INFORMS Journal on Computing
Vol. 12, No. 1, Winter 2000

0899-1499/100/1201-0024 $05.00

Statistical Analysis of Computational Tests of Algorithms and Heuristics

MARIE COFFIN AND MATTHEW J. SALTZMAN / *Department of Mathematical Sciences, Clemson University, Clemson, SC 29634-0975, Email: mcoffin@clemson.edu, mjs@clemson.edu*

(Received: July 1998; revised: November 1998, March 1999; accepted: November 1999)

Statistical analysis is a powerful tool to apply when evaluating the performance of computer implementations of algorithms and heuristics. Yet many computational studies in the literature do not use this tool to maximum effectiveness. This paper examines the types of data that arise in computational comparisons and presents appropriate techniques for analyzing such data sets. Case studies of computational tests from the open literature are re-evaluated using the proposed methods in order to illustrate the value of statistical analysis for gaining insight into the behavior of the tested algorithms.

A common problem in operations research involves comparing several algorithms or heuristics to determine which perform best and what factors affect performance. There are two basic approaches to understanding algorithm performance:

- Theoretical analyses focus on provable fundamental characteristics of the algorithms. Worst-case and average-case asymptotic analysis ("big-*O*" results) describe the limiting behavior of running time (in terms of fundamental operation counts) as a function of parameters describing problem size. Worst-case and average-case bounds or asymptotic results can be obtained for the quality of solutions obtained by heuristics. These results may also be parameterized by problem size or other characteristics.
- Empirical analyses involve implementing the algorithm or heuristic in a computer code and evaluating the actual running time, operation counts, or solution quality on selected instances of the problem.

These two approaches are complementary. Each analysis can provide insights to aid in carrying out the other, and both can provide insights into the characteristics of algorithms that may assist a decision maker in choosing the best tool for attacking a particular problem.

Often the results of computational studies are reported as tables of operation counts, CPU times, solution qualities, etc. that are used to support the authors' contentions. Publication standards suggest that statistical analysis of such results is appropriate, but few guidelines are given as to how the analyses should be performed. This paper explores some of the difficulties and subtleties of such analyses and demonstrates how a careful statistical treatment can lead to sometimes surprising and often stronger conclusions than a cursory examination would suggest. Our approach is case based: we will consider data sets from computational experiments published in the open literature, and we will show how careful statistical analysis can quantify the conclusions drawn in the papers.

An *algorithm* is a description of a mechanical procedure for carrying out a computational task. As observed by McGeoch,[48] a particular algorithm may be more or less instantiated, i.e., it may include more or fewer detailed descriptions of implementation features. Different levels of instantiation may be appropriate for different kinds of analyses. For example, a very general description may be enough to determine whether an algorithm completes its specified task or if it has a worst-case running time that is exponential in the length of the input. More instantiation may be required in order to count critical operations as a function of the input, and a complete instantiation (a *computer code*, implemented by a particular programmer, in a given language, using a specified compiler, and running on a particular machine) is necessary to generate CPU times for particular inputs. As the level of instantiation increases, more and more features must be specified in greater and greater detail. In a computer code, the specification is exact (possibly up to feature choices or parameter values that must be set by the user as part of the input for each particular run).

The optimization literature focuses on algorithms for computing solutions to constrained optimization problems. In this context, the term *exact* or *optimal algorithm* refers to a procedure that computes a provably optimal solution (and the term *algorithm* is often overloaded to mean *exact algorithm*). A *heuristic algorithm* (often shortened to *heuristic*) is an algorithm that is not guaranteed to produce an optimal solution. Heuristics may be *constructive* (producing a single solution) or *local search* (starting from one or more given or random solutions and moving iteratively to other nearby solutions) or a combination (constructing one or more solutions and using them to start a local search). An *approximation algorithm* has a guaranteed level of performance (in terms of solution quality or convergence), but a heuristic need not, in general, have such a guarantee. A *metaheuristic* is a framework for producing heuristics, such as simulated annealing or tabu search. To produce an actual heuristic for

Subject classifications: Algorithms and heuristics
Other key words: Statistical analysis, algorithms, heuristics

a particular problem from a metaheuristic, problem-specific characteristics must be defined. These may include the definition of a feasible solution, the neighborhood of a solution, rules for transforming solutions, and rules for setting certain parameters during the course of execution.

Empirical studies of algorithm performance generally rely on computer codes, which are the most complete instantiations of algorithms. The codes serve as concrete proxies for the abstract algorithms that they implement. In our case studies, we concentrate on measuring running time and solution quality of exact or heuristic algorithms for optimization problems. The techniques described are equally applicable to algorithms for problems from other areas and can be extended to other measures of algorithm quality, such as convergence rates.

Section 1 provides some pointers to the literature on algorithm analysis and considers some of the suggested criteria for algorithm comparison. Section 2 expands this discussion by outlining some of the choices that must be made in devising a comparative study. Some concrete guidelines for using various statistical techniques are provided in Section 3 and are illustrated in Section 4. Section 4 consists of detailed analyses of eight data sets available in the open literature. Section 5 offers a summary of our recommendations.

1. A Brief Survey of Statistical Analysis of Algorithms

The operations research literature devoted to the methodology of algorithm comparison is sparse and not widely distributed. Papers that actually compare algorithms are of highly variable quality. We attempt to survey the literature on statistical comparisons of algorithms and related work. Papers related to this subject are scattered widely throughout the literature, so undoubtedly we have missed some here. Such omissions are either unintentional or due to space considerations.

One of the earliest computational studies in the field of OR[32] appeared in 1953. Several presentations at a 1976 National Bureau of Standards conference on *Computers and Mathematical Programming*[67] addressed evaluation methods. A panel discussion held in conjunction with the track titled "Issues in the Evaluation of Mathematical Programming Algorithms" eventually led to two papers of Crowder et al.[13, 14] These papers, however, primarily address the issues of reproducibility and reporting standards for computational tests.

The controversial claims of Karmarkar[41] for the computational superiority of his interior-point LP algorithm over the simplex method prompted a flurry of self-examinations by the OR community regarding standards for reporting computational results. Notes by Jackson[36] and Shayan[63] eventually led to the articles by Jackson et al.[37, 38] and Greenberg,[24] among others. Drafts of articles written in conjunction with a 1996 DIMACS workshop[6, 40] address related issues.

Journals that accept articles on computation, such as *Operations Research*, the *INFORMS Journal on Computing*, *Mathematical Programming*, *ACM Transactions on Mathematical Software*, and the *Journal on Heuristics*, have published standards or guidelines for reporting on computational tests.[8, 13, 14, 24, 38, 57] These standards focus primarily on *what* is to be measured and compared in an article describing computational research. They address such issues as criteria for comparison and measures according to which the criteria are evaluated. While these papers mention the importance of careful experimental design, they (quite properly) do not address in detail the actual methodology for appropriate design and analysis. The kinds of questions that might be addressed in a computational study are varied enough that including such specific information in a publication standard would not be appropriate.

Of course, there are a huge number of papers that contain computational tests and analyses in support of some particular research hypothesis, i.e., those that are not about testing and analysis per se, but that apply (to a greater or lesser degree) the principles of empirical analysis. These are far too numerous to list here. We select several examples of data sets that have appeared in the open literature and consider how alternative analyses might reveal additional insights beyond those published.

The papers of most immediate interest here are those that directly address the problems of statistical analysis of algorithms. The papers discussed in this section either focus on or feature nontrivial statistical methods.

1.1 Analysis Methodologies

A review of early papers reporting computational test results was undertaken by Jackson and Mulvey.[39] They survey 50 published papers, technical reports, and theses from the period 1953–1977. They apply a 60-item questionnaire to each paper and break the results down according to a number of criteria. Included are various items addressing the reproducibility of results, plus items addressing statistical techniques used.

Hoffman and Jackson[33] attempt to develop guidelines for a software testing methodology. They discuss issues related to the definition of the experiment, selection of an appropriate measure, and warnings about confounding factors, selection of test problems (e.g., hand picked, randomly generated, or randomly perturbed), and a brief survey of experimental design. They also include data on 275 papers that report results of computational experiments.

The most comprehensive of the publication guidelines is given by Barr et al.[8] They provide reporting guidelines (including several detailed examples), emphasize formal statistical methods (though, again, not including details), and include a thorough treatment of time and quality metrics and tradeoff issues.

An excellent review of simulation and statistical analysis of algorithms by McGeoch[48] appeared as a feature article in the *INFORMS Journal on Computing* together with commentaries[43, 56, 64] and a rejoinder.[47] McGeoch includes a comprehensive set of basic references on appropriate statistical techniques plus general advice on designing and conducting experiments, including anecdotes and warnings of pitfalls. The commentary by L'Ecuyer[43] details issues with random number generators that are used to generate problems or

make randomization decisions in algorithms. It is our intention to build on the outline of McGeoch[48] by illustrating the application of statistical techniques to data from studies that have appeared in the open literature.

A number of papers attempt to apply statistical analysis to quantify the relationship between algorithm performance and various factors describing problem characteristics. Some are examples of linear regression analysis.[16, 21, 42] Two examples implement experimental design and analysis of variance techniques.[4, 5] Other methodological studies discuss visualization,[52] exploratory data analysis,[29, 65] and distribution-free hypothesis tests, and estimation of optimal values based on heuristic solutions.[23]

1.2 Simulation Methodology

McGeoch[46] provides a case study of algorithm analysis by simulation. She describes several techniques for reducing the variance of sample data generated by simulation. Her recommendations include some that are common among optimization case studies (e.g., comparison of algorithms on common problem instances) and others that are less well known (e.g., incorporation of control variates, antithetic variates, conditional expectations, splitting, stratification, and poststratification). In addition, she notes that the number of sample points generated in a given time can be increased if the results of algorithm runs themselves can be simulated.

1.3 Subjective Analysis

For a practitioner using a code to solve a real-world problem, there may not be a single criterion that dictates the choice of code. For example, robustness and usability may be as important or more important for some users than raw speed. Some researchers have proposed techniques for incorporating the decision maker's preferences into the evaluation process. Lootsma[44] and Golden et al.[22] describe applications of Saaty's analytic hierarchy process (AHP),[60] which develops priorities for codes based on the decision maker's pairwise comparisons among various criteria. Golden and Stewart[23] describe a technique for estimating a decision maker's expected utility for solution quality of heuristics using risk-aversion parameters supplied by a decision maker.

One difficulty with using subjective measures in research publications is that they are designed to take into account the preferences of the decision maker performing the evaluation. In a publication, the evaluator is usually the author and the designer/implementor of at least one algorithm under consideration. The author's subjective preferences may be (unintentionally) biased toward this algorithm and, in any case, are not likely to agree with the preferences of all readers. Thus, while there may be merit in suggesting that practitioners make use of these techniques when selecting methods for solving their particular problems, we cannot recommend their use in comparative studies designed for publication.

1.4 Comparison Criteria

Algorithms are often compared with respect to measured running times or solution quality. Other performance metrics, such as parallel speedups or efficiencies, function evaluation counts, etc. can be grouped with running times. Although our case studies are drawn from the optimization literature, running-time comparisons may be of interest for any algorithm at all. Typically (in optimization), running-time comparisons are applied to exact algorithms, although comparison of running times of approximation algorithms and heuristics is appropriate when the methods under consideration produce solutions of similar quality or when solution quality is a controllable stopping criterion. Solution-quality comparisons are generally appropriate only for approximation and heuristic methods (although, to the extent that solution quality matters for iterative equation solvers, these kinds of analyses also may be appropriate for them).

The statistical methods described in the case studies are appropriate for analysis of all kinds of algorithms, whenever running-time or solution-quality comparisons are of interest. The basic ideas can also be extended to other criteria of interest, including (for example) empirical convergence-rate comparisons or models of the trade-off between solution quality and running time.

1.4.1 Solution Quality

When choosing a measure of solution quality, some care must be taken since intuitively appealing measures can produce misleading results. Zemel[68] observes that many of the commonly used measures of relative solution quality for 0–1 programming problems are not invariant with respect to scaling or several other transformations that yield equivalent problems. For example, the common measure of relative quality $(z^* - z)/|z^*|$ (where z^* is the optimal value of a maximization problem and z is the value of a heuristic solution) fails this invariance test for many problems. In such cases, transforming the data can change the outcome of the statistical analysis. Choosing a different measure of solution quality avoids this problem.

Zemel defines a class of transformations under which problems are "obviously 'equivalent,' but may appear different when viewed superficially." He then proceeds to derive conditions on quality measures that make them invariant with respect to these transformations. Constructing such measures for particular problem classes requires a certain amount of cleverness but avoids many difficulties with solution-quality analyses.

1.4.2 Speed

By far the most common measure of algorithm speed is the CPU time required for a particular code to solve a problem. Many studies include head-to-head comparisons of running times of two or more codes on a common set of problems, designed to demonstrate the superiority of the method proposed by the study's author. Despite their popularity, these kinds of studies have been criticized on a number of grounds, including variability of results across instantiations (different programmers, languages, compilers, and machines can affect the outcome) and the failure of many such studies to offer insight into the relationship between problem characteristics and algorithm performance. Various at-

tempts have been made to offer alternative measures that can clarify these relationships.

Miele and Gonzalez[49] study criteria for comparing speeds of algorithms for unconstrained optimization. In an attempt to make the measure machine-independent while accounting for function, gradient, and Hessian evaluations and overhead costs, the authors propose a normalized evaluation count that they call the *time-equivalent number of function evaluations*. They observe that when making machine-independent comparisons using function evaluation counts, certain naïve assumptions that are often made about the relative costs of functions, gradients, Hessians, and overhead are generally unjustified. Instead, empirical estimates of these values should be used.

Shayan[63] proposes an implementation-independent comparison criterion based on the number of elementary arithmetic operations performed. Unfortunately, the method appears to be impractical.

Ahuja et al.[1] and Ahuja and Orlin[2] describe a refinement of the operation count criterion, called *representative operation counts*, in which certain operations that occur at key points in the algorithm are counted. The operations are selected so that CPU time on any reasonable machine is bounded by a constant multiple of the operation count. Regression models are used to identify bottleneck operations and to construct a *virtual running time*.

Sedgewick,[62] in a tour de force of theoretical algorithm analysis using operation counts, provides an analysis of the Quicksort algorithm (Hoare[30]).

Hooker[35] criticizes a common type of analysis—the competitive comparison of two methods based on total CPU time. He champions a more sophisticated analysis that seeks to understand not just which method is better on some set of problem instances but what properties of the methods or the test problems influence the performance of the methods being analyzed. He also recommends against measuring CPU seconds, suggesting instead that other structural measures of computational effort will provide more insight.

We note that, while the above critiques of CPU time as the principal basis for comparison of algorithms are valid, CPU time still represents the most common comparison criterion. In some cases, CPU time may even be the most appropriate comparison criterion. This is likely to be so, for example, when the algorithms being compared have significantly different architectures and do not have obviously comparable fundamental operations (e.g., combinatorial methods versus LP-based integer programming methods). It is also quite possible to construct experiments using CPU time that do provide insights into the relationship between performance and problem characteristics. Nevertheless, it is important to look beyond CPU time when other criteria can be used reasonably.

2. Dimensions of Analysis and Statistical Considerations

We advocate a general procedure for conducting a computational analysis, outlined as follows. The experimenter wishes to compare several algorithms or heuristics and make recommendations as to which should be used, when, and why. This requires gathering data. *Experimental design* refers to the plan the experimenter uses to gather the data. Some common experimental designs are *completely randomized*, *randomized block*, *factorial*, and *fractional factorial*[28]. When all the data have been gathered, an exploratory analysis is done—making graphs of the data, looking for patterns and trends, and noting how means and variances change across treatment groups. These observations may suggest that the data should be transformed before further analysis is done (see below). Finally, a formal statistical analysis is performed. Typically, this will involve testing hypotheses (for example, the null hypothesis that all algorithms/heuristics behave the same way against the alternative that some differences exist), estimating parameters, and calculating confidence intervals.

One commonly used sequence (illustrated in the case studies that follow) is to choose a model consistent with the experimental design, test for the presence of any interesting effect, and then calculate confidence intervals for the size of the effects found. Other sequences can also be used. For example, some statisticians advocate *interval-based* analysis, in which formal hypothesis testing is avoided as much as possible. This approach has many advantages. However, in the case of a factorial experimental design, a few preliminary hypothesis tests can save a great deal of work later.

When a hypothesis test is performed, it is appropriate to report the *p-value*, which is the probability of obtaining a test statistic at least as far as the observed value from the value specified in the null hypothesis, calculated under the assumption that the null hypothesis is true. (In the case of a *one-sided* alternative hypothesis, only values consistent with the alternative hypothesis are used to calculate this probability.) Loosely speaking, smaller p-values indicate that the data are inconsistent with the null hypothesis. Thus, reporting the p-value tells the reader how much evidence against the null hypothesis was provided by the data. Just how small a p-value should be before the null hypothesis is rejected is somewhat subject-specific. In fields of study where large amounts of random variability are expected, criteria for rejecting the null hypothesis may be less stringent. In addition, when sample sizes are quite small, hypothesis tests have less power, and thus some experimenters require less evidence in such cases. As these circumstances seldom pertain in computational testing, the traditional *critical values* of 0.10, 0.05, or 0.01 seem reasonable. For more details on this and other statistical concerns, the reader is referred to Box et al.[11] and Neter et al.[55] An extremely readable and friendly introduction to statistical practice is provided by Moore and McCabe.[51]

Empirical analyses of algorithms can be distinguished along three dimensions. When designing a study, it may be helpful to give some consideration to each dimension and the corresponding statistical recommendations.

2.1 Running Times vs. Solution Quality

When measuring running times or some proxy (such as operation counts), some statistical considerations are necessary. The population of running times is unlikely to be normally distributed, so analyses that rely heavily on nor-

mality are often inappropriate. Many analyses are robust to nonnormality when sample sizes are large; thus, planning to compare a large enough number of problems is often wise (but see Section 3.2). For the kinds of nonnormality often seen in running times, samples in the range of 30–40 observations per treatment group should be sufficient. In addition, running times should be expected to display nonconstant variance (heteroskedasticity), which is easily detectable from a graph. Usually the variance will increase as the running time increases, due to the fact that times are bounded below at zero; that is, the shorter the running time, the less room there is for variability. In such cases, methods of analysis that rely on constant variance should be avoided. A transformation of the running times (e.g., log transformation) often improves this situation.

In choosing a metric for solution quality, it is worthwhile to consider the statistical properties as well as the intuitive appeal. A metric that results in normally distributed (or at least not too badly skewed) observations with equal or nearly equal variances is obviously desirable.

2.2 Randomly Generated Problems vs. Library Test Sets

Comparisons can be based on real-world problem instances (which Greenberg[24] refers to as "library analysis") or on problems generated at random over some space of problems. Greenberg terms the latter "statistical analysis," although a better term might be "simulation." It is our contention that statistical analyses using library data sets are both feasible and desirable. Furthermore, many of the available problem libraries consist of collections of randomly generated problems. They are libraries simply because they have come to be shared among researchers as a way of standardizing the analyses.

One type of algorithm comparison involves randomly generating a set of problems from some larger domain, where some problem parameter (or set of parameters) is varied. Generally, the best experimental design is to randomly generate *one* set of n problems for each combination of parameter values, so that all algorithms are compared on the same set of problems. This reduction in problem-to-problem variability will make any differences among algorithms more obvious. The resulting data are properly analyzed using an analysis of variance (or possibly mixed model) technique, as will be demonstrated below. The term *mixed model* refers to the statistical analysis of a linear model containing both categorical and continuous explanatory variables (Case Study VII is an example of a mixed model).

Algorithms can also be compared on library test sets such as the NETLIB set.[20] Such sets are generally not well-parameterized and should be analyzed somewhat differently than data from randomly generated problems with controlled parameter values. If several algorithms are to be compared, the problems themselves should be treated as blocks, the algorithms as treatments, and the data should be treated as a complete block design with no replication. In the literature, such data are often treated as independent samples (by calculating total or average solution times for each algorithm, for example); this analysis is incorrect and often obscures the true nature of the results. Some appropriate analyses for these data are analysis of variance for complete block designs, analysis of means for complete block designs,[53] and Friedman's nonparametric test. Some analyses may be more powerful than others, as the case studies in Section 4 will illustrate.

If only two algorithms are compared, the complete block design reduces to a paired-sample design, which can be analyzed using the paired-sample t-test, the paired-sample exact test, the sign test, or the signed rank test. It also may be possible to parametrically model the results from one algorithm as a function of the other. When this seems plausible, doing so will generally provide more information than is given by hypothesis testing alone (see Sections 2.3 and 2.4).

2.3 Head-to-Head vs. Parameterized Comparisons

Algorithms can be modeled with problem parameters or functions of problem parameters (e.g., problem size) as independent variables. Alternatively, different algorithms or heuristics can be compared head to head. For example, one might fit a regression model where one variable represents the running time of the old algorithm and the other represents the running time of the new algorithm. In this case, the designation of independent and dependent variables is somewhat arbitrary, but it will often be helpful to use the new algorithm as the dependent variable so that its running times can be expressed as a function of the old.

When the problems can be distinguished by problem parameters or functions of problem parameters, these should be treated as independent variables in the analysis. The different algorithms or heuristics can be indicated with dummy variables, which will also be used as independent variables in the model. If a linear (in the statistical sense) model seems appropriate, the analysis may be multiple regression, analysis of variance, or analysis of means. Robust versions of multiple regression or analysis of variance should be considered if strong outliers are likely.

2.4 A Complexity-Theoretic Interpretation of Regression Models of Running Time

Regression models of running time can be thought of as empirical models of average complexity (at least with respect to the particular problem instances tested). In this section, we provide an informal justification for this interpretation.

Consider an algorithm with expected running time $T(n) = O(n^k)$, where the input length is parameterized by n. A functional form representing this situation is

$$T(n) \approx Cn^k.$$

If C and k are unknown, we could consider estimating them using regression on the log-transformed model:

$$\ln T(n) = \ln C + k \ln n + \epsilon.$$

Note that conditions affecting the constant factor of running time are loaded into the intercept of the regression model. For example, the same algorithm run on the same test problems on different machines should produce the same estimate of k, and the ratio of values of C should capture the

relative speed of the machines. We refer to the estimated function $O(n^{\hat{k}})$ as the *empirical complexity* of the algorithm.

Two examples of this sort of analysis are given by Vanderbei.[66] He describes the running time of variants of the simplex method for linear programming as a function of the number of rows and columns in the LP, based on a sample of NETLIB problems.

Head-to-head comparisons can be interpreted similarly, but a bit more subtlety is required. Suppose ALGORITHM A has expected running time $T_A(n) = O(n^p)$ and ALGORITHM B has expected running time $T_B(n) = O(n^q)$. Let $f(t)$ be a function of time with the property that $T_B(n) = O(f(T_A(n)))$. Then in order for $T_B(n) = O(f(T_A(n))) = O(n^q)$ to hold, $f(t) = O(t^{q/p})$. The regression model for estimating $k = q/p$ from the running times of algorithms A and B on a set of test problems is

$$\ln T_B(n) = \ln C + k \ln T_A(n) + \epsilon.$$

We refer to $O(t^k) = O(t^{q/p})$ as the *relative complexity* of ALGORITHM B with respect to ALGORITHM A and the estimated function $O(t^{\hat{k}})$ as the *empirical relative complexity* of B with respect to A. We call $\hat{k}$ the *empirical relative complexity coefficient*.

Again, conditions that affect the constant factor are loaded into the intercept of the regression line. For example, the same algorithm run on the same test set on two different computers should yield $k = 1$, and the relative speed of the two machines would be captured by the value of C. The value of k captures the relative growth in running time of B as the running time of A increases. If $k < 1$, then the running time of B grows slowly with respect to that of A and, for large enough running times of ALGORITHM A, ALGORITHM B will be faster.

Experimenters who find themselves evaluating the performance of new algorithms against the results of historical tests or against tests of codes or on machines that are not available can still use the head-to-head model to derive useful conclusions about the relative growth rates, with the irrelevant machine differences factored out.

As with any regression analysis, the general form of the model has been assumed, and only the parameters are subject to estimation. Similar models can be derived for other running time functions. For example, if $T(n) = O(k^n)$, then the log transformation produces the regression model:

$$\ln T(n) = \ln C + n \ln k + \epsilon.$$

A final caveat: the usual assumptions of regression analysis must be satisfied by the transformed model. The estimated values of the parameters are, of course, only as representative as the problems in the test set.

3. Further Statistical Considerations

Several statistical issues relevant to problems in algorithm analysis are now discussed. In addition, the distinction drawn by statisticians between exploratory and confirmatory analyses may also affect the choice of models and techniques to be applied.

3.1 Experimental Units

For conclusions based on a statistical analysis to be valid, it is necessary to identify the experimental unit and to analyze results in terms of the unit. An experimental unit is often defined as the unit to which treatment is applied. When analyzing experiments on heuristics and algorithms, the experimental unit will often be a single computer run. For example, if an experiment involves solving five different randomly generated problems at each of four different values of n, the experiment has 20 experimental units. To see if n has an effect on the outcome, it is necessary to analyze all 20 outcomes. It would be incorrect to calculate four sample means and treat these as four observations. Doing so completely ignores the problem-to-problem variability, which a correct statistical analysis would have to consider. Analyzing means rather than observations will artificially inflate the statistical significance of the treatments being considered.

3.2 Sample Size Considerations

Researchers sometimes have the impression that statistical analysis can only be performed if a great deal of data has been collected. This is simply not true. Not only can statistical analysis be done on moderately sized sets of data, but the results of such an analysis are often at least as convincing and useful as analyses of very large data sets.

In planning an experiment, it is indeed important to collect enough data. If very few data points are observed, hypothesis tests will have low power, and confidence intervals will be uninformatively wide. It will not be clear if the resulting failure to reject the null hypothesis is due to the genuine plausibility of the null hypothesis or if the conclusion simply reflects the lack of information in the experiment. In addition, if only a few observations are taken of severely nonnormal data, many traditional statistical analyses are precluded. For these reasons, researchers are often cautioned to take enough data.

However, when planning the sample size for a factorial experiment, the experimenter should keep the goals of the analysis in mind. Often the goal is to distinguish those factors and interactions which have large and important effects (and thus deserve more attention) from those whose effects are less noteworthy. In the analysis of heuristics and algorithms, there is no monetary cost to gathering data. When hundreds or thousands of observations are taken, almost every null hypothesis will be rejected, and every factor and interaction will be deemed "highly significant." In effect, huge amounts of data make it possible to distinguish very small treatment effects that have no practical significance. A better experimental plan would be to decide what size of effect the experimenter wants to detect and choose a sample size sufficient to detect that magnitude of effect with high power. The textbook by Neter et al.[55] provides an excellent set of tables for planning such experiments. When the first analysis has detected a few important factors/interactions, a second experiment can be planned (if necessary) to provide more precise estimates of effects. This two-stage approach often uses fewer total experimental runs, provides data that are more easily organized for anal-

ysis, and yields more useful results than one massive experiment alone can provide.

3.3 Censoring

Censoring is a statistical term that refers to observations that are lost or incomplete. Often, such observations are simply ignored in the ensuing analysis. Sometimes this is the only thing to be done, but in other cases censored observations can offer insight and should not be disregarded.

Failures of memory and failures to converge should be noted as their occurrence may in itself argue against the algorithm in question, but not much more can be done about them. Subsequent analysis should be performed only on the complete part of the data. Failure to terminate in the allotted time is sometimes treated differently. When we can assume that, given enough time, a solution would eventually be found, we have partial information (a lower bound) on the solution time. This partial information can be used in a modified analysis of variance[54] or regression analysis[12]. Similar observations apply when comparing solution quality.

3.4 Traditional vs. Robust Analyses

The statistical literature contains many traditional methods such as the analysis of variance *F*-test and least-squares regression as well as robust alternatives such as Friedman's test, L_1-regression, etc. Often it is not clear to the practitioner which methods are preferable for which kinds of data and why.

3.4.1 Analysis of Variance Methods

It is widely believed that constant variance is necessary for the usual analysis of variance methods to apply and that, when heteroskedasticity is detected, a nonparametric method such as Friedman's test should be used instead. This is wrong on two counts: 1) nonparametric tests, in fact, assume equal variances (in fact, they assume distribution functions that differ only in location[34]) and 2) the usual analysis of variance is extremely robust to unequal variances, provided the number of observations is approximately the same in each treatment group.[50, 55] In addition, the analysis of variance *F*-test can be used with any number of factors, whereas the nonparametric alternatives are restricted to at most two factors. Thus, when nonconstant variances are expected, the practitioner should plan to have the same (or nearly the same) number of observations per treatment group and use the *F*-test. The test can be made more powerful by finding a variance-stabilizing transformation of the data.

Friedman's test is appropriate when the following criteria are all met:

- The data show strong evidence of nonnormality (e.g., strong outliers).
- The sample sizes are not large.
- A transformation to approximate normality cannot be found or is undesirable.

In addition, Friedman's test is appropriate in the (relatively rare) instances where the responses cannot be compared directly but can only be ranked from best to worst. Case Study VI is an example of this.

If the experiment contains more than two factors, it will have to be divided into two-factor subsets and Friedman's test will have to be applied separately to each subset. This increases the experiment-wise error rate above the nominal α-level. An upper bound on the experiment-wise error rate (the probability of Type I error for the entire experiment) should be reported.

The analysis of means test is an alternative to the analysis of variance test. It is often more powerful when most of the treatment groups show a similar result, with one or two groups that differ widely from the others.

3.4.2 Two-Sample Comparisons

The *t*-test is the two-sample version of the *F*-test, and the various rank tests (sign test, signed rank test, etc.) are (or are similar to) two-sample versions of Friedman's test. Thus, the same considerations listed in Section 3.4.1 apply.

In some situations, either the sign test or the signed rank test may be more powerful than the *t*-test (for example, when most observations are clustered near zero but a few observations are large). See Case Study I for an example of this.

3.4.3 Regression Methods

A variety of robust regression methods are available. Here again, robust methods are not necessary for unequal variances but may be indicated if strong outliers are present. Most robust methods disregard or downplay the influence of outliers. Some popular methods are *least absolute residuals* (also called *minimum L_1-norm regression*)[59] and *rank regression*[27].

If heteroskedasticity is obvious, some care should be taken in performing hypothesis tests and constructing confidence intervals with the regression model, and prediction intervals should be avoided. If such inference is desired, weighted least squares should be used.

4. Case Studies

To illustrate the methods and pitfalls discussed in Section 3, we develop several case studies based on problems and data sets from the open literature. These data sets were chosen to demonstrate a variety of statistical techniques as well as for historical interest, general applicability, and the completeness with which the data were reported. Our intent is to improve and strengthen the original conclusions rather than to find fault with them.

Case Study I: Paired Comparisons

Golden et al.[22] compare the running times of two nonlinear optimization packages (GINO and OPTISOLVE) run on a set of 35 test problems provided by Hock and Schittkowski[31]. The observed running times varied from 16 seconds to 2100 seconds. On several problems, one or both packages failed to converge, while on two others internal errors terminated one program before a solution was found. Golden et al. analyzed these data using Saaty's analytic hierarchy process. Standard statistical techniques will be used here.

A scatterplot of solution times is shown in Fig. 1, with a

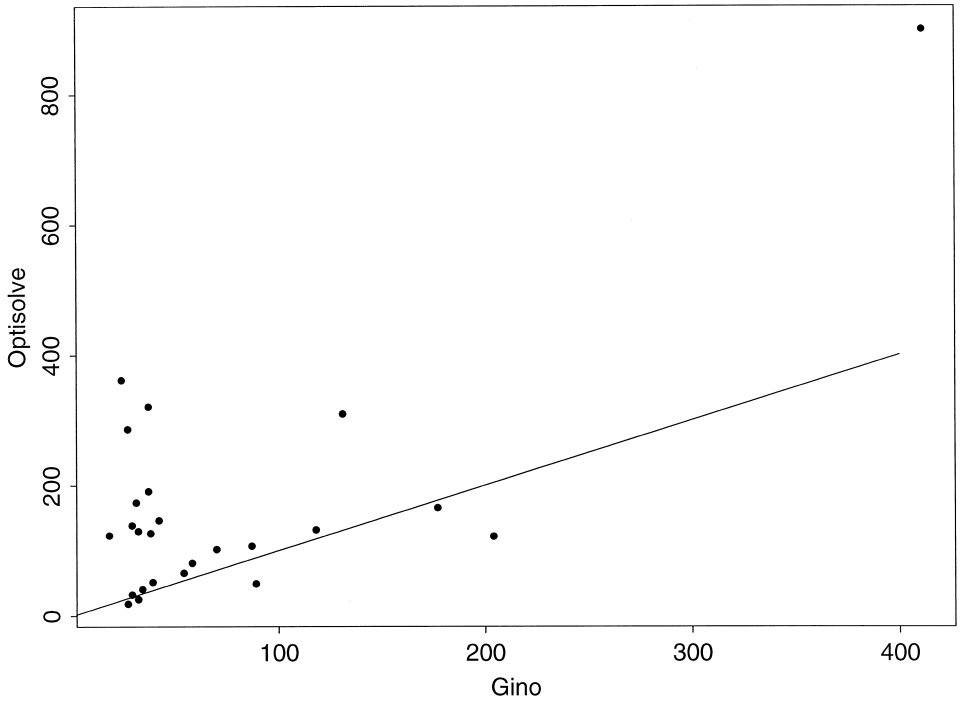

Figure 1. Scatterplot of OPTISOLVE vs. GINO solution times.

45° line for reference. (Problems for which one or both algorithms failed to find the optimal solution are omitted; thus, 25 of the original 35 problems are displayed.) Neither the scatterplot of times nor the scatterplot of log-transformed times exhibits any obvious trend, so regression analysis is not appropriate (see Fig. 2 also). Most of the points lie above the 45° line, indicating that OPTISOLVE is generally slower. To demonstrate this, a paired-sample test is appropriate. The paired-sample t-test, sign test, or signed rank test could be used. Because the sample size is not large and the differences (see below) are severely skewed, the sign test was chosen rather than the paired-sample t-test. (The signed rank test could also have been used.) The pairwise differences in solution times calculated as $d_i = y_i - x_i$, where y_i = OPTISOLVE solution time on the i-th problem and x_i = GINO solution time on the i-th problem, are

$$d = (-6, 34, 9, -13, 1, 107, 225, 20, 92, 330, 115, 98,$$
$$75, 70, 49, 90, 142, 278, 289, 95, 134, 17, 100, 159, 868).$$

The sign test is a distribution-free hypothesis test for the population median where the test statistic is based on counts of positive and negative values. Let M be the population median of pairwise differences. Then $M = 0$ indicates that on a randomly chosen test problem OPTISOLVE is just as likely as GINO to perform more quickly, whereas $M > 0$ implies that OPTISOLVE is likely to take longer and $M < 0$ implies that GINO takes longer. Because we have no *a priori* reason to suppose either algorithm is faster, we will test H_0:$M = 0$ versus $H_a : M \neq 0$. The test statistic is $S = \Sigma_{i=1}^{25} I(d_i < 0) = 2$, which has a p-value of 1.05×10^{-5}. (Note: $I(A)$ is 1 if A is true and is 0 otherwise.) We reject H_0 and conclude that the algorithms perform differently. Since the test statistic is extremely small, it seems that GINO performs more quickly than OPTISOLVE, at least on problems where an optimal solution is found. GINO failed to find the optimal solution on four problems, while OPTISOLVE failed on five and experienced internal errors on two others. Thus, it appears that GINO is the preferred package. A 95% confidence interval for M (based on the binomial distribution) is (49, 115), indicating that the median difference in solution times is between 49 seconds and 115 seconds.

The scatterplot shown in Fig. 1 does not show a linear trend. A scatterplot of log-transformed running times shows that some points seem to follow a linear trend while a cluster of others do not. These are all problems on which OPTISOLVE performs very poorly. This observation should not be given too much weight since any scatterplot can be made to look linear by removing enough data points. However, it is possible that more insight into the differences between the two programs could be obtained by examining the problems that don't follow the trend in more detail.

Case Study II: Log-Linear Regression

Lustig et al.[45] describe an implementation of a primal-dual Newton barrier algorithm for linear programming. The paper includes an extensive set of computational comparisons of various features of the algorithm plus a comparison of the authors' code (OB1) with a well-known simplex code (MINOS 5.3). We consider an alternative analysis here and later use the data to demonstrate a method for analyzing data in the face of censoring.

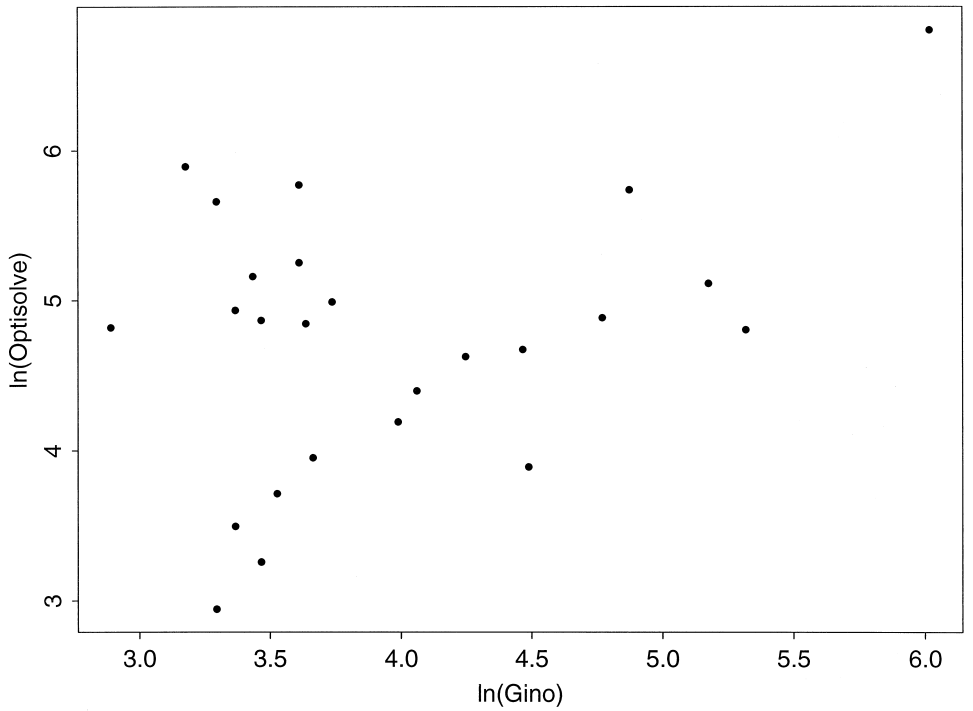

Figure 2. Scatterplot of log-transformed OPTISOLVE vs. GINO solution times.

The authors model ln(ratio of execution times) as a function of problem size (defined as the total number of rows and columns). The fitted model is

$$\hat{y} = 0.0180x^{0.5032},$$

where x = (rows + columns) and y = ln(MINOS/OB1), which has an $r^2 = 0.273$ (incorrectly labeled "correlation"). This model has several problems: 1) it does not allow the reader to compare the two algorithms in any obvious way, 2) on a set of library test problems which are dissimilar in many ways, problem size = rows + columns does not seem to be an effective parameterization, and 3) the low value of r^2 makes the conclusions unconvincing.

Here we consider a somewhat different analysis that compares the two algorithms head to head. A scatterplot of the original data is shown in Fig. 3. Note the pattern of increasing variance in the plot. This is to be expected when time measurements span an order of magnitude or more. The lower bound of zero means that as problem size increases, the possible variation in solution times increases as well. Although there appears to be some relationship between solution times for the two methods, the form of the relationship is not clear from the scatterplot. A possible model that is both simple and sensible is

$$y = \beta_0 x^{\beta_1}\epsilon, \tag{1}$$

where y = (OB1 running time), x = (MINOS running time), and ϵ represents the random error. The model assumes that the errors are independent and identically distributed with mean 0 and variance σ^2. Model (1) implies that:

- When MINOS = 0, OB1 = 0 as well.
- Assuming β_0 and β_1 are positive, as times for the MINOS algorithm grow, times for the OB1 algorithm will grow also.
- As times for the MINOS algorithm grow, the variance of the OB1 times grows in proportion.
- $\beta_1 = 1$ implies that the two algorithms have times that are approximately proportional, with proportionality β_0.

An additional advantage to the model is that it is log-linear:

$$\ln y = \ln \beta_0 + \beta_1 \ln x + \ln \epsilon, \tag{2}$$

which implies that ln y and ln x are linearly related, with an *additive* error structure and constant error variance. Thus, a log transformation is performed. (See Section 2.4 for further discussion of this model.) Fig. 4 is a scatterplot of the log-transformed solution times. It shows that ln MINOS and ln OB1 appear to be linearly related with a few possible outliers, which have been labeled. Using ordinary least squares, the fitted model is

$$\widehat{\ln y} = 1.18 + 0.7198 \ln x.$$

Recall from Section 2.4 that the slope of the regression line is the empirical relative complexity coefficient of OB1 with respect to MINOS. If the slope is less than one, the running time of OB1 is asymptotically smaller than the running time of MINOS, although for short problems this relationship may not hold. To see if the two algorithms have different relative complexities, we perform the hypothesis test H_0: $\beta_1 = 1$ versus H_a:$\beta_1 < 1$. Note that the alternative hypothesis here is

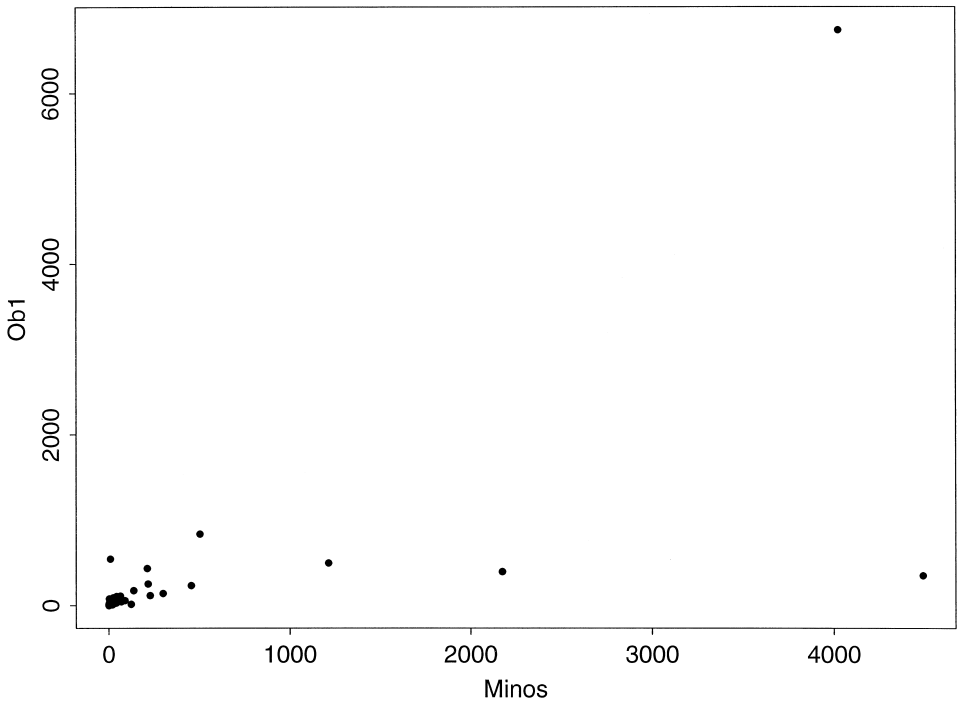

Figure 3. Lustig, Marsten, and Shanno data.

one-sided because Lustig et al.[45] claim that OB1 is *faster* than MINOS. This test requires either a large sample or approximate normality of the residuals, both of which apply here. (Since the sample size is 71, the normal probability plot is omitted.) This test has a p-value less than 0.0001 leading us to conclude that OB1 is asymptotically faster than MINOS. By transforming the ordinary least-squares (OLS) fitted equation, we obtain the model

$$\hat{y} = 3.25x^{0.72},$$

where $\hat{y}$ represents the predicted running time of OB1 on a problem whose MINOS running time is x. This indicates that for short problems, OB1 times will be about 3.25 MINOS times, while for long problems, OB1 times grow as (MINOS times)$^{0.72}$. The crossover point is approximately $\exp(\hat{\beta}_0/(1 - \hat{\beta}_1)) = 67.6$ CPU seconds.

Case Study IIB: Regression on Censored Data

In comparing running times of computer codes, it sometimes happens that a code will fail to finish in a reasonable amount of time on some problem instances. In such cases, the time at which the test was terminated is called the *censoring time*. This time provides partial information on the algorithm and can be used to fit a distribution-free regression line and make inferences on the population parameters. To illustrate this and to facilitate comparison with uncensored estimates, we have censored the Lustig et al.[45] data set at 100 CPU seconds. The limit of 100 seconds produces a moderate amount of censoring (approximately 20%).

The presence of censored observations makes ordinary least-squares regression unsuitable. The slope and intercept estimates (in this case, of the log-transformed data) must be estimated differently to account for the censoring (see Akritas et al.[3] and Coffin and Saltzman[12] for formulas and computational details).

At 100 CPU seconds, 12 observations have censored x-coordinates (MINOS values), 14 observations have censored y-coordinates (OB1 values), and 11 of these observations are censored in both x and y. The estimated regression line is

$$\widehat{\ln y} = 0.918 + 0.691 \ln x$$

or

$$\hat{y} = 2.5x^{0.691}$$

As would be expected, these values vary slightly from those calculated on the complete sample, but would lead us to approximately the same conclusions.

For testing

$$H_0{:}\beta_1 = 1 \text{ vs. } H_a{:}\beta_1 < 1,$$

the p-value is 0.008, again leading us to reject H_0. The empirical relative complexity coefficient of OB1 with respect to MINOS is less than 1. This test assumes the residuals are symmetric about zero.

To test the robustness of the distribution-free censored inference procedure, other levels of censoring were tried as well. Censoring times above 30 CPU seconds all led to the same conclusions (point estimates differ slightly). Censoring times below 30 CPU seconds censored so many observations (21 in x, 25 in y) that the difference from H_0 could no longer be detected. Thus, regression inference with censored obser-

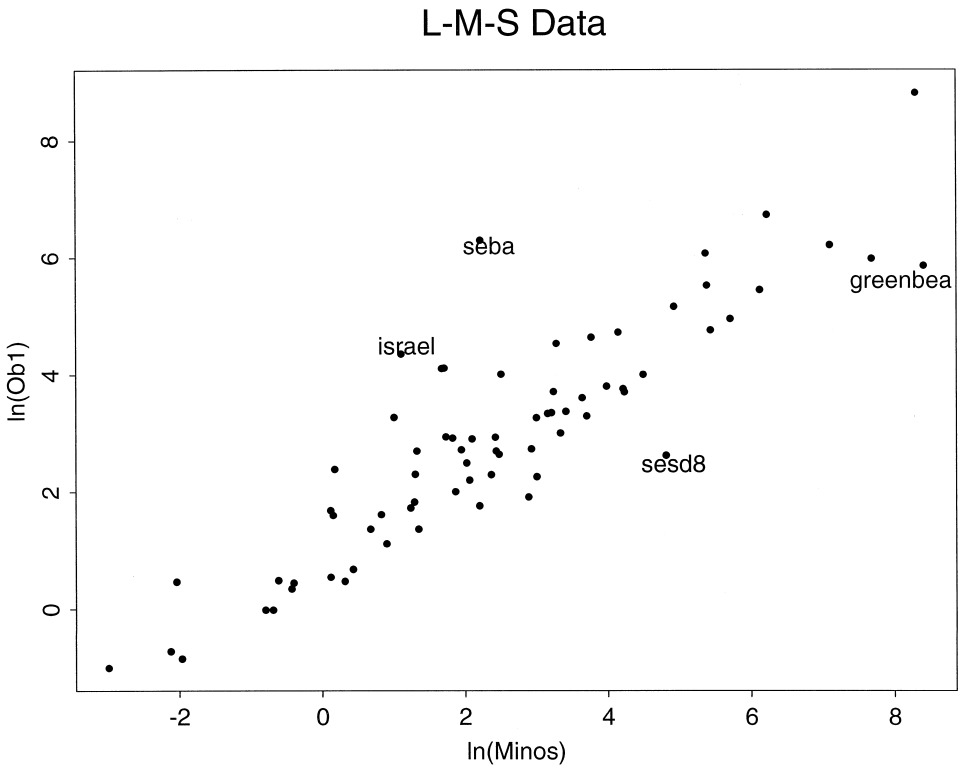

Figure 4. Lustig, Marsten, and Shanno data, log-transformed.

vations seems safe and effective, provided the censoring is not extreme.

Case Study III: Log-Linear Regression with Outliers

Fourer and Mehrotra[19] consider an augmented-system approach for solving least-squares problems in an interior point method for linear programming. The authors explore the suitability of the augmented-system approach for least-squares problems with dense columns by running both the normal equations and the augmented-system solution on the NETLIB problems. From a cursory examination of the data, the authors conclude:

> [T]he augmented-system approach achieves a predictably great advantage in solving those problems (FIT1P, FIT2P, ISRAEL, SEBA) that are well known to have dense columns. We also observe substantial improvement in another group of problems (AGG, CAPRI, D2106C, GANGES) whose dense columns are not so readily identified. For other problems the normal equations are acceptably sparse, and hence the augmented system is more expensive to solve; even so, performance is rarely worse by more than a factor of 2.

Here we present a more detailed analysis of the Fourer and Mehrotra data that leads to stronger conclusions.

A scatterplot of AUGMENTED versus NORMAL CPU times is given in Fig. 5. As in the previous case study, the scatterplot shows increasing variance, indicating that a log transformation may be needed. Fig. 6 shows a scatterplot of ln AUGMENTED versus ln NORMAL.

The scatterplot looks quite linear (and the error variance appears constant) with the exception of the three data points marked with open circles. These data points are (left to right) AGG, ISRAEL, and SEBA, which are all problems with very dense columns. Note that on the original scatterplot, these points (also marked with open circles) are not obviously outliers.

The three outliers are removed, and a log-linear model is fit to the remaining data points. This is the same model used in Case Study II, with y = (AUGMENTED CPU time) and x = (NORMAL CPU time). The fitted model is

$$\widehat{\ln y} = 0.42 + 0.97 \ln x,$$

from which we can obtain:

$$\hat{y} = 1.53x^{0.97}$$

(which is not an OLS estimate but a transformation of the OLS estimate from the previous equation). This indicates that AUGMENTED is approximately 1.5 × NORMAL, except in those cases of very dense columns. This is a somewhat different conclusion than that of Fourer and Mehrotra.

The test H_0: $\beta_1 = 1$ versus H_a: $\beta_1 < 1$ has $p = 0.033$. This provides some evidence that the empirical relative complexity coefficient of AUGMENTED with respect to NORMAL is less than one. However, the model implies that AUGMENTED does

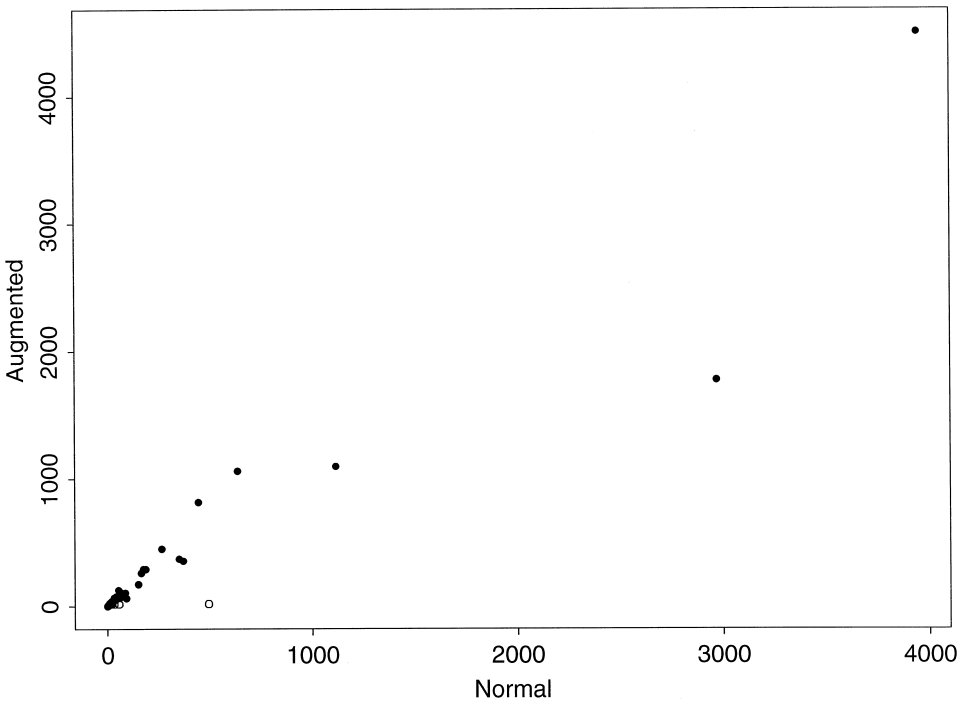

Figure 5. Fourer and Mehrotra results.

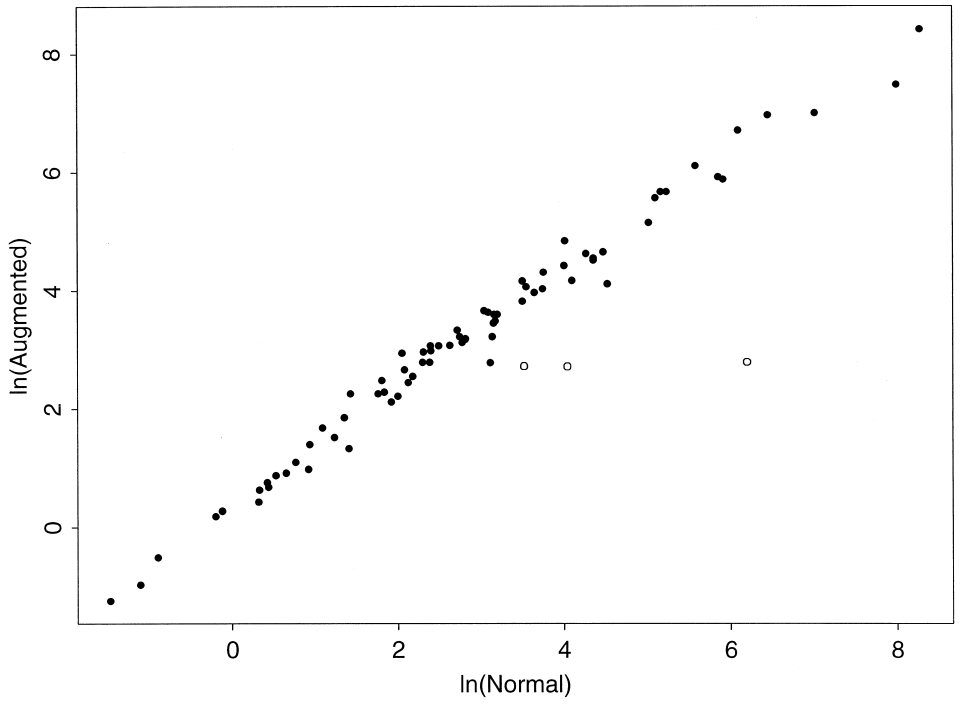

Figure 6. Fourer and Mehrotra, log-transformed results.

not overtake NORMAL until computation time exceeds $\exp(\hat{\beta}_0/(1 - \hat{\beta}_1)) = 4.8 \times 10^6$ seconds (about two CPU months).

To consider how the algorithms compare on smaller problems, we test the hypotheses H_0: $\beta_0 = 1$ against H_a: $\beta_0 > 1$. The p-value for this second test is less than 0.0001, indicating

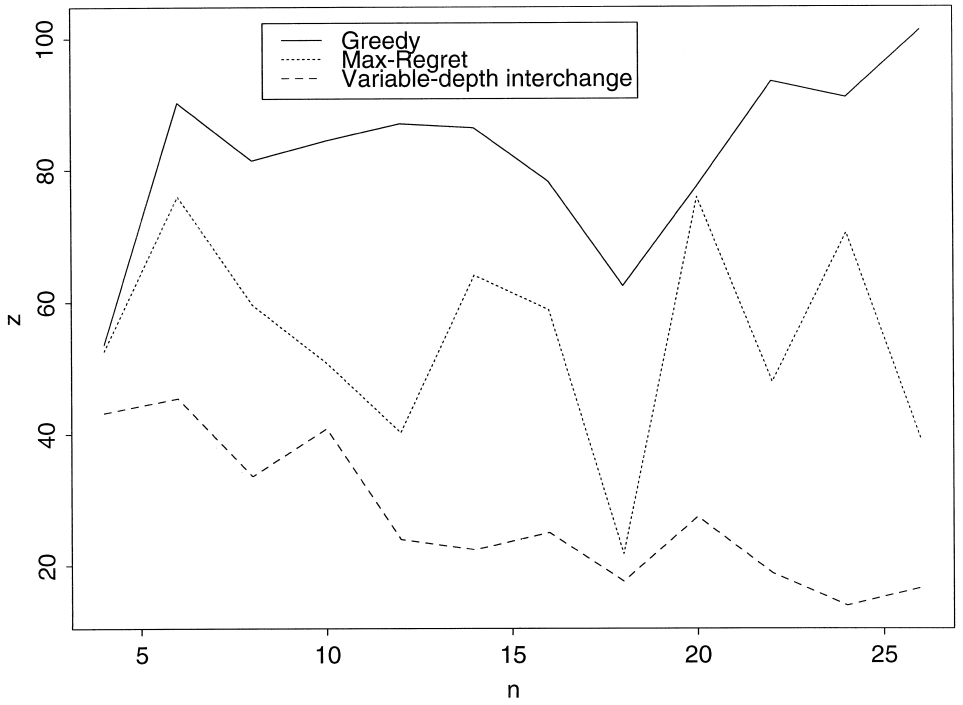

Figure 7. Means of Balas and Saltzman data.

that for small to medium problems, AUGMENTED does take proportionally longer. It should be noted that by performing two hypothesis tests, we have slightly inflated the experiment-wise error rate.

Case Study IV: Analysis of Variance

To illustrate the comparison of heuristics on a set of randomly generated test problems, the data of Balas and Saltzman[7] will be used. Although Balas and Saltzman report only mean results for each treatment group, the original data are used in the analysis presented here. Three primal heuristics (GREEDY, MAX-REGRET, and VARIABLE-DEPTH INTERCHANGE, a local-search procedure applied to the MAX-REGRET solution) are compared to the optimal solution for the three-index assignment problem. The size of the problems varied over $n = 4, 6, \ldots, 26$. (The integer programming formulation requires n^3 variables and $3n$ constraints.) For each size, five problems were generated at random with costs in the range 0–100. Fig. 7 shows the mean solution values for all heuristics as a function of problem size. From largest to smallest they are: GREEDY, MAX-REGRET, and VARIABLE-DEPTH INTERCHANGE. Since the optimal solution is known (and positive), the metric chosen is $(z - z^*)/z^*$ (where z is the heuristic solution value and z^* is the optimal value), which expresses the result as a proportion of the optimal solution. (This performance metric does not meet the criteria of Zemel,[68] but we will not address this issue here.) Fig. 8 shows the mean value of this metric for each heuristic, again as a function of problem size n. It can be seen that for all heuristics, the metric increases nonlinearly with n. The residual plot (not shown) shows that residual magnitude also increases with n, which indicates that a variance-stabilizing transformation will likely simplify the model and increase the power of any hypothesis tests. Because of the trend shown in Fig. 8, because of the increasing magnitude of the residuals, and because the metric is nonnegative, a log transformation is indicated.

Fig. 9 shows the means of the log-transformed metric values. As with the previous figures, it appears that VARIABLE-DEPTH INTERCHANGE outperforms MAX-REGRET, which in turn outperforms GREEDY, for all n. All three heuristics' metrics seem to be different and seem to depend upon n.

In order to test these observations, we fit the model

$$y_{nij} = \mu + \alpha_n + \beta_i + \gamma_{ni} + \epsilon_{nij},$$

where the α_n are the sample size effects; the β_i are the heuristic effects; the γ_{ni} are the interaction effects; $y_{nij} = \ln((z - z^*)/z^*)$ for a particular observation; $n = 4, 6, \ldots, 26$; $i = 1$ for GREEDY, 2 for MAX-REGRET, and 3 for VARIABLE-DEPTH INTERCHANGE; and $j = 1, 2, 3, 4, 5$ indicates the randomly generated problem. This is the classic analysis of variance model for two factors (n and heuristic) with interaction. The resulting hypothesis tests show that both n and the heuristic type are significant ($p < 0.0001$) and that the interaction is insignificant ($p = 0.7225$). Single-degree-of-freedom contrasts are used to compare GREEDY to other heuristics and to compare MAX-REGRET to VARIABLE-DEPTH INTERCHANGE. These contrasts are significant ($p < 0.0001$ in both cases), showing that VARIABLE-DEPTH INTERCHANGE provides a significant (in the statistical sense) improvement over MAX-REGRET.

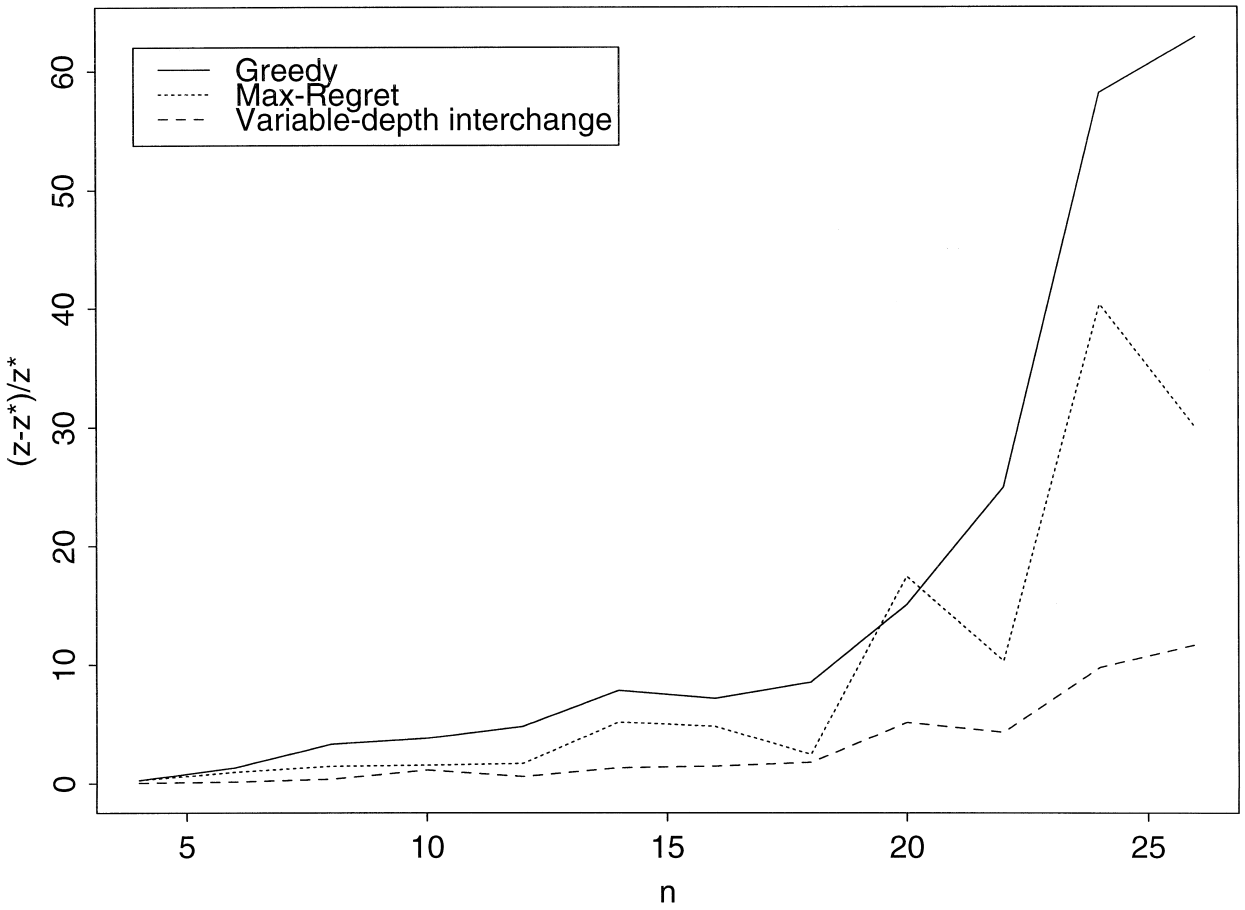

Figure 8. Means of $(z - z^*)/z^*$.

One might infer from Fig. 9 that y is linearly increasing in n for all three heuristics, although the amount of noise makes this unclear. In order to test this, we fit the model

$$y_{nij} = \beta_{0i} + \beta_{1i} n + \epsilon_{nij},$$

where the β_{0i} are intercepts, the β_{1i} are slopes, and ϵ_{nij} is the random variation. This model allows for three lines with different slopes and intercepts. The resulting hypothesis tests show that the three heuristics do not have significantly different slopes (i.e., H_0: $\beta_{11} = \beta_{12} = \beta_{13} = 0$ is not rejected; $p = 0.60$), but they do have different intercepts (i.e., H_0: $\beta_{01} = \beta_{02} = \beta_{03} = 0$ is rejected with $p < 0.0001$). However, for H_0: $\beta_{02} = \beta_{03}$, $p = 0.1568$, so this model does not provide much evidence that VARIABLE-DEPTH INTERCHANGE outperforms MAX-REGRET. Again, this result is likely due to the large amount of noise in the linear trend.

Case Study V: Multiple Comparison Procedures

Ernst and Kirshnamoorthy[18] develop lower-bounding techniques using shortest paths to solve large hub location problems. The hub location problem involves selection of p nodes in a complete graph on n nodes to act as hubs. All traffic between nodes is routed from the origin through one or two hubs to the destination. Inter-hub shipment costs are lower than hub-to-client costs. In the single-allocation problem, each non-hub node is allocated to a single hub; in the multiple-allocation problem, non-hub nodes may be allocated to multiple hubs. Each arc may have unit costs associated with hub-to-hub and hub-to-non-hub shipments. The objective is to minimize total shipment costs.

The new technique (labeled EK) involves partitioning the nodes into clusters and distributing the hubs to be allocated among the clusters. The method's performance depends on the choice of r, the initial number of clusters. To explore this dependency, the authors chose two multiple-allocation problems from a dataset called "AP" consisting of mail flows in an Australian city[17]—one with $n = 50$ locations and $p = 2$ hubs and the other with $n = 50$ and $p = 4$. The problems were solved repeatedly for different values of r. For each problem, CPU time was minimized when $r = 12$, which is approximately $50/4$. Thus, the authors concluded that $r = n/4$ would be a good choice in all cases. To confirm this, the authors compare CPU times for MILP, EK using $r = p$, and EK using $r = n/4$. The techniques were compared on the AP problems. The authors report:

> For all the single-allocation problems, the results with $r = n/4$ are consistently better, in terms of running time

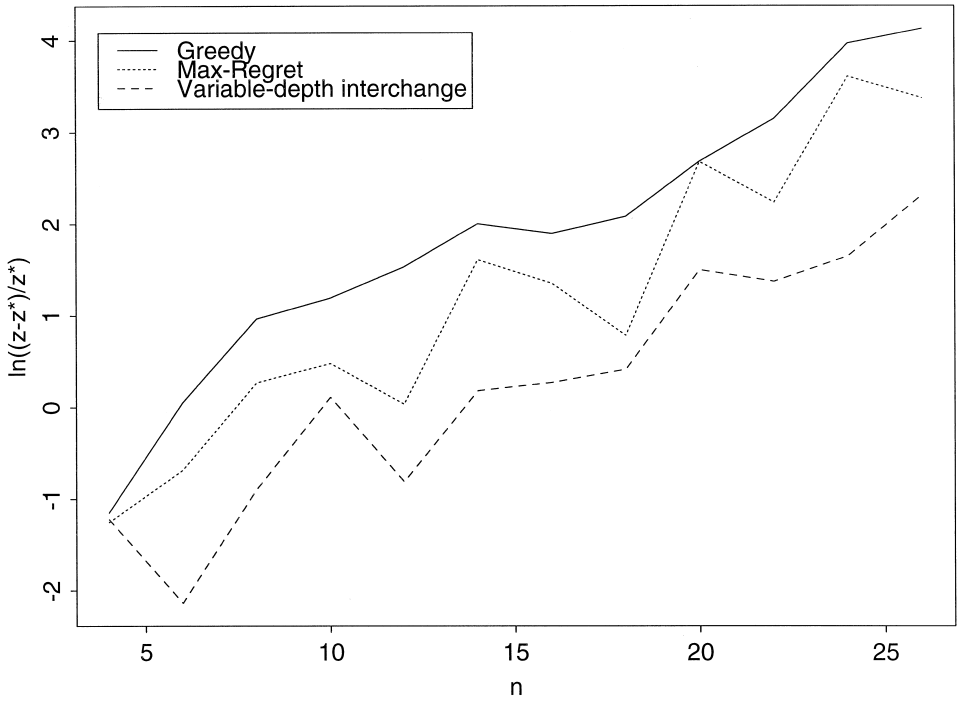

Figure 9. Means of $\ln((z - z^*)/z^*)$.

and the number of tree nodes, than those with $r = p$. However, the differences become significant only for the larger problems.

Because the number of nodes is not a meaningful comparison between MILP and the non-MILP methods, we use CPU time as the response variable. To check for heteroskedasticity, the mean and standard deviation of CPU times were calculated for each solution method for each level of n. Fig. 10 clearly shows that as the mean CPU time increases, the standard deviation increases as well. Because of the small sample size and marked heteroskedasticity, a variance-stabilizing transformation is indicated. Several transformations were considered: $\sqrt{\text{CPU}}$, $\sqrt[4]{\text{CPU}}$, and ln CPU. Of these, ln CPU showed no relation between the mean and the standard deviation, so it was chosen as a good variance-stabilizing transformation. As a check on the transformation, a normal probability plot was made of the residuals from the model (3) below. The linearity of this plot also supports the chosen transformation. A Box-Cox procedure[10] for choosing a transformation could also be used.

The data were analyzed as an unreplicated factorial experiment with three factors: $n \in \{10, 20, 25, 40, 50\}$, $p \in \{2, 3, 4, 5\}$, and solution method (MILP, EK with $r = p$, or EK with $r = n/4$). This provides 60 observations, of which one is censored. The authors terminated the program at 1500 CPU seconds. In the following analysis, the single censored observation is omitted. If a significant portion of the data was censored, a censored-data analysis could be performed.

An analysis of variance was performed on the transformed data using the model with all main effects and two-factor interactions. Because the experiment is unreplicated, the three-factor interaction is used as the error estimate. The model is

$$y_{inp} = \mu + \alpha_i + \beta_n + \gamma_p + (\alpha\beta)_{in} + (\alpha\gamma)_{ip} + (\beta\gamma)_{np} + \epsilon_{inp}, \tag{3}$$

where y_{inp} = ln CPU time for the ith solution method with a specified choice of n and p, μ is the overall mean, α_i is the effect of the ith solution method, β_n is the effect of the level of n, γ_p is the effect of the level of p, the multiplicative terms represent interactions, and ϵ_{inp} is the random error term.

All effects show significant differences: the $n \times p$ and $n \times$ (solution method) interactions have p-values of 0.0135 and 0.02568, respectively; all other p-values are smaller than 10^{-4}. This indicates that the solution method does have an effect.

Two single-degree-of-freedom contrasts were formed—one compares MILP to EK, while the other compares the two choices of r. The contrasts show that MILP has longer solution times, while the choices of r do not show significance. A closer look at the data shows that part of the reason the choice of r does not appear to have an effect is that for the first five problems n is so small that choosing $r = p$ is equivalent to choosing $r = n/4$. Thus, these problems are essentially replicate observations, which we would expect to show no difference except possibly measurement error. Including these observations in a comparison of the two choices of r may bias the results. However, even when the first five observations are removed, a contrast comparing the

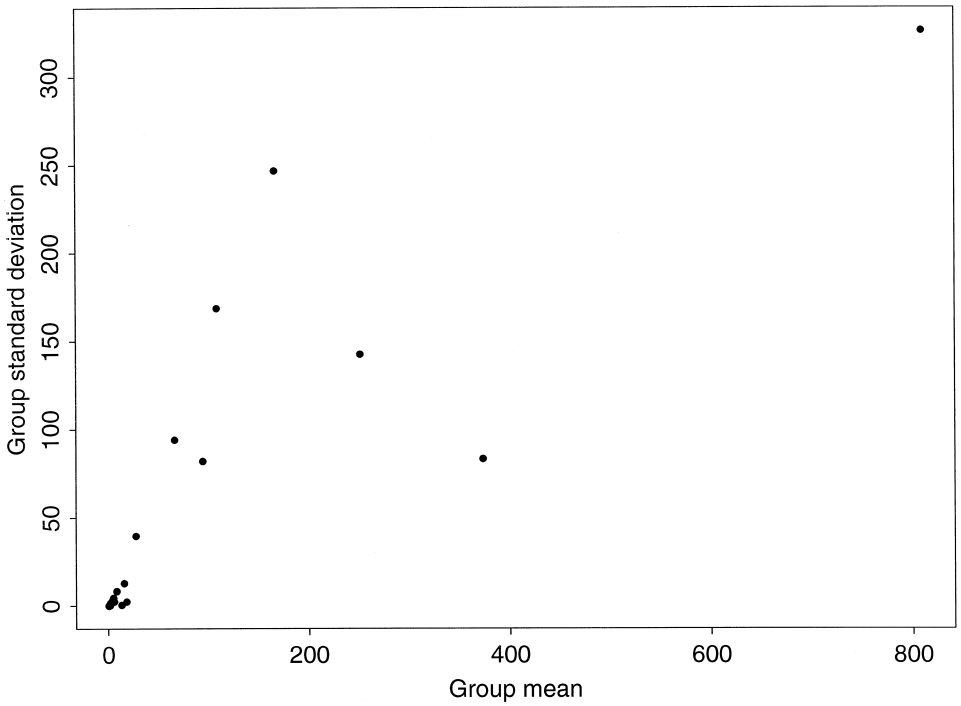

Figure 10. Relationship between sample mean and standard deviation.

choices of r on the remaining observations does not show a significant difference in the transformed data.

If a significant difference *had* been found, it would be desirable to make a distribution-free confidence interval (sometimes called a *Tukey interval*)[34] for the median difference in solution times between the two choices of r. A normal-based confidence interval is not appropriate because the untransformed data exhibit heteroskedasticity and nonnormality and the sample sizes are small. The Tukey interval is obtained by inverting the Wilcoxon signed rank test statistic; that is, the interval consists of all the values of M for which the Wilcoxon test statistic would not lead to rejecting the null hypothesis. In this case, the distribution-free confidence interval is (0, 0.5), indicating that the choice of r makes little difference, at least for the size of problem considered here.

It should be noted that the two-factor interactions are all significant at the 0.05 level, although only the interaction between solution method and p is significant at the 0.01 level. This indicates that the solution methods compare differently at different levels of p, even when n is held constant. Fig. 11 illustrates the nature of the interaction. This may be an indication that choosing r only as a function of n is not the best choice.

A better experimental design might have avoided this difficulty. Recall that the original experiment consisted of two problems: one with $n = 50$, $p = 2$ and the other with $n = 50$, $p = 4$. The authors' choice of $r = n/4$ was based on the observation that, for these two problems, solution time was minimized with $r \approx 12$. If the authors wished to know how to choose r optimally for problems with different values of n and p (as appears to be the case), a more efficient way to answer the question would be to design an experiment with that in mind. A grid of problems with different values of n, p, and r could be solved, and the resulting solution times could be analyzed as the responses in a three-factor experiment. Such an analysis would provide a sound mathematical basis for choosing the functional form of r.

Case Study VI: Randomized Block Design

Rochat and Taillard[58] present a probabilistic technique to diversify, intensify, and parallelize a local search adapted for solving vehicle routing problems. In addition to wide applicability and parallel-processing capability, the authors claim that their technique improves the quality of first-level tabu search results and generates solutions that are often better than the best published solutions, and that a further improvement can often be gained via a post-optimization procedure.

The authors demonstrate their technique on a set of 56 vehicle routing problems with time windows. For each problem, the number of vehicles and total length of all tours are reported for the best published solution, the best solution found with the authors' tabu search technique, and the best solution found by the authors. Each solution is listed as a pair—number of vehicles and distance traveled. A solution with fewer vehicles is superior to a solution using more vehicles, even if the latter reduces the total distance traveled. Under these conditions, it is impossible to compare the solutions on a nominal scale. Instead, the three solutions for each problem are ranked from best to worst, and the analysis is based on the ranks of the solutions.

Since all three sets of solutions were obtained from the

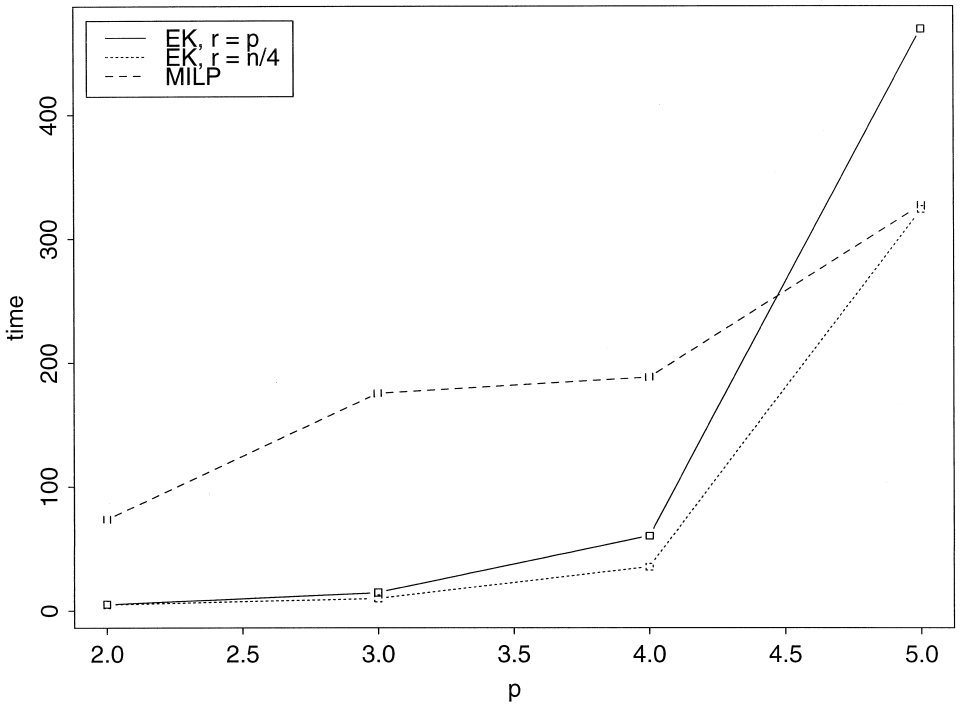

Figure 11. Interaction between solution method and p for $n = 40$.

same set of 56 problems, the experimental design is a randomized block design with one observation per cell. Friedman's test[34] is used to analyze the data; this is very similar to an analysis of variance F-test for a randomized block design, except that the analysis is performed on the ranks rather than on the solutions themselves.

The null hypothesis under consideration is that all three solution sets have the same median, and the general alternative hypothesis is that at least two solution sets have different medians. To perform the test, the three solutions are ranked from 1 (best) to 3 (worst) for each problem instance. In the case of ties, each tied solution is given rank equal to the average of the tied ranks. Then Friedman's test with a correction for ties is performed. For large numbers of blocks, the test statistic has an approximately chi-squared distribution. The p-value calculated from this approximation is 1.0×10^{-8} (calculated using S-Plus[15]), which indicates very strong evidence against the null hypothesis.

Since our hypothesis test shows significant differences among the three groups, we proceed to make pairwise comparisons. First, we compare the best solution found to the best solution previously published, and then we compare the best solution found to the best solution found with tabu search. The average ranks for the three groups are: 2.03 for best published solution, 2.54 for best tabu search, and 1.43 for best solution found. (The median ranks for the three groups are 2, 2.75, and 1, respectively.) Formally, let M be the median difference between the best solution found and the best solution previously published. The comparison of interest is

$$H_0{:}M = 0 \text{ vs. } H_a{:}M < 0.$$

This comparison, using the Wilcoxon signed-rank test statistic, has a p-value of 0.0004, indicating that the authors' best solution found definitely shows a general improvement over the best published solutions.

As the mean and median ranks above indicate, it is *not* the case that the authors' tabu search results are generally better than the best published solution, although there are some instances where tabu search shows an improvement.

Rochat and Taillard indicate that most but not all of their best solutions were found using the post-optimization technique. Thus, it seems that the post-optimization technique shows great promise in local searches and other vehicle routing problems.

Case Study VII: Mixed Linear Model

Gu et al.[25] experiment with a number of issues regarding the implementation of lifted cover inequalities in a branch-and-cut algorithm. The original test set was a set of 15 MIPLIB[9] problems: of the 29 available pure 0–1 programs, 15 were chosen as being suitable to the algorithm, nontrivial, and capable of solution in a reasonable amount of time.

In these 15 problems, the number of variables ranges from 27 to 2,756, the number of rows (constraints) from 6 to 252, the number of knapsack constraints from 2 to 382, and the number of knapsack constraints with nonoverlapping generalized upper bounds (GUB) from 0 to 352.

On this set of problems, a number of different algorithmic choices were considered—whether to use approximation or exact algorithms to compute the minimal cover and the lifting coefficients, whether to partition the cover, and what lifting order to use. This provides four different treatment groups on which to compare the lifted cover inequalities

Table I. Summary of MANOVA results*

Response	Affected by Linearization (ordered worst to best)†	Estimated Differences ±95% Margin of Error‡
Lower constraints		
Upper constraints	EGW—DF—CGW§	252 ± 75, 57 ± 25
lpvalue		
lpcpu	EGW—(CGW–DF)	0.59 ± 0.21
pivots	DF—(EGW–CGW)§	6781 ± 2185
totalcpu	DF—EGW—CGW§	40.13, 14.06
nodes	DF—(EGW–CGW)§	531 ± 249

*CGW, Condensed Glover-Woolsey linearization; EGW, Expanded Glover-Woolsey linearization; DF, Dantzig-Fortet linearization.
†Linearizations grouped together in parentheses were not found to be significantly different. Blank lines indicate that no significant differences were found.
‡Margins of error are provided when the assumptions of the confidence interval analysis are not violated.
§Difference was only significant for Balas and Mazzola's problems.

(LCIs) and another four groups on which to compare lifted GUB cover inequalities (LGCIs). Based on their results, the authors make a number of recommendations on which implementation choices are appropriate under some circumstances and close with the following:

> For all other algorithmic options our computational results are inconclusive, i.e., they do not single out a choice that clearly dominates the others.
>
> Our experiments have shown that the algorithmic choices that have to be made when implementing a branch-and-cut algorithm with cover inequalities may or may not have a substantial impact on its performance, and therefore, it may be important to experiment with various possible options.

We analyzed the first dataset (LCIs) as a mixed linear model, where the algorithm choices were treated as fixed levels of treatment, and the numbers of variables, constraints, knapsack constraints, and knapsack constraints with nonoverlapping GUBs were treated as regressor variables. The model is

$$y_{ij} = \mu + \beta_{0i} + \beta_{1i}x_{1j} + \beta_{2i}x_{2j} + \beta_{3i}x_{3j} + \beta_{4i}x_{4j} + \epsilon_{ij},$$

where y_{ij} is the change in solution time for the jth problem solved with the ith algorithm choice, β_{0i} is the effect of the ith algorithm choice, and $x_{1j}, \ldots, x_{4j}$ are the numbers of variables, constraints, knapsack constraints, and knapsack constraints with nonoverlapping GUBs on the jth problem, respectively. In our analysis, the algorithm choice cannot be shown to affect CPU time, either through the intercept or through any covariate. While it is true that the set of problems is small, the analysis provides 46 degrees of freedom for error, and several of the covariates were shown to have significant effects. Thus, the data provide no evidence that algorithm choice has any effect on CPU time. A further analysis of algorithm choice might focus on a more systematic examination of different problem classes rather than on the relatively unstructured problems in MIPLIB.

Case Study VIII: Multiple Analysis of Variance

Hansen et al.[26] present a survey of nonlinear programs in 0–1 variables with nonlinear constraints. As a part of this survey, they conduct extensive computational tests of linearization, algebraic, enumerative, and cutting-plane methods. They compare three linearization methods—Dantzig-Fortet, Condensed Glover-Woolsey, and Expanded Glover-Woolsey—using a selection of Taha's and Balas and Mazzola's problems (see Hansen et al.[26] for details of these methods and problems). The authors report the total CPU time used, the number of pivots, the number of lower constraints (M1), the number of upper constraints (M2), the LP value, the CPU time used for solving the first LP relaxation, and the number of nodes in the branch-and-bound search tree. Because there is more than one response variable, we use a multiple analysis of variance (MANOVA) to analyze the data. Here, the response variables are TOTALCPU, PIVOTS, LPVALUE, LPCPU, M1, M2, and NODES; the explanatory variables are LINEARIZATION, and PTYPE (see below); and the problem instance is treated as a blocking variable. No units are given for the CPU times. We have assumed, as seems likely, that the times were reported in seconds.

An exploratory analysis of the data shows nonhomogeneity of errors as well as evidence that, as the original investigators suspected, the choice of linearization interacts with the problem type (Taha's versus Balas and Mazzola's). Before proceeding further, the CPU times (TOTALCPU and LPCPU) are given a log transformation, and a new categorical variable PTYPE, to indicate Taha's or Balas and Mazzola's, is introduced.

The results of the MANOVA analysis, carried out in SAS[61] using PROC GLM, are summarized in Table I.

It should be noted that, with regard to most responses, all three linearizations were indistinguishable on Taha's problems. These problems displayed small values and little variability on nearly all responses. With today's computers, one would expect this problem to be even greater. If another

such survey were to be conducted, it would seem advisable to discard such small problems in favor of more interesting ones.

In motivating their survey, Hansen et al. observe that "Stecke discusses various linearizations but considers *a priori* that the most compact linearizations are the best." In summarizing the computational results, they conclude that "compact linearizations do not appear to be clearly better than expanded ones giving rise to tighter linearizations in constrained nonlinear 0–1 programming." This statement is only partially supported by the statistical analysis. When comparing the Condensed and Expanded Glover-Woolsey linearizations, one observes the following:

- The condensed and expanded linearizations are statistically indistinguishable in terms of lower constraints, lpvalue, pivots, and nodes.
- The condensed linearization is statistically superior in terms of upper constraints, lpcpu, and totalcpu.
- There was no response on which the expanded linearization showed a statistically demonstrable superiority.

Thus, while it is true that the compact linearization was not shown to be better in every regard, the data provide very little evidence that the expanded linearization is better in any regard. A better case might be made for the expanded linearization if one chose problems for which the tighter relaxation of the expanded linearization were more pronounced or replaced Taha's problems (on which nothing much could be shown) with problems of greater size and difficulty.

5. Conclusion

Published comparisons of algorithms and heuristics often contain a great many numbers, some frequently misapplied summary statistics, and very little data analysis. Comparisons of algorithms or heuristics begin with the consideration of what criteria should be measured and what experimental design will best elicit information from the data.

Exploratory data analysis is an invaluable tool that is underutilized in the OR literature. Simple graphs and summary statistics, while not an end in themselves, often suggest which models, tests, and intervals will most clearly answer the experimenter's questions. The kinds of data generated in computational tests (e.g., running times) are often nonnormal and can be analyzed more easily after transformation. We found log transformations to be particularly useful for variance stabilization, outlier detection, and model formulation.

Formal statistical analysis, while it must be used with care, can lead to surprising conclusions. A parametric model that fits well provides a concise description of the underlying process. In some cases, the investigator will be able to make much stronger statements on the basis of a statistical analysis than can be made from the examination of individual instances. In other cases, the investigator's intuition will not be supported by the data. This should be considered a negative conclusion only in the sense of formal logic—in actual practice, such results are invaluable for eliminating fruitless areas of investigation and suggesting more promising directions of inquiry.

It is our hope that the guidelines and case studies that we have provided will encourage better formal statistical analysis of results.

Acknowledgments

This research was supported in part by ONR Grant N00014-97-1-0784. We acknowledge the valuable and detailed comments provided by the referees on earlier versions of the manuscript.

References

1. R.K. AHUJA, T.L. MAGNANTI, and J. B. ORLIN, 1993. *Network Flows*, Prentice Hall, Englewood Cliffs, New Jersey.
2. R.K. AHUJA and J.B. ORLIN, 1996. Use of Representative Operation Counts in Computational Testing of Algorithms, *INFORMS Journal on Computing 8:3*, 318–330.
3. M.G. AKRITAS, S.A. MURPHY, and M.P. LAVALLE, 1995. The Thiel-Sen Estimator with Doubly Censored Data and Applications to Astronomy, *Journal of the American Statistical Society 90:429*, 170–177.
4. M.M. AMINI and R.S. BARR, 1993. Network Reoptimization Algorithms: A Statistically Designed Comparison, *INFORMS Journal on Computing 5:4*, 395–409.
5. M.M. AMINI and M. RACER, 1994. A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem, *Management Science 40:7*, 868–890.
6. R.J. ANDERSON, 1996. The Role of Experiment in the Theory of Algorithms, in *Fifth DIMACS Challenge Workshop: Experimental Methodology Day*, www.cs.amherst.edu/~dsj/methday.html.
7. E. BALAS and M.J. SALTZMAN, 1991. An Algorithm for the Three-Index Assignment Problem, *Operations Research 39:1*, 150–161.
8. R.S. BARR, B. L. GOLDEN, J.P. KELLY, M.G.C. RESENDE, and W.R. STEWART, JR., 1995. Designing and Reporting on Computational Experiments with Heuristic Methods, *Journal of Heuristics 1*, 1–32.
9. R.E. BIXBY, E.A. BOYD, and R.R. INDOVINA, 1992. MIPLIB: A Test of Real-World Mixed-Integer Programming Problems, *SIAM News 25*, 16.
10. G.E.P. BOX and D.R. COX, 1964. An Analysis of Transformations, *Journal of the Royal Statistical Society 26*, 211–243.
11. G.E.P. BOX, W.G. HUNTER, and J.S. HUNTER, 1978. *Statistics for Experimenters*, Wiley, New York, 205–207.
12. M. COFFIN and M.J. SALTZMAN, 1999. Robust Regression with Doubly Censored Data, Technical Report 671, Department of Mathematical Sciences, Clemson University, Clemson, SC.
13. H.P. CROWDER, R.S. DEMBO, and J.M. MULVEY, 1978. Reporting Computational Results in Mathematical Programming, *Mathematical Programming 15*, 316–329.
14. H.P. CROWDER, R.S. DEMBO, and J.M. MULVEY, 1979. On Reporting Computational Experiments with Mathematical Software, *ACM Transactions on Mathematical Software 5:2*, 193–203.
15. DATA ANALYSIS PRODUCTS DIVISION, MATHSOFT, 1998. *S-Plus 5 for UNIX Guide to Statistics*, Seattle, WA.
16. R.S. DEMBO and J.M. MULVEY, 1976. On the Analysis and Comparison of Mathematical Programming Algorithms and Software, in *Computers and Mathematical Programming*, W.W. White (ed.), National Bureau of Standards, Washington, DC, 106–116.

17. A.T. ERNST and M. KRISHNAMOORTHY, 1996. Efficient Algorithms for the Uncapacitated Single Allocation *p*-hub Median Problem, *Location Science 4*, 139–154.
18. A.T. ERNST and M. KRISHNAMOORTHY, 1998. An Exact Solution Approach Based on Shortest-Paths for *p*-hub Median Problems, *INFORMS Journal on Computing 10:2*, 149–162.
19. R. FOURER and S. MEHROTRA, 1991. Performance of an Augmented System Approach for Solving Least-Squares Problems in an Interior-Point Method for Linear Programming, *COAL Newsletter 19*, 26–31.
20. D.M. GAY, 1985. Electronic Mail Distribution of Linear Programming Test Problems, *COAL Newsletter 13*, 10–12.
21. J. GILSINN, K. HOFFMAN, R.H.F. JACKSON, E. LEYENDECKER, P. SAUNDERS, and D. SHIER, 1977. Methodology and Analysis for Comparing Discrete Linear L_1 Approximation Codes, *Communications in Statistics, Simulation, and Computations B6:4*, 399–413.
22. B.L. GOLDEN, A.A. ASSAD, E.A. WASIL, and E. BAKER, 1986. Experimentation in Optimization, *European Journal of Operational Research 27*, 1–16.
23. B.L. GOLDEN and W.R. STEWART, 1985. Empirical Analysis of Heuristics, in *The Traveling Salesman Problem*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.) Wiley, New York, 207–249.
24. H.J. GREENBERG, 1990. Computational Testing: Why, How and How Much, *ORSA Journal on Computing 2:1*, 94–97.
25. Z. GU, G.L. NEMHAUSER, and M.W.P. SAVELSBERGH, 1998. Lifted Cover Inequalities for 0-1 Integer Programs: Computation, *INFORMS Journal on Computing 10:4*, 427–437.
26. P. HANSEN, B. JAUMARD, and V. MATHON, 1993. Constrained Nonlinear 0-1 Programming, *INFORMS Journal on Computing 5:2*, 97–119.
27. T. P. HETTMANSPERGER, 1984. *Statistical Inference Based on Ranks*, Wiley, New York.
28. C.R. HICKS, 1982. *Fundamental Concepts in the Design of Experiments, 3rd ed.*, Holt, Reinhart & Winston, New York.
29. D.C. HOAGLIN, V.C. KLEMA, and S.C. PETERS, 1982. Exploratory Data Analysis in a Study of the Performance of Nonlinear Optimization Routines, *ACM Transactions on Mathematical Software 8:2*, 145–162.
30. C.A.R. HOARE, 1962. Quicksort, *Computer Journal 5:1*, 10–15.
31. W. HOCK and K. SCHITTKOWSKI, 1981. *Test Examples for Nonlinear Programming Codes*, number 187, in Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, New York.
32. A. HOFFMAN, M. MANNOS, D. SOLOKOWSKY, and N. WEIGMANN, 1953. Computational Experience in Solving Linear Programs, *SIAM Journal 1*, 1–33.
33. K.L. HOFFMAN and R.H.F. JACKSON, 1982. In Pursuit of a Methodology for Testing Mathematical Programming Software, in *Evaluating Mathematical Programming Techniques*, J.M. Mulvey (ed.), number 199, in Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, New York, 177–199.
34. M. HOLLANDER and D.A. WOLFE, 1999. *Nonparametric Statistical Methods 2nd ed.*, Wiley, New York, 56–59.
35. J.N. HOOKER, 1995. Testing Heuristics: We Have It All Wrong, *Journal of Heuristics 1*, 33–42.
36. R.H.F. JACKSON, 1985. On the State of the Art of Computational Testing of Mathematical Programming Algorithms, *COAL Newsletter 12*, 8–13.
37. R.H.F. JACKSON, P.T. BOGGS, S.G. NASH, and S. POWELL, 1989. Report of the Ad Hoc Committee to Revise the Guidelines for Reporting Computational Experiments in Mathematical Programming, *COAL Newsletter 18*, 3–14.
38. R.H.F. JACKSON, P.T. BOGGS, S.G. NASH, and S. POWELL, 1991. Guidelines for Reporting Results of Computational Experiments: Report of the Ad Hoc Committee, *Mathematical Programming 49*, 413–425.
39. R.H.F. JACKSON and J.M. MULVEY, 1978. A Critical Review of Comparisons of Mathematical Programming Algorithms and Software (1953–1977), *Journal of Research of the National Bureau of Standards 83:6*, 563–584.
40. D.S. JOHNSON, 1996. A Theoretician's Guide to the Experimental Analysis of Algorithms, in *Fifth DIMACS Challenge Workshop: Experimental Methodology Day*, www.cs.amherst.edu/~dsj/methday.html.
41. N. KARMARKAR, 1984. Special presentation, *TIMS/ORSA National Meeting*, Dallas.
42. C.H. LAYMAN and R.P. O'NEILL, 1976. A Study of the Effect of LP Parameters on Algorithm Performance, in *Computers and Mathematical Programming*, W.W. White (ed.), National Bureau of Standards, Washington, DC, 251–260.
43. P. L'ECUYER, 1996. Simulation of Algorithms for Performance Analysis, *INFORMS Journal on Computing 8:1*, 16–20.
44. F.A. LOOTSMA, 1982. Performance Evaluation of Nonlinear Optimization Methods via Multi-criteria Decision Analysis and via Linear Model Analysis, in *Nonlinear Optimization 1981*, M.J.D. Powell (ed.), Academic, New York, 419–453.
45. I.J. LUSTIG, R.E. MARSTEN, and D.F. SHANNO, 1991. Computational Experience with a Primal-Dual Interior Point Method for Linear Programming, *Linear Algebra and its Applications 152*, 191–222.
46. C. MCGEOCH, 1992. Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups, *Computing Surveys 24:2*, 195–212.
47. C. MCGEOCH, 1996. Challenges in Algorithm Simulation, *INFORMS Journal on Computing 8:1*, 27–28.
48. C. MCGEOCH, 1996. Toward an Experimental Method for Algorithm Simulation, *INFORMS Journal on Computing 8:1*, 1–15.
49. A. MIELE and S. GONZALEZ, 1978. On the Comparative Evaluation of Algorithms for Mathematical Programming Problems, in *Nonlinear Programming, Vol. 3*, O.L. Mangasarian, R.R. Meyer, and S.M. Robinson (eds.), Academic, New York, 337–359.
50. D.C. MONTGOMERY, 1991. *Design and Analysis of Experiments*, Wiley, New York.
51. D.S. MOORE and G.P. MCCABE, 1999. *Introduction to the Practice of Statistics*, W.H. Freeman, New York.
52. L. NAZARETH and F. SCHLICK, 1976. The Evaluation of Unconstrained Optimization Routines, in *Computers and Mathematical Programming*, W.W. White (ed.), National Bureau of Standards, Washington, DC, 117–131.
53. P.J. NELSON, 1998. Application of the Analysis of Means, *Proceedings of the SAS Users Group International Conference. 13*, 225–230.
54. W. NELSON, 1982. *Applied Life Data Analysis*, Wiley, New York.
55. J. NETER, M.H. KUTNER, C.J. NACHTSHIEM, and J.W. WASSERMAN, 1998. *Applied Linear Statistical Models, 4th ed.*, Irwin, Homewood, IL.
56. J.B. ORLIN, 1996. On Experimental Methods for Algorithm Simulation, *INFORMS Journal on Computing 8:1*, 21–23.
57. H.D. RATLIFF and W. PIERSKALLA, 1981. Reporting Computational Experience, *Operations Research, 29:2*, xi–xiv.
58. Y. ROCHAT and E.D. TAILLARD, 1995. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing, *Journal of Heuristics 1:1*, 147–165.
59. P.J. ROUSSEEUW and A.M. LEROY, 1997. *Robust Regression and Outlier Detection*, Wiley, New York.

60. T. SAATY, 1980. *The Analytic Hierarchy Process*, McGraw-Hill, New York.
61. SAS INSTITUTE, 1989. *SAS/STAT User's Guide, Version 6, 4th ed.*, SAS Institute, Cary, NC.
62. R. SEDGEWICK, 1977. The Analysis of Quicksort Programs, *Acta Informatica 7*, 327–355.
63. E. SHAYAN, 1986. A Methodology for Algorithm Comparison in Mathematical Programming, *COAL Newsletter 15*, 3–11.
64. D.R. SHIER, 1996. On Algorithm Analysis, *INFORMS Journal on Computing 8:1*, 24–26.
65. J.W. TUKEY, 1977. *Exploratory Data Analysis*, Addison-Wesley, Reading, MA.
66. R.J. VANDERBEI, 1996. *Linear Programming: Foundations and Extensions*, Kluwer, Boston, MA.
67. W.W. WHITE, Ed., 1976. *Computers and Mathematical Programming*, National Bureau of Standards, Washington, DC.
68. E. ZEMEL, 1981. Measuring the Quality of Approximate Solutions to Zero-One Programming Problems, *Mathematics of Operations Research 6:3*, 319–332.