when n is moderately large. Second, no formula is known for $\tau(n, m)$; although the analyses suggest that Move-to-Front is best when m is small and Transpose is best when m is large, a transition point in terms of P cannot be found analytically.

Experiments have been used to investigate these and related open questions. Previous experimental studies of the search rules are surveyed in Section 4. The next section describes a series of experiments developed to compare the expected request cost $\mu(n, m)$ for Move-to-Front to the expected cost $\tau(n, m)$ for Transpose for requests drawn according to Zipf's distribution. The goal is to obtain a simulation program that is efficient and that produces results having small variance.

2. VARIANCE REDUCTION AND THE SEQUENTIAL SEARCH PROBLEM

We begin by establishing some terminology. In general, algorithm A represents the *simulation model* for which we wish

to quantify the expected value θ of some performance measure, in terms of some parameters (x, y) which describe the distributional properties of the input.

A simulation program is implemented to study the algorithm. A *sample point* is determined by fixing values of the input parameters. A *trial* corresponds to a single input instance (pseudo-) randomly generated for a sample point. A set of t independent trials forms a *run* of the experiment.

Output measure X_j is a random variable produced by the simulation program at the jth trial such that the mean X of the X_j's is an *estimator* of θ at the sample point. We want the estimator to be *unbiased*: that is, $E[X] = \theta$. The variance in X_j is defined as $\mathrm{Var}(X_j) = E[(X_j - \theta)^2]$. In a simulation study where θ is not known, we measure the sample variance by

$$V(X_j) = \sum_{j=1}^{t} \frac{(X_j - X)^2}{t - 1}.$$

Our goal is to reduce $\mathrm{Var}(X_j)$, which is estimated by $V(X_j)$.

The remainder of this section describes a series of experiments to investigate the expected costs of two rules for sequential search. Variates M_j and T_j give the costs of the mth request in the jth trial for search lists ordered under Move-to-Front and Transpose, respectively. Their difference is $D_j = M_j - T_j$. Variates M, T, and D denote means taken over t trials; clearly M, T, and D are unbiased estimators of the expected request costs $\mu(m, n)$, $\tau(m, n)$, and their difference $\delta(m, n) = \mu(n, m) - \tau(n, m)$.

All experiments below report the cost of the 1000th request on a search list of size 20 (i.e., the sample point $n = 20$, $m = 1000$), with requests generated according to Zipf's distribution. For notational convenience, the quantities to be estimated at this particular sample point are denoted μ, τ, and δ.

The random requests were generated by the method of *aliasing*. By this method, a random real U is generated uniformly over the range $[1, n + 1)$ and split into an integer part I (uniform on $1 \ldots n$) and a decimal part F (uniform on $[0, 1)$). If F is below some threshold value $T(I)$, then I becomes the request generated; otherwise, an *alias* value $A(I)$ becomes the request. This method requires $O(n)$ initialization time to set up the threshold and alias tables and $O(1)$ time per request. For more discussion of this and other methods see Bratley et al. [1983] and Devroye [1986].

The uniform variates U were generated by an additive method where $U_x = U_{x-24} + U_{x-55} \bmod 2^{30}$ (see Knuth [1981, section 3.2.2]). To check the random number generator, the first and last experiments were replicated using the system generator provided by Sun Unix Version 3.4. All experiments and timing measurements were performed on a Sun 3/50 workstation with no floating-point accelerator.

2.1. The First Experiment

The first experiment is a straightforward implementation of the search rules. The simulation program for Move-to-Front produces M_j at trial j by the following steps:

(1) Initialize the search list to some random order.
(2) Perform steps (3) and (4) $m - 1$ times.
(3) Generate a random integer r from the range $1 \ldots n$ according to Zipf's distribution.
(4) Find r in the search list. Move r to the front of the list.
(5) Generate the last request r; set M_j equal to r's position in the search list; and report M_j.

The simulation program for Transpose is identical except for the obvious change in step (4). A perhaps more appealing strategy would be to report, rather than the cost of the mth request, the average of the first m requests. That approach is discussed in a later section where it is shown to give a biased estimator of μ

(and τ). The initial program obeys a very good rule of thumb: simulate *exactly* the theoretical model.

The results of four runs of the first experiment are summarized below. Each run represents the results of 50 trials at the sample point ($n = 20$, $m = 1000$). From the discussion of the previous section, we know that $\mu = \mu(20, 1000) = 7.26$ [Bitner 1979] and that the expected cost of the optimal ordering is $\sum_{i=1}^{n} ip_i = 5.56$. The cost of the optimal ordering gives a lower bound on τ and μ.

M	$V[M_j]$	T	$V[T_j]$	D	$V[D_j]$
7.94	35.85	5.52	23.81	2.42	62.96
7.30	24.41	4.92	27.75	2.38	41.32
6.74	22.87	5.80	26.88	0.94	53.82
7.92	32.67	6.92	34.19	1.00	64.76

Large sample variance can cause experimental results to vary greatly across runs, indicating that the results from any particular run are not reliable. The estimator M varies from its known expectation 7.26 by as much as 9%, and the estimators D vary from one another by as much as 60%. Typically, sample variance in T_j is five times as large as the mean T. Indeed, two values for T are lower than the known lower bound on τ: although T must approach τ as the number of trials increases (by the law of large numbers), it appears that 50 trials are not enough to give an accurate estimation of τ.

One way to reduce variance is to increase the *quantity* of data with more trials per run. This can consume significant computational resources. The following sections describe ways to improve the *quality* of the data while maintaining the same number of trials.

2.2. Common Random Numbers

The technique of *common random numbers* reduces variance in the difference of two output measures by making trials as similar as possible between the two measures. In the search problem, for example, we have

$$\begin{aligned}\mathrm{Var}[D_j] &= \mathrm{Var}[M_j - T_j] \\ &= \mathrm{Var}[M_j] + \mathrm{Var}[T_j] \\ &\quad - 2\,\mathrm{Cov}[M_j, T_j].\end{aligned}$$

Variance in D_j is decreased if the covariance can be made positive. If separately generated request sequences are used for the two rules (as is the case in the first experiment), then the trials are independent and $\mathrm{Cov}[M_j, T_j]$ equals 0. There is reason to believe, however, that identical inputs would produce positive covariance; for example, a request for a rare item would probably have high cost for both rules.

The second experiment applies common random numbers to the search problem. The search lists for the two rules are initialized to identical (random) orders, and identical request sequences are used in each trial. Note that besides the possible reduction in variance, this technique gives a constant-factor improvement in simulation efficiency, since the cost of input generation is cut in half. Each run of the first experiment typically required 46.9 seconds while the second experiment required 26.2 seconds.

The second experiment produced the following statistics, where D_j^r and D^r represent the new estimator for δ obtained by application of common random variates.

D^r	$\mathrm{Var}[D_j^r]$
0.94	24.08
0.88	21.19
0.36	15.51
1.60	19.24

Expectations and variances of M_j and T_j are not affected by the use of this technique since the input distribution is not changed for either rule. Typical variance in D_j^r, however, is nearly a third of that in the first experiment.

The technique of common random numbers should be considered whenever algorithms are being compared, and there is reason to believe that their perfor-

mance might be positively correlated with respect to input instances.

2.3. Control Variates

Common random numbers are used to capitalize on positive correlation between algorithms; the technique of *control variates* exploits positive correlation *within* a single algorithm. Graphical exploration of the data from the second experiment revealed that, under Transpose, items tend to end up very near their optimal positions. Since, by our naming convention, item i is in position i in the optimal list, we can conclude that a strong correlation exists between item names and their positions in the Transpose list. (A look at Move-to-Front shows much weaker correlation.)

Let R_j represent the name of the last item requested at the jth trial; let R represent the mean over t trials; and let ρ represent its expectation, equal to 5.56 for Zipf's distribution at this sample point. Construct a new output variate $T_j^c = T_j - a(R_j - \rho)$, where a is a constant described below. The new estimator T_j^c uses the discrepancy between the observed value of R_j and its (known) expectation ρ to "correct" T_j towards its (unknown) expectation. Variate R_j is called a control variate for T_j.

The expectations $E[T^c]$ and $E[T]$ are equivalent, since $E[R_j - \rho]$ equals 0. Therefore, the new estimator is unbiased for any constant a, although variance in T_j^c is minimized by optimal choice of a. Finding the optimal a is difficult, however, since it depends on the variance and covariance of T_j and R_j, which are not known. Estimation of optimal a from measurements over many sample points can give erroneous results [Bratley et al. 1983; Kleijnen 1974; Nelson 1987]. In this case a reasonable strategy is to find a good a by linear regression.

The third experiment uses R_j as a control variate for M_j and T_j, reporting new variates M_j^c and T_j^c at each trial. Regression on R_j and T_j at this sample point gives $a = 1.16$; for simplicity the experiment sets $a = 1$. The results of four runs are given below.

M^c	$V[M_j^c]$	T^c	$V[T_j^c]$
6.29	30.51	5.28	7.88
6.08	32.52	5.72	5.50
8.44	25.11	5.97	2.20
6.98	14.51	6.97	7.92

Sample variance in T_j^C is smaller than that for T_j by a factor of 5. The weaker correlation between M_j and R_j gives only a small reduction for M_j^c, if any. Variance in D_j is not altered from that of the second experiment, since the control variate R_j is canceled out in the subtraction $D_j^C = M_j^C - T_j^C$. This VRT neither increases nor decreases computation time, since the only change in the implementation is the reporting of the last request R_j.

In general, suppose that we wish to estimate some quantity θ and that we have an unbiased estimator X. It may be possible to find a control variate Y with known expectation ψ. Setting $Z = X - a(Y - \psi)$ gives a new unbiased estimator for θ; variance in Z is reduced whenever Y is positively correlated with X.

In algorithm studies the control variate might be a measure of some property of the input, as is R_j. It might be a measure of the initial state of a data structure: for example, if the search cost for item R_j in the initial (random) ordering is K_j, (which has known expectation $\chi = (n + 1)/2$), then we might estimate τ by $T_j^K = T_j - (K_j - \chi)$ under the assumption that a high initial cost is correlated with a high cost at time m. (This assumption is probably invalid for large m). Other control variates may be obtained from lower bounds on the performance measure, simpler algorithms, or secondary performance measures. Some further examples are given in Section 4.

2.4. Antithetic Variates

Suppose as before that X is an estimator of θ. The method of *antithetic variates* requires a second estimator X with the

same distribution as X. Their mean, $(X + \hat{X})/2$, is an unbiased estimator of θ, and we have

$$\mathrm{Var}\left[(X + \hat{X})/2\right] = \frac{\mathrm{Var}[X]}{4} + \frac{\mathrm{Var}[\hat{X}]}{4} + \frac{\mathrm{Cov}[X, \hat{X}]}{2}.$$

For reduced variance the last two terms must have a negative sum, implying that covariance in X and $\hat{X}$ must be negative.

The usual scenario for use of antithetic variates assumes that output variate X is a monotonic function of a uniformly distributed input variate U. Taking a second trial with antithetic variate $\hat{U} = 1 - U$ gives the desired $\hat{X}$. With a little effort this technique can be generalized to nonuniform input distributions. Observe from the second experiment that T_j tends to increase with R_j. Therefore, an antithetic input variate $\hat{R}_j$ could be used to obtain an antithetic $\hat{T}_j$. The new estimator of τ would be $T_j^a = (T_j + \hat{T}_j)/2$.

How shall we generate $\hat{R}_j$ so that it is negatively correlated with R_j but is distributed according to Zipf's law? There appears to be no known way of modifying the method of aliasing (which was used in previous experiments) to accomplish this task. However, the *inversion* method (see Bratley et al. [1983] and Devroye [1986]) can be applied. This method generates a uniform variate U_j and uses a table-lookup scheme to determine the appropriate request value R_j; a second lookup using $\hat{U}_j = 1 - U_j$ will produce the antithetic value $\hat{R}_j$.

For generating requests according to Zipf's law, the inversion method is computationally more expensive than the method of aliasing because the table lookup requires $\Omega(\log n)$ time per request on average. This VRT, then, could increase the running time of the experiment, which may be acceptable if variance is substantially reduced. To minimize the extra cost, the following steps were implemented in the fourth experiment to produce antithetic variates.

(1) Generate R_j by the method of aliasing.
(2) Generate U uniformly from the range $[F(R_{j-1}), F(R_j))$. Set $U = 1 - U$.
(3) Generate $\hat{R}_j$ using the inversion method with $\hat{U}$.

Thus, only the second of each pair of variates is generated by inversion, requiring only one table lookup instead of two. Antithetic variates are generated for the final request only (not for the first $m - 1$ requests).

The fourth experiment applies antithetic variates to the Transpose rule only. Instead of producing 50 variates T_j, the experiment produces 25 pairs of variates $(T_j, \hat{T}_j)$ and reports their means. The running time of the fourth experiment (21.3 seconds) was slightly increased over the third.

Four runs of the fourth experiment produced the following statistics.

T^a	$\mathrm{Var}[T_j^a]$
5.98	6.73
5.68	7.33
5.96	3.56
5.74	5.02

Variance in T_i^a is comparable to that for control variates used in the third experiment. Antithetic variates give no substantial improvement in variance and a small increase in running time; therefore it is not particularly useful for this problem. The lack of strong correlation in R_j and M_j suggests that the technique would be of even less use in the simulation of Move-to-Front.

In general, it can be difficult to tell which VRTs will be beneficial for a particular problem. Kleijnen [1974] points out that the joint application of common random variates and antithetic variates can produce an overall *increase* in variance of differences D_i, and he derives conditions on the covariance structure for which variance is reduced. Unfortu-

nately, if the problem is tractable enough to permit analysis of the covariance structure, then there is probably no need for a simulation study. A small pilot study conducted at the beginning of the experimental study can reveal whether a given approach has any merit.

2.5. Conditional Expectation

Suppose that X is an estimator of θ and that there is another variate Y for which the expectation of X given Y is a known function of Y. Set $Z = E[X|Y] = f(Y)$. Then, Z is an unbiased estimator of θ also, and Var$[Z]$ is less than Var$[X]$ whenever Y has smaller variance than X.

Instead of generating X directly, the technique of *conditional expectation*, sometimes called *Conditional Monte Carlo*, generates Y and then reports $Z = f(Y)$. This replaces variate X by its expectation $E[X|Y]$. Informally, conditional expectation exploits an intermediate random variate Y from which the expected value of X can be calculated.

For the search problem, let the intermediate variable be π_j, the list ordering produced by the first $m - 1$ requests in trial j. It is easy to compute $Z_j = L(\pi_j)$, the expected cost of a particular ordering. Certainly, the mean Z over t trials is an unbiased estimator of expected request cost.

The fifth experiment applies conditional expectation to the search problem by generating π_j with the first $m - 1$ requests and then directly computing $L(\pi_j)$. Therefore, the last request R_j is not generated at all. Results of the fifth experiment are given below. Note that the estimator D^z was obtained by the joint application of conditional expectation and common random numbers.

M^z	Var$[M_j^z]$	T^z	Var$[T_j^z]$	D^z	Var$[D_j^z]$
7.30	0.45	5.90	0.021	1.39	0.45
7.19	0.38	5.87	0.016	1.31	0.36
7.18	0.28	5.88	0.017	1.29	0.29
7.48	0.45	5.91	0.027	1.56	0.37

The experiment required 22 seconds, a slight increase over the 21.3 seconds required for the second experiment. This is expected, since computation of list costs requires time proportional to n, rather than time proportional to n/H_n. In this case, however, the small increase in running time is well worth the dramatic decrease in variance, especially in the estimation of τ.

2.6. Simulation Shortcuts

Now we consider the problem of speeding up the simulation program. The key idea is that we wish to *simulate* an algorithm, not necessarily to *implement* it. It may be possible to apply algorithmic improvements to permit more trials per unit of computation time while leaving unchanged the distributional properties of the output variates. While not technically a variance reduction technique (because variance remains the same in each run), a simulation shortcut can reduce variance by allowing more trials per unit of time.

A simulation speedup might be obtained by exploiting knowledge that would not be available in an application. For example, we can exploit the knowledge that item names are integers and obtain a shortcut for the Transpose rule. In this rule, each sequential search is followed by a swap of the requested item with its predecessor. The sequential search for each item can be replaced by a constant-time table lookup: it is only necessary to maintain an auxiliary array `Location`, where `Location[i]` records the location of item `i` in the search list. After a "transpose" operation in the search list, only two entries in the location table must be modified. The average cost of the Transpose rule is reduced therefore from at least n/H_n per request (the expected cost of a search in the optimal list) to constant time per request. Note that this shortcut cannot be applied to Move-to-Front because of the large cost of modifying the location table after a move.

A second shortcut arises by observing

that the Move-to-Front rule only records the order of last appearance of each item. Instead of performing the first $m - 1$ searches (again with cost at least n/H_n per search), it is necessary only to record the last time of request for each item. This can be done in constant time per request using an array `LastTime`, indexed by item names. To restore the search list at the time of the mth request, it is necessary only to sort the search list elements according to the entries in array `LastTime`.

If both of the above shortcuts are used, the simulation program performs no searching at all; nevertheless, the output measures have exactly the same statistical properties as in a straightforward implementation. The running time of a simulation program that uses both speedups (as well as conditional expectation) is $\Theta(m + n \log(n))$, compared to $\Theta(mn/H_n)$ for the fifth experiment.

Other shortcuts may be gained by increasing the number of observations from a single trial. Simulations of search rules by previous authors (surveyed in the next section) have used this idea by reporting the average cost of the first m requests rather than just the cost of the mth request. Although this approach certainly reduces variance, it also produces a *biased estimator* of $\mu(n, m)$ and $\tau(n, m)$. Since requests costs are higher in the initial stages of the simulation, averaging tends to artificially raise the reported cost.

A good heuristic solution to the problem of bias is to discard some number of initial measurements and to begin accumulating costs after the simulation appears to be nearing asymptotic behavior. This corresponds to taking a *batched mean* of the last b requests and averaging list costs for requests $m - b + 1$ through m. Choosing a good value of b is a difficult problem in general, although some rules of thumb do exist [Bratley et al. 1983; Pawlikowski 1990].

Whether any of the above shortcuts produce significant speedups in the sequential search problem depends on the choice of sample points. The sixth experiment, tuned for the sample point (20, 1000), implements the `Location` array for Transpose and applies batched means with a conservative batch size of 10. Conditional expectation is applied to each of the last 10 lists obtained in each trial. For comparison with earlier results, the sixth experiment produces 50 observations by taking only 5 trials with 10 data points at each trial.

The table below compares the first experiment (top row of each pair) to the sixth experiment (bottom of each pair) at three sample points. The column marked "**Time**" shows an improvement in running time by more than a factor of 15. Estimators of μ, τ, and δ along with observed variance are given in the remaining columns. Variance in the estimator of μ is reduced by a factor of about 70, in τ by a factor of 1000, and in δ by a factor of about 250.

n	m	Time	μ	Var	τ	Var	δ	Var
20	1000	46.9	7.94	35.85	5.52	23.81	2.42	62.96
		3.1	7.02	0.259	5.89	0.021	1.14	0.238
20	2000	93.7	6.24	26.14	6.53	27.05	−0.32	50.54
		5.6	7.26	0.209	5.85	0.013	1.37	0.223
20	3000	141.2	7.46	28.2	6.46	35.96	1.00	65.64
		8.3	7.52	0.322	5.83	0.015	1.58	0.355

Figure 1, seen previously in the Introduction, gives a graphical view of the improvement in estimators of δ for $n = 20$ and several settings of m. The left panel shows values of D_j obtained from the first experiment, and the right panel shows values obtained from the sixth experiment.

3. OTHER VARIANCE REDUCTION TECHNIQUES

Some variance reduction techniques of general use were not applied to the

sequential search problem. This section contains a brief description of each and how they might be used.

3.1. Splitting

Considering the search problem as a two-step process—generate a random list ordering π_j by the first $m - 1$ requests, then generate a final request—suggests the application of *splitting*. For a fixed list ordering π_j, variate T_j may be viewed as an estimator of expected list cost for π_j. A better estimator is obtained by generating repeated requests R_{j1}, $R_{j2}, \ldots R_{jk}$ without changing π_j. Variate $\overline{T}_j$ is set equal to the mean of search costs $T_{j1}, T_{j2}, \ldots T_{jk}$. Instead of producing one estimator of expected list cost for π_j, this produces k estimators, which is guaranteed to reduce variance.

Splitting may be useful when a simulation produces two variates in sequence, where the first represents some "state" and the second is an estimator of expected cost for that state. When the generation of the state is expensive, this technique greatly reduces the amount of computing time required to obtain estimates of expected cost. The method of conditional expectation used in the fifth experiment extends this idea to direct computation of the expected cost of the state; splitting should be considered when direct computation is not feasible.

3.2. Stratification

Stratification reduces variance by distinguishing classes, or "strata," of input and generating input instances so that the number of instances occurring in each class corresponds exactly to its expectation. In the context of the search problem, we might identify three classes of request sequences A, B, and C, such that the probability that a particular sequence falls in each class is respectively p_A, p_B, and p_C. Instead of generating t request sequences (trials) independently, we could generate tp_A sequences from class A, tp_B from B, and tp_C from class C.

Another application of stratification to the search problem could be used to improve splitting. First, generate $m - 1$ requests to produce a list ordering π_j. Then, instead of generating a random set of k requests $R_{j1}, R_{j2}, \ldots R_{jk}$, divide the set of possible requests R_j into s *request strata* S_ℓ, for $\ell = 1, \ldots s$. Let q_ℓ denote the sum of probabilities of requests in stratum S_ℓ. Generate the set of k requests so that exactly kq_ℓ are from stratum S_ℓ for each ℓ.

While the technique may be of use in general, stratification was not applied in the search study because of the difficulty of finding appropriate classes, probabilities, and generation methods.

3.3. Poststratification

A method for "correcting" an estimator by use of auxiliary information is by *poststratifying* the experiment. Suppose input classes A, B, and C can be identified as above, but that stratification is not applied: rather, the instances are generated independently in a straightforward manner. Poststratification works by calculating differences between the expected number tp_A of instances in class A and the actual number of instances generated in the run. The differences for each class are then used to "correct" the output measure towards its expectation.

This technique could be applied to individual requests as well. Let $E[T_j|R_j \in S_\ell]$ denote the expectation of T_j, given that the last request R_j is from stratum S_ℓ. As before, q_ℓ denotes the sum of probabilities of requests in S_ℓ; this is equivalent to the probability that a random request falls in S_ℓ. Poststratification uses the fact that

$$\tau = E[T] = \sum_{\ell=1}^{s} E\left[T_j|R_j \in S_\ell\right] q_\ell$$

(see Nelson [1987]). Let N_ℓ equal the total number of last requests (over all t trials) falling in strata S_ℓ, and suppose that the request R_j falls in stratum S_{ℓ_j}. Then a new estimator is given by sum-

ming over all trials as follows:

$$T^p = \sum_{j=1}^{t} \frac{q_{\ell_j}}{N_{\ell_j}} T_j.$$

This estimator is unbiased as long as each N_ℓ is greater than zero. Poststratification can be applied in this case because the theoretical distribution of R_j is known, R_j is positively correlated with T_j, and the request R_j is independent of the list ordering at time m.

4. APPLICATIONS TO OTHER ALGORITHMS

This section describes some variance reduction techniques that have been applied (or could have been applied) to algorithm simulation studies in the literature. This is not an exhaustive survey; the intent is to show that these techniques have been applied profitably in a wide variety of problem domains. In most cases below, the VRTs were not mentioned by name in the cited works nor was there a stated intention to reduce variance in the data. Even without formal justification, VRTs have been recognized informally as components of good experimental method.

4.1. More Sequential Search

Several authors describe experimental studies of various sequential search rules including Move-to-Front and Transpose [Bitner 1976; Rivest 1976; Tenenbaum 1978]. All of these studies apply *batched means* by averaging search costs over several requests in a single trial. As was noted in the section on simulation shortcuts, averaging over time introduces bias in the estimation of the cost of the mth request because initial costs are high. Care should be taken in using such results to make inferences about convergence in the theoretical model.

For example, Tenenbaum's [1978] experiments start with a search list that is initially empty. Each item is added to the list after its first appearance in the request sequence. Once the search list contains all n items, the request cost is calculated by averaging over all subsequent requests. That is, this study applies batched means with the batch size equal to $m - f$, where m is the total number of requests and where f is the number of requests needed to fill the list.

One trial produced average cost $T = 6.924$ for the Transpose rule, using Zipf's distribution at the sample point $n = 25$, $m = 12{,}000$. Tenenbaum reports that none of his results varied by more than 2.5% in four independent trials at each sample point, implying in this case that the observations differed by no more than 0.173. Although the results of Section 2 are not directly comparable to Tenenbaum's because of the difference in sample points ($n = 20$, $m = 1000$), it is instructive to note that T varied by as much as 2.0 in four runs of the first experiment and by 0.04 in the fifth experiment. Batched means can reduce variance in a straightforward implementation, but other techniques can work as well or better.

4.2. Bin Packing

The *one-dimensional bin-packing* problem is as follows: given a list L of n weights in the range $(0, 1]$, pack them into unit-capacity bins so as to minimize the total number of bins required. This problem is NP-Hard [Garey and Johnson 1979], and several heuristic packing algorithms have been proposed.

Several experimental comparisons of various packing algorithms have appeared (for example, Bentley et al. [1983], Csirik and Johnson [1991], Johnson [1973], McGeoch [1986a], and Ong et al. [1984]). Typically, the input lists are formed of weights drawn uniformly and independently from the range $(0, u]$, for some upper bound u such that $0 < u \leq 1$. Motivated by worst-case analyses, one traditional goal has been to estimate β, the expected number of bins needed to pack a list according to a given packing algorithm. The number of bins required in trial j, denoted B_j, is an unbiased estimator of β.

Now, in trial j the sum of weights W_j in the weight list is a lower bound on B_j. In fact W_j is a *control variate*, since it is positively correlated with B_j and its expectation ω is easily computed. Therefore, β could be estimated by $B_j^c = B_j - a(W_j - \omega)$.

All of the experimental papers cited above report results for the "First Fit Decreasing" (FFD) algorithm. Two studies use B_j (or functions which depend on B_j) as the primary measure ([Johnson 1973; Ong et al. 1984]). Later studies [Bentley et al. 1983; Csirik and Johnson 1991; McGeoch 1986a] adopt *empty space*, defined by $E_j = B_j - W_j$, as the measure of packing quality. Note that $E_j + \omega$ is an unbiased estimator of β. Empty space was not used directly to estimate β in these studies; nevertheless, this alternative measure produces clearer results because of the reduced variance. Analytical bounds on β were subsequently derived based on the experiments measuring empty space.

Although none of the studies explicitly report variance for each sample point, it is possible to infer that in one case variance in B_j is near 126 for 20 trials at the sample point $n = 1000$, $u = 1$ [Ong et al. 1984, Table V]. In the studies which used empty space as the measure, a typical run of 25 trials at $n = 128{,}000$, $u = 1$ produced variance in B_j near 105, while variance in empty space E_j was about 41 (personal notes). Thus, using W_j as a control variate for B_j could give a 50% reduction in variance.

In more extensive studies of bin packing [McGeoch 1986a], FFD was studied for weights drawn uniformly from $(0, u]$, where $u \leq 0.5$. A second control variate is suggested by these experiments; the empty space in the last "catchall" bin in the packing is a source of noise positively correlated with total empty space. In experiments measuring "empty space in all but the last bin," the observed variance is smaller by a factor of 12,000 over the original measure B_i. *Common random numbers* were also applied in some of these experiments.

Another VRT that might be profitable for bin-packing heuristics is described below. Suppose several algorithms are to be compared for lists of n weights drawn uniformly from the range $(0, u]$. For each weight list $L = \{x_1, x_2, \ldots x_n\}$, let the *antithetic* list be computed by $\hat{L} = \{u - x_1, u - x_2, \ldots u - x_n\}$. The antithetic output variates are the bin counts produced for each pair of lists. Whether this technique produces a true reduction in variance depends on whether empty space is negatively correlated for two such lists. A small pilot study could determine whether this is the case.

4.3. Algorithms on Random Graphs

Kershenbaum and Van Slyke [1972] apply an elegant *simulation shortcut* in their study of spanning trees in random graphs. They use experiments to estimate $h(p)$, the probability that a spanning tree occurs in a random graph in which each edge appears with probability p.

A naive simulation program would generate many random graphs and record the existence or not of a spanning tree in each; this process must be repeated for each p of interest, say $p_1, p_2, \ldots, p_k$. Instead, their program constructs a complete graph G with edge weights $w(x, y)$ drawn from a uniform distribution. Random variate Q is the largest weight edge needed to connect the graph, and $h(p_i)$ is set to 0 if $Q > p_i$ and to 1 otherwise, for each $i = 1 \ldots k$. This shortcut program allows a single trial to produce simultaneous estimates for all p_i. The authors also apply *antithetic variates* by building two graphs G and $\hat{G}$ in each trial by using antithetic edge weights w and $\hat{w} = 1 - w$.

This shortcut technique could easily be generalized: generate a weighted graph G and assume that an edge (x, y) "exists" in the random graph if $w(x, y)$ is greater than p_i. Indeed, this relationship has been exploited to derive theoretical bounds on expected solution cost for problems on random weighted graphs,

including traveling salesman tours, bottleneck TSP, assignment, weights of minimum spanning trees, and k-cliques, in addition to bounds on solutions to some greedy algorithms [Lueker 1981; Weide 1980]. Note that these results provide a pool of possible *control variates* for heuristic algorithms on randomly weighted graphs, since their expectations are known. It is only necessary to establish (empirically, if necessary) that there is a positive correlation between the control variate and the cost being studied.

4.4. The Traveling Salesman Problem

Heuristic algorithms for the traveling salesman problem have received extensive experimental scrutiny. Rather than review the vast literature on the subject (see Lawler et al. [1985] for an introduction), we will highlight a few studies and discuss VRTs that have been applied and others that might be applied.

The use of *common random variates* when comparing TSP algorithms is widespread. Traditionally, there has existed a set of about 20 benchmark problems on which every newly proposed TSP algorithm is expected to be tested. One difficulty with these benchmark problems is that they become obsolete; although the largest such problem is around 500 nodes and 1200 edges, some recent TSP experiments have used random inputs with as many as one million nodes [Bentley et al. 1990]. Bentley proposes a new set of benchmark inputs that are arguably difficult for some algorithms and which are scalable in size, and he uses common random variates to present a detailed comparison of the tradeoffs between several TSP algorithms [Bentley 1990].

In studies of TSP algorithms, there may be several opportunities for finding *control variates*, depending on the input model used. Control variates for graphs with random edge weights were described in the previous section. For tour improvement heuristics, the initial tour might be a possible control variate for the final tour, if the expected cost of the former can be computed.

Finally, we describe a *simulation shortcut* for the Strip heuristic on Euclidean point sets [Beardwood et al. 1959]. This algorithm begins by breaking the input points into some number of vertical "strips," finds an optimal path through each strip, and then connects the paths in adjacent strips. For this algorithm, the expected total path length is a simple function of the expected cost in a single strip; instead of directly implementing the algorithm (generate points, determine membership in strips, sort points in strips, connect strips) it is possible to generate repeatedly (and quickly) a random path length for a single strip and apply the formula.

4.5. Quicksort

The basic strategy of Quicksort is well known. Given a list of items to be sorted, Quicksort chooses an element from the list and *partitions* the list so that items smaller than the partition element are to its left and items greater are to its right; therefore the partition element is in its correct sorted position. Quicksort then recurs to sort the sublists on either side of the partition.

Much of the efficiency of Quicksort depends on the choice of the partition element. A good partition element splits the array nearly in half. A standard strategy is to choose the partition element by selecting the median of a random sample of the current subarray. For example, median-of-three Quicksort takes a sample size of 3 at each stage. There is a tradeoff in determining the right sample size: a large sample gives better partitions but incurs a higher median selection cost. One experimental study examines strategies in which sample size is allowed to vary with subarray size at each recursive stage [McGeoch 1986a, 1986b].

One measure of interest is β, the total number of exchanges (swaps) of list items performed on a random list of size N. It

is easy to compute the expected number of exchanges $\hat{b}(n, r)$ at any recursive stage, given the subarray size n and the rank r of the partition element in that subarray. *Conditional expectation* was applied by calculating $\hat{b}(n, s)$ directly rather than by counting the actual number of swaps in a given trial.

The random variate $B(N)$ used to estimate β was found by summing the $\hat{b}(n, s)$ values over all stages. This technique is applied similarly to three other output measures. In fact, all that is required in the simulation program is generation of a random variate for the rank r of the partition element at each recursive stage (which determines subarray size at subsequent levels). The desired statistics are obtained even though the simulation program neither generates nor sorts any list.

Note that this VRT gives also a *simulation shortcut*; the simulation program has overall running time $\Theta(N)$ whereas a direct implementation of Quicksort requires $\Theta(N \log N)$ time. The use of conditional expectation for this problem was suggested by a similar application in Bentley's study of Hoare's selection algorithm [Bentley 1988, Chapter 15].

SUMMARY

Variance reduction techniques can provide a valuable tool for studying algorithms by simulation. This paper documents the power of variance reduction techniques for algorithm problems and provides tutorial discussion. Since even complex heuristic algorithms tend to have precise mathematical specifications and a great deal of structure, there is much potential for exploiting partial understanding of the underlying model. Furthermore, precision and clarity of results may be critical for development of the insight necessary for asymptotic analysis of functional growth.

The techniques applied in the search problem and some guides for their general application are summarized in the list below. Assume throughout that random variate X is an estimator of θ.

Common random numbers. When performance among two or more algorithms is likely to be positively correlated with respect to input instances, compare the algorithms using identical inputs.

Control variates. When a variate Y exists which is positively correlated with X and which has known expectation, adjust X towards its expectation θ according to deviations of Y from its expectation. Control variates may be thought of as well-understood sources of "noise" in the simulation.

Antithetic variates. When variate $\hat{X}$ can be generated which is negatively correlated with X but has identical distribution, generate pairs of antithetic variates and compute their means.

Conditional expectation. Whenever justified, replace the sampling of a random variate by the calculation of its expectation.

Simulation shortcuts. Exploit problem-specific knowledge to produce equivalent results more efficiently or to obtain many results from a single trial.

Splitting. When the simulation may be seen as a two-step process which first produces a "state" and then estimates the cost of that state, it may be cost-effective to allocate extra effort to the estimation problem for each state.

Stratification. Arrange input generation so that the frequencies of appearance of certain input measures agree with their expectations.

Poststratification. "Correct" an output measure towards its expectation according to variation in the distribution of the input.

REFERENCES

BEARDWOOD, HALTON, AND HAMMERSLEY. 1959. The shortest path through many points. In *Proceedings of the Cambridge Philosophical Society*, vol. 55. pp. 299–327.

BENTLEY, J. L. 1986. *Programming Pearls*. Addison-Wesley, Reading, Mass.

BENTLEY, J. L. 1988. *More Programming Pearls: Confessions of a Coder*. Addison-Wesley, Reading, Mass.

BENTLEY, J. L. 1990. Experiments on traveling salesman heuristics. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Data Structures and Algorithms* (San Francisco, Jan.). ACM, New York, pp. 91–99.

BENTLEY, J. L., JOHNSON, D. S., LEIGHTON, F. T., AND MCGEOCH, C. C. 1983. An experimental study of bin packing. In *Proceedings of the 21st Allerton Conference on Communication, Control, and Computing*. (Univ. of Illinois, Urbana-Champaign) pp. 51–60.

BENTLEY, J. L., JOHNSON, D. S., LEIGHTON, F. T., MCGEOCH, C. C., AND MCGEOCH, L. A. 1984. Some unexpected expected-behavior results for bin packing. In *Proceedings of the 16th Symposium on Theory of Computation* (Apr.). ACM, New York, pp. 279–288.

BENTLEY, J. L., JOHNSON, D. S., MCGEOCH, L. A., AND ROTHBERG, E. E. Near optimal solutions to very large traveling salesman problems. To be published.

BENTLEY, J. L., AND MCGEOCH, C. C. 1985. Amortized analysis of self-organizing sequential search heuristics. *Commun. ACM 28*, 4 (Apr.), 404–411.

BITNER, J. R. 1976. Heuristics that dynamically alter data structures to reduce their access time. Ph.D. thesis, Univ. of Illinois, Urbana-Champaign, Ill.

BITNER, J. R. 1979. Heuristics that dynamically organize data structures. *SIAM J. Comput. 8*, 1 (Feb.), 82–110.

BRATLEY, P., FOX, B. L., AND SCHRAGE, L. E. 1983. *A Guide to Simulation*. Springer-Verlag, New York.

CSIRIK, J., AND JOHNSON, D. S. 1991. Bounded space on-line bin packing: best is better than first. In *Proceedings of the 2nd ACM-SIAM Symposium on Algorithms and Data Structures* (San Francisco, Jan.). ACM, New York, pp. 309–319.

CHENG, R. C. H. 1986. Variance reduction methods. In *Proceedings of the Winter Simulation Conference* (Washington, D.C.). ACM/IEEE, New York, pp. 60–68.

DEGROOT, M. H. 1975. *Probability and Statistics*. Addison-Wesley, Reading, Mass.

DEVROY, L. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.

EPPINGER, J. 1983. An empirical study of insertion and deletion in binary trees. *Commun. ACM 26*, 9 (Sept.).

GAREY, M. R., AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco.

GONNET, G. H., MUNRO, J. I., AND SUWANDA, H. 1982. Exegesis of self-organizing linear search. *SIAM J. Comput. 10*, 613–637.

HAMMERSLEY, J. M., AND HANDSCOMB, D. C. 1964. *Monte Carlo Methods*. Wiley, New York.

JOHNSON, D. S. 1973. Near-optimal bin packing algorithms. Ph.D. thesis, Project MAC TR-109, Massachusetts Institute of Technology, Cambridge, Mass.

KERSHENBAUM, A., AND VAN SLYKE, R. 1972. Computing minimum spanning trees efficiently. In *Proceedings of the 25th ACM Conference*. ACM, New York, pp. 518–527.

KLEIJNEN, J. P. C. 1974. *Statistical Techniques in Simulation, Part I*. Marcel Dekker, New York.

KNOTT, G. D. 1975. Deletion in binary storage trees. Tech. Rep. STAN-CS-75-491, Ph.D. thesis, Stanford Univ., Stanford, Calif.

KNUTH, D. E. 1973a. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, Mass.

KNUTH, D. E. 1973b. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, Reading, Mass.

KNUTH, D. E. 1981. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2d ed. Addison-Wesley, Reading, Mass.

LAW, A. M., AND KELTON, W. D. 1982. *Simulation, Modeling and Analysis*. McGraw-Hill, New York.

LAWLER, E. L., LENSTRA, J. K., RINOOY KAN, A. H. G., AND SHMOYS, D. B. 1985. *The Traveling Salesman Problem*. Wiley, New York.

LUEKER, G. S. 1981. Optimization problems on graphs with independent edge weights. *SIAM J. Comput. 10*, 2 (May), 338–351.

MCGEOCH, C. C. 1986a. Experimental analysis of algorithms. Tech. Rep. CMU-CS-87-124, Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon Univ.

MCGEOCH, C. C. 1986b. An experimental study of median-selection in quicksort. In *Proceedings of the 24th Allerton Conference on Communication, Control, and Computing*. (Univ. of Illinois, Urbana-Champaign) pp. 19–28.

NELSON, B. L. 1987. A perspective on variance reduction in dynamic simulation experiments. *Commun. Stat. Simul. 16*, 2, 385–426.

ONG, H. L., MAGAZINE, M. J., AND WEE, T. S. 1984. Probabilistic analysis of bin packing heuristics. *Oper. Res. 32*, 5 (Sept.–Oct.), 983–999.

PAWLIKOWSKI, K. 1990. Steady-state simulation of queuing processes: A survey of problems and solutions. *ACM Comput. Surv. 22*, 2 (June), 123–170.

PAYNE, J. A. 1982. *Introduction to Simulation*. McGraw-Hill, New York.

RIVEST, R. 1976. On self-organizing sequential search heuristics. *Commun. ACM 19*, 2 (Feb.), 63–67.

SHOR, P. W. 1986. The average case analysis of some online algorithms for bin packing. *Combinatorica 6*, 179–200.

SLEATOR, D. D. K., AND TARJAN, R. E. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM 28*, 2 (Feb.), 202–208.

TENENBAUM, A. 1978. Simulations of dynamic sequential search algorithms. *Commun. ACM 21*, 9 (Sept.), 790–791.

WEIDE, B. W. 1980. Random graphs and graph optimization problems. *SIAM J. Comput. 9*, 3 (Aug.), 552–557.

WILSON, J. R. 1983. Variance reduction: The current state. *Math. Comput. Simul. 26*, 55–59.

WILSON, J. R. 1984. Variance reduction techniques for digital simulation. *Amer. J. Math. Manage. Sci. 4*, 277–312.

Received June 1991; final revision accepted May 1992.