Journal of Heuristics, 1: 9–32 (1995)
© 1995 Kluwer Academic Publishers

Designing and Reporting on Computational Experiments with Heuristic Methods

RICHARD S. BARR
Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122, phone: (214) 768-2605, fax: (214) 768-3085, email: barr@seas.smu.edu

BRUCE L. GOLDEN
College of Business and Management, University of Maryland, College Park, MD 20742, email: bgolden@umdacc.umd.edu

JAMES P. KELLY
College of Business and Administration, University of Colorado at Boulder, Boulder, CO 80309, email: james.kelly@colorado.edu

MAURICIO G.C. RESENDE
Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974-2040, email: mgcr@research.att.com

WILLIAM R. STEWART, JR.
School of Business Administration, The College of William and Mary, Williamsburg, VA 23185, email: wrstew@mail.wm.edu

Abstract

This article discusses the design of computational experiments to test heuristic methods and provides reporting guidelines for such experimentation. The goal is to promote thoughtful, well-planned, and extensive testing of heuristics, full disclosure of experimental conditions, and integrity in and reproducibility of the reported results.

Key Words: heuristics, algorithms, experimental design, computational testing

While heuristic methods have always been a part of human problem solving, the mathematical versions are growing in their range of application as well as their variety of approach. New heuristic technologies are giving operations researchers, computer scientists, and practitioners the ability to routinely solve problems that were too large or complex for previous generations of algorithms.

The effectiveness of any proposed methodology for solving a given class of problems can be demonstrated by theoretical analysis and empirical testing. This article focuses on the issues involved in designing computational experiments to test heuristic methods and gives guidelines for reporting on the experimentation. When a new heuristic is presented in the computational and mathematical sciences literature, its contributions should be evaluated scientifically and reported in an objective manner, and yet this is not always done.

We follow in the footsteps of those pioneers who have championed high-quality reporting of computational experiments with mathematical programming software. These efforts began in the late 1970s with Crowder, Dembo, and Mulvey (1980), Gilsinn et al. (1977), Jackson and Mulvey (1978), with additional writing on the subject appearing more recently in Ahuja, Magnanti, and Orlin (1993), Barr and Hickman (1993), Golden et al. (1986), Greenberg (1990), and Jackson et al. (1990).

Reporting on experimentation with heuristic methods involves many of the same concerns as with optimization algorithms but has its own distinctive issues. While some elements of this topic have been explored elsewhere (Lin and Rardin, 1979; Golden and Stewart, 1985; Nance, Moose, and Foutz, 1987; Barr and Hickman, 1993; Golden et al., 1986; Reeves, 1993a), this article takes a comprehensive view of the issues involved and provides directions for researchers in this important and expanding area.

1. Heuristic Methods

A *heuristic method* (also called an *approximation algorithm*, an *inexact procedure*, or, simply, a *heuristic*) is a well-defined set of steps for quickly identifying a high-quality solution for a given problem, where a *solution* is a set of values for the problem unknowns and "quality" is defined by a stated evaluation metric or criterion. Solutions are usually assumed to be *feasible*, meeting all problem constraints. The purpose of heuristic methods is to identify problem solutions where time is more important than solution quality, or the knowledge of quality.

Some heuristic methods are associated with problems for which an optimal, correct, or exact solution exists and can be computed by an *optimization* or *exact* algorithm. Heuristic methods are often used to identify "good" approximate solutions to such problems in less time than is required for an exact algorithm to uncover an exact solution. *Embedded* heuristics are those used within exact algorithms to expedite the optimization process.

Heuristics can be straightforward or more complex. Straightforward algorithms tend to have well-defined termination rules, as with greedy and local-neighborhood-search methods, which stop at a local optimum. More complex algorithms may *not* have standard termination rules and typically search for improved solutions until an arbitrary stopping point is reached (see Figure 1). Most metaheuristics—such as tabu search, simulated annealing, genetic algorithms, neural nets, and GRASP—are examples of more complex algorithms. It is essential that the experimenter fully specify the steps and stopping rules of new methods, especially complex ones.

2. Computational Experiments with Heuristics

Since an algorithm is an abstraction, it is evaluated indirectly by experimenting with a specific implementation. An *experiment* is a set of tests run under controlled conditions for a specific purpose: to demonstrate a known truth, to check the validity of a hypothesis, or to examine the performance of something new. Investigators in all fields of study perform experiments to demonstrate theory, to uncover knowledge about a particular process, and to measure the effect of one or more factors on some phenomena. A *factor* is any controllable variable in an experiment that influences the outcome or result of an experiment.

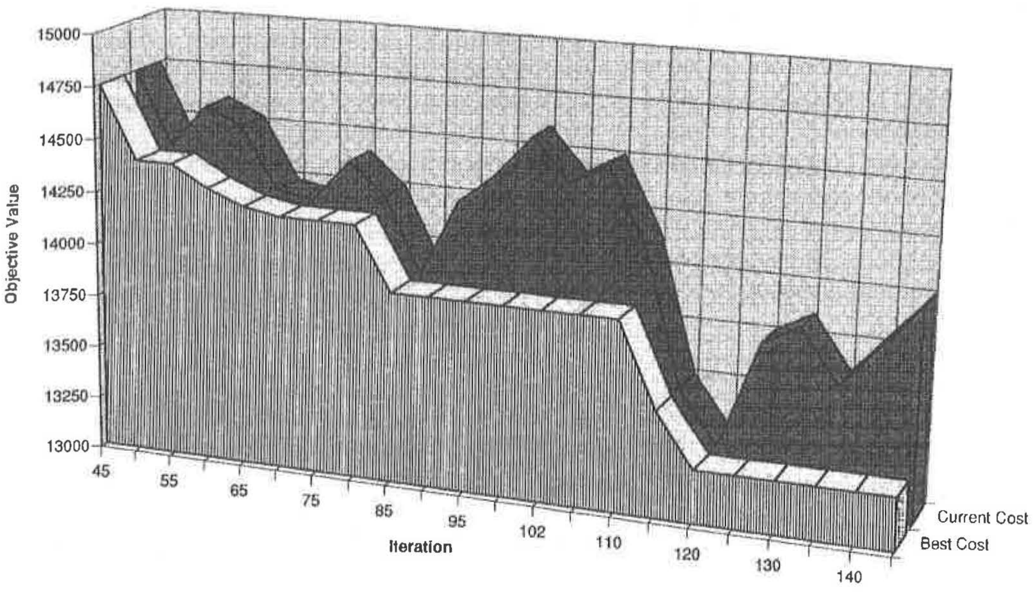

Figure 1. Current and best-found solution quality versus iteration number for a tabu search metaheuristic, as it visits five local optima (adapted from Knox, 1994).

In the computational testing of an algorithm, an experiment consists of solving a series of problem instances using a computer implementation. The experimenter has great latitude in selecting the problems, implementing the algorithm, choosing a computing environment, selecting performance measures, setting algorithm options, and reporting the results. The choice made for each factor can have a substantial effect on the results and significance of the experiment. Therefore, to ensure that the reported information is meaningful, the researcher should document and use an experimental design that considers as many factors as is possible and practicable and that effectively measures their impacts on the results.

Experimentation is a process whose steps can be viewed as in Table 1 (adapted from Montgomery, 1984). In the following sections we address the heuristic-testing issues associated with each step and recommend approaches to each.

Table 1. Experimentation steps.

1. Define the goals of the experiment.
2. Choose measures of performance and factors to explore.
3. Design and execute the experiment.
4. Analyze the data and draw conclusions.
5. Report the experiment's results.

3. Setting Experimentation Goals

A research experiment should have a purpose, stated clearly and defined prior to the actual testing. This is a statement of the questions to be answered and the reasons that experimentation is required. This purpose can guide and focus the efforts of the investigator and help identify the type of results to seek, the hypotheses to test, the tests to run, the factors to explore, the measures to use, and the data needed to support all of the above.

The purpose of the research should be related to the ultimate goal of heuristic methods: fast, high-quality solutions to important problems. Although no set standards exist for publishable algorithmic research, it is generally accepted that a heuristic method makes a contribution if it is

- Fast—producing high-quality solutions quicker than other approaches;
- Accurate—identifying higher-quality solutions than other approaches;
- Robust—less sensitive to differences in problem characteristics, data quality, and tuning parameters than other approaches (Hopfield and Tank, 1985);
- Simple—easy to implement (Dyer and Frieze, 1985; Lin and Kernighan, 1973);
- High-impact—solving a new or important problem faster and more accurately than other approaches (Rothfarb et al., 1970);
- Generalizeable—having application to a broad range of problems (Feo and Resende, 1995; Glover, 1989; Holland, 1975; Metropolis et al. 1953);
- Innovative—new and creative in its own right.

In addition, research reports about heuristics are valuable if they are:

- Revealing—offering insight into general heuristic design or the problem structure; establishing the reasons for its performance and explaining its behavior;
- Theoretical—providing theoretical insights, such as bounds on solution quality (Held and Karp, 1970, 1971; Hochbaum and Shmoys, 1985; Johnson and Papadimitriou, 1985).

Whatever the contribution of the heuristic being reported, some computational experimentation will be necessary to demonstrate that the procedure does what the author claims it will do.

Computational experiments with algorithms are usually undertaken (1) to compare the performance of different algorithms for the same class of problems or (2) to characterize or describe an algorithm's performance in isolation. While these goals are somewhat interrelated, the investigator should identify what, specifically, is to be accomplished by the testing (e.g., what questions are to be answered, what hypotheses are to be tested). Since comparative and descriptive experiments differ in their primary goals, their testing concerns differ as well.

3.1. Comparing Algorithms

The most prevalent computational experiment concerns the relative "effectiveness" (in terms of stated performance measures such as computational effort or quality of solution) of different heuristic methods in solving specific classes of problems. At this initial stage in the process, the investigator must choose or develop the algorithms and computer codes for study and the problem classes to address.

In selecting a problem class, effective solution methods for general classes are more highly prized than those for special cases or problems with a predetermined structure. The more specialized the problem structure, the greater the efficiencies that should result and the heavier the investigator's burden to demonstrate relevance and contribution.

Often the experimental goal is to compare a new approach to established techniques. In general, algorithms should be tested against their best competition. Also, well-known or published heuristics provide valuable points of reference (even if they are not the state of the art) and the new heuristic should be compared with them. Rather than making comparisons with published results on different problems and machines, it is preferable to obtain (or create, if necessary) software for the competing methods and make comparisons within the same computing environment.

If other methods do not exist, then a more general method, such as one based on linear or integer programming, or a simple greedy approach should serve as a baseline. Some heuristics based on probabilistic elements (such as GRASP, genetic algorithms, and probabilistic tabu search) may be compared with a simple random restart procedure, in the absence of other alternatives. The procedure should use the same neighborhood employed by the heuristic being tested to find local optima, starting from a randomly generated initial point. The proposed heuristic should perform "significantly better" than this simple-minded approach as one demonstration of its effectiveness.

3.2. Describing Algorithm Performance

Descriptive experiments are created to characterize a given algorithm, rather than compare it with others. The objective is to gain understanding of the behavior of the methodology, and the factors that influence that behavior.

One category of descriptive experimentation is the use of simulation to characterize algorithm performance (Hooker, 1995; McGeoch, 1995). With this approach, a mathematical model of the algorithm is built and a controlled experiment is carried out to determine the effect of one or more factors on the performance of the algorithm. To gain insight into the effect of specific factors on an algorithm, one tests two codes that are identical except for the singled-out factor within a well-designed experiment (such as a factorial design) using techniques to determine the effect of the factor on specific algorithm performance measures such as analysis of variance.

4. Choosing Performance Measures and Factors

Within a computational experiment there is a set of dependent variables—the performance measures or results—that are affected by a set of independent variables—the problem, algorithm, and test-environment factors. Since the goals of the experiment are achieved by analyzing observations of these factors and measures, they must be chosen with that end in mind.

4.1. Measuring Performance

Perhaps the most important decision one makes in an experimental study of heuristics is the definition, or characterization, of algorithm performance. The literature contains a variety of criteria for evaluating a heuristic method and the choice of performance measures is a critical one. Essentially, most researchers and practitioners wish to answer the following questions when testing a given heuristic on a specific problem:

1. What is the quality of the best solution found?
2. How long does it take to determine the best solution?
3. How quickly does the algorithm find good solutions?
4. How robust is the method?
5. How "far" is the best solution from those more easily found?
6. What is the tradeoff between feasibility and solution quality?

As these questions might suggest, performance measures have tended to cluster in three areas: solution quality, computational effort, and robustness. Measures from each category should be used in a well-rounded study so that a key dimension is not ignored. And, as detailed in the following sections, there are numerous metrics that can be used.

Note that the third and sixth questions involve tradeoffs between two of these dimensions. Perhaps the most popular and illuminating exhibits of heuristic performance is a graph of solution quality as a function of time, as illustrated in Figure 2. The shape of this graph clearly reflects the power of a heuristic and is useful to practitioners as well as researchers.

4.1.1. Quality of Solutions. When testing an algorithm that finds an optimal solution to a given problem, the important issues are speed and rate of convergence to the optimal solution. For heuristics, the additional consideration of how close the heuristic solution comes to optimality is generally the primary concern of the researcher. When possible,

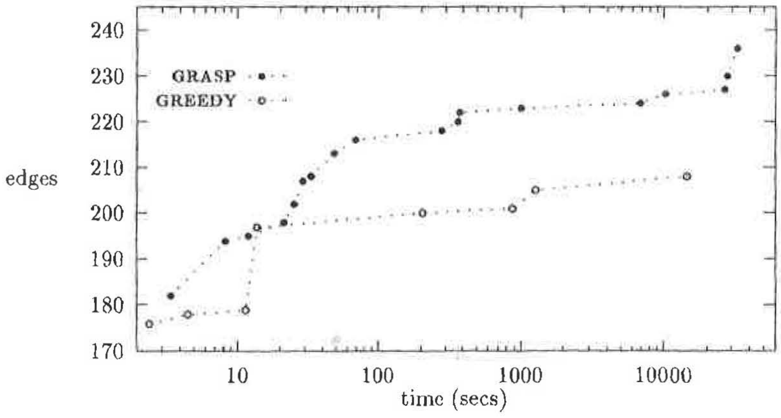

Figure 2. Quality-versus-time comparison of GRASP and pure GREEDY in search of planar subgraph of input graph with maximum number of edges (from Resende and Ribeiro, 1995).

the heuristic solutions obtained should be compared to the optimal solutions. Generally the percent deviation from optimality is reported.

When the optimal solution is unavailable, the percent deviation from a tight lower (upper) bound can be an effective measure of the quality of solution, as depicted in Figure 3. (For good examples of tight bounds, see Cornuejols, Sridharan, and Thizy (1991), Golden et al. (1986), Held and Karp (1970, 1971), Johnson (1990), Kelly, Golden, and Assad (1992), Martello and Toth (1990).) Of course, a gap of 20–30 percent between the bound and the heuristic solution is probably not tight enough to convey useful information (for example, minimal-spanning-tree value as a lower bound for the optimal traveling-salesman tour length).

For most standard problems, heuristic results exist in the open literature and direct comparison of a heuristic algorithm's performance to earlier heuristic solutions should be made in much the same way as comparisons are made to optimal solutions (see Aarts et al., 1994; Gendreau, Hertz, and Laporte, 1994).

4.1.2. Computational Effort. While heurisitcs that produce superior solutions are important, the speed of computation is a key factor. There are many portions of the process that should be timed, including:

- Time to best-found solution: This is the time required for the heuristic to find and report the solution the author is using in assessing the quality of the heuristic. This timing should include all processing involved (such as computation of distance matrices, and so on) along with all preprocessing.

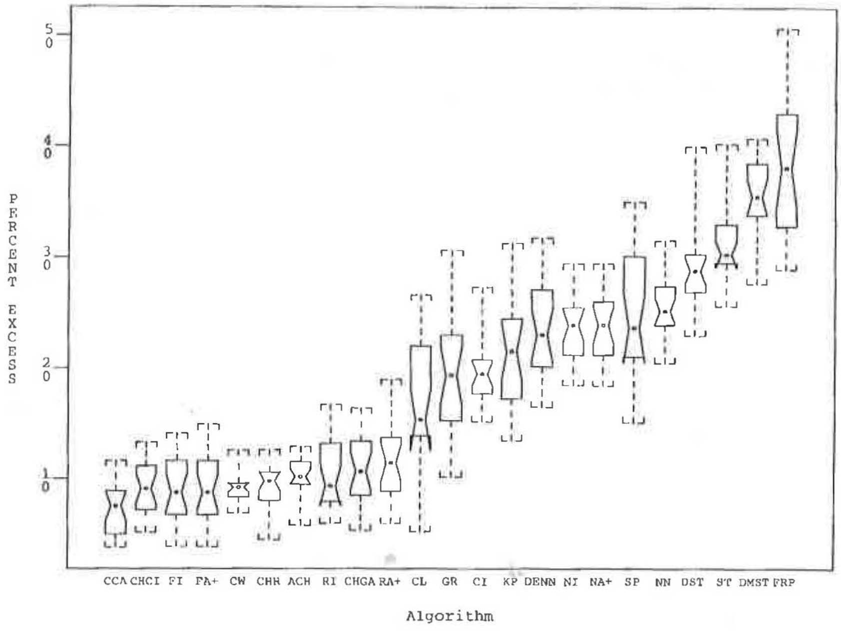

Figure 3. Boxplots for 23 heuristics showing percent excess over Held-Karp lower bound on 100-city traveling-salesman problems (from Johnson et al., 1995).

- Total run time: This is the algorithm's execution time prior to termination by its stopping rule. Since some complex heuristics have no standard termination criteria and can be allowed to run for an indefinite period of time, care must be given to reporting both the time required to produce the best-found solution and the total time of the run that produced it. Readers are naturally skeptical of results in which good solutions are identified shortly before the reported termination point.
- Time per phase: Where the heuristic is multi-phase, or composite (that is, initial solution, improved solution, final solution), the timing of each phase and the quality of solution at the end of each phase should also be reported. What is the "bang for buck" contribution of each phase?

All reported times should be for either a single set of algorithm parameter values or a specific rule that establishes the parameter values for each instance. When the best solution is reported from a set of runs with different parameter values, this should be clearly identified as such.

The rate at which heuristics converge to a solution close in value to that of the "best found" solution should be measured. A graph of the quality of solution versus the time expended, per Figure 2, illustrates this classic tradeoff. The quality-effort relationship can also be captured with descriptive measures, such as the ratio of time to produce a solution within 5 percent of the best-found solution value to the time to produce the best:

$$r_{0.05} = \frac{\text{time to within } 5\% \text{ of best}}{\text{time to best found}}.$$

Note that on different systems there may be several ways to measure time: including user, system, and real time. Parallel implementations introduce additional timing complications (see Barr and Hickman, 1993, for guidelines in this situation).

Running times, however, may not translate well from one computing system to another, so other measures of computational effort may be appropriate—especially in descriptive experiments. Combinatorial measures, such as data structure updates and nodes in the search tree, sometimes correlate well with running times and are more system- and programmer-independent (Ahuja, Magnanti, and Orlin, 1993; Hooker, 1995; McGeoch, 1992). Counting operations, major subtasks, and memory requirements mimics the traditional theoretical analysis of algorithms (Garey and Johnson, 1979). The benefits of this approach are many, producing experiments in which factors such as programming skills, timing, tuning, and test set selection are all irrelevant. In theory, however, results are asymptotic and often one needs very large instances to achieve the behavior experimentally.

4.1.3. Robustness. Clearly, a heuristic that can only obtain an excellent solution for only one instance of a problem is not robust and arguably not very interesting. Generally, robustness is based on the ability of a heuristic to perform well over a wide range of test problems and is usually captured through measures of variability. For example, the quality-versus-time graph in Figure 4 also includes standard deviation bars with the average quality points.

Furthermore, heuristic strategies and parameters should either remain constant over the set of test problems or should be automatically set using individual test problem attributes.

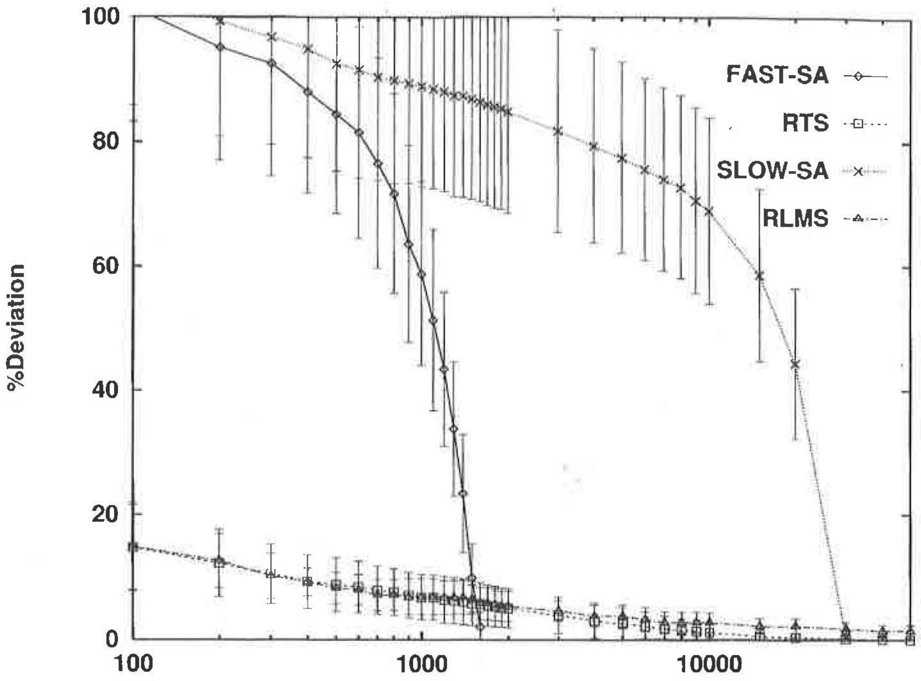

Figure 4. Average quality percent deviation from optimal versus computational effort for various heuristics on a QAP, with sigma bars (adapted from Battiti and Tecchiolli, 1994).

Robustness should be demonstrated prior to fine-tuning a heuristic for a single problem instance. Where parameter values are chosen, some measure of the sensitivity of the heuristic performance to small changes in parameter settings will indicate algorithm robustness and should be included (see Stewart, Kelly, and Laguna, 1995).

Authors are encouraged to report negative results. For example, if a heuristic performs well over a wide range of problems but fails on a specific type, then the authors should report, rather than hide, these results. Alternatively, if a heuristic does not guarantee feasibility (e.g., see Gendreau et al., 1994; Kelly et al., 1992, 1993), the authors should report on those cases where the heuristic does not generate a feasible solution.

4.1.4. Selection of Measures. How performance is measured and ultimately computed is as important as the measure itself. McGeoch (1995) notes that reliable prediction of running times as a function of different performance measures over a wide spectrum of computing systems remains an open problem. She offers a list of guidelines for finding good performance measures, including: look at data representing differences as well as ratios, since these often have different properties; use measures that have a small variance within a sampling point as compared to the variance observed between different sampling points; and apply variance reduction techniques (Bratley, Fox, and Schrage, 1983; McGeoch, 1992) to suggest measure choices.

4.2. Factors to Study

There are three main categories of factors that affect the performance of algorithms in computational experiments: problem, algorithm, and test environment. Since each category contains a multiplicity of influences on the test results, the investigator must be judicious in selecting

- Which factors to study (such as problem size, heuristic method, number of processors),
- Which to fix at some level (such as problem class, stopping rule, computing environment), and
- Which to ignore, and hope that they will not influence the experimental results (such as distribution of problem costs, system job mix).

The choice of experimentation factors is central to both comparative and descriptive experiments. We urge researchers to carefully consider this decision in designing an experiment. All factors, studied or otherwise, will need to be documented and addressed at the reporting stage.

4.2.1. Problem Factors. A variety of problem characteristics—such as dimensions, structure, parametric distributions—can affect the results of a given observation. Their effects, if any, are important for assessing a code's robustness.

At a minimum, the effect of problem sizes (such as number of variables and equations) should be included, but many other parоblem characteristics can have a significant impact as well, such as problem parameter distributions (Glover et al., 1974) and data presentation order in neural networks. Some factors can be controlled easily, as when certain problem generators are used (see Section 5.2), and can be analyzed accurately with statistical experimental design techniques.

It is important to "stress test" the codes by runnng as large instances as possible. Many factors do not show up on small instances, and running on smaller sizes may not yield accurate predictions for larger, more realistic problems.

4.2.2. Algorithm Factors. Algorithm factors include the selection of heuristics and computer codes to test (in comparative experiments) and the internal control settings to use. Well-known codes provide helpful baselines and should be included where relevant and feasible.

The application of a heuristic method usually involves algorithm-specific choices, since a heuristic typically contains multiple strategies and multiple parameters that control these strategies (such as initial-solution-construction procedures and associated search parameters). Readers are not only interested in the final results but in the relative contributions of the various strategies. Computational testing should demonstrate the contribution and computational burden of each strategy within a complex heuristic, so as to identify innovative ideas that may be used in other contexts.

Values of any parameters employed by the heuristic should be defined and, where problem-dependent, the rules for establishing appropriate values should be specified. Whether these values are fixed or computed, there should be a reliable way of determining effective parameter settings for a new instance of the same problem class.

Since performance is tied to the strategy and parameter choices, much research effort can be spent in choosing the appropriate options and parameter settings. The process used to make these choices is of interest to the reader and should be documented. The process may involve some sampling and statistical analysis—such as design optimization methodologies (Barton and Ivey, Jr., 1996; Box and Draper, 1969; Mason et al., 1989; Nelder and Mead, 1965; Taguchi and Wu, 1979) or machine learning (Nygard, Juell, and Kadaba, 1990). Robust heuristic that perform well over a range of parameter values are generally superior to heuristics that require unique settings for every problem instance, unless the heuristic is designed to self-adjust based on problem characteristics. Parameter sensitivity analyses are useful in evaluating robustness.

One algorithm factor of this type that *must* be addressed is the stopping rule. Since many techniques do not have a generally accepted termination criterion, the investigator has great discretion in selecting one. An unethical approach is to make lengthy runs of the code and then devise a self-serving rule. Documenting the process used to devise the criterion will help justify its use.

4.2.3. Test Environment Factors. Ideally, competing algorithms would be coded by the same expert programmer and run on the same test problems on the same computer configuration. The results of these runs in terms of time to solution and quality of the solution produced on each problem instance would be directly comparable for the two heuristics.

In this manner, the wide range of environmental factors could be controlled, including: hardware (brand, model, size of memory, CPU speed, number of processors, processor communication scheme), software (operating system, language, compiler, compiler settings), system (job mix), and programmer (coding expertise, tuning ability) factors. Since this is often not the case, the experimenter must determine the effect of those factors that are not held uniform across the runs or, at a minimum, identify those that vary and might influence the results.

5. Designing and Executing the Experiment

Experimental design is the process of planning an experiment to ensure that the appropriate data will be collected. A good experimental design is unbiased, achieves the experimental goals, clearly demonstrates the performance of the tested process, uncovers reasons for performance, has a justifiable rationale, generates supportable conclusions, and is reproducible. All of these characteristics are of particular value in the testing of heuristic methods.

5.1. Choosing an Experimental Design

The best means of achieving such results are through the use of well-known statistical experimental design methods (Mason, Gunst, and Hess, 1989; Montgomery, 1984). *Statistical design of experiments* (DOE) is an approach to experimental design that ensures that the collected data can be analyzed by statistical methods to reach valid and objective conclusions. DOE is a process for collecting data and analyzing it with an established statistical model, such as full-factorial or Latin-squares. When an experiment's results can vary with the test conditions (problems solved, computer, parameter settings), statistical methodology is the only objective approach to analysis. Hence, the design of the experiment and the statistical analysis of the data are interrelated.

DOE is a simple process for structuring and analyzing experiments and is designed to minimize the testing effort and maximize the information acquired. It is based on the principles of replication (repeating tests), randomization (performing tests in random order to offset unincluded factors), and blocking (eliminating the influence of known, but extraneous, factors). DOE also has the side benefit of automatically constructing a model of the process or algorithm being examined, thus it is useful for descriptive studies. See Amini and Barr (1990) for a concise introduction to this methodology for operations researchers.

In the physical sciences, engineering, and medicine, DOE is employed routinely. Standards for empirical testing in the computing and mathematical sciences have been much less rigorous, and there is a broad range of accepted practice. Despite exhortations for high standards in reporting, demonstration or proof-of-concept studies with minimal or ad hoc testing are more the norm than carefully constructed experimental designs with statistically validated conclusions. (As a step toward changing this, it is our opinion that all doctoral students of operations research should receive training in DOE and employ it in any empirical research.) However, see Amini and Barr (1990), Barton and Ivey, Jr. (1996), Lin and Rardin (1979), and Nance, Moose, and Foutz (1987) for examples of using DOE to test optimization and heuristic algorithms.

Results on standard benchmark problems and problems of practical interest should always be a part of an algorithmic experiment, even though they are not part of a formal design. These results provide valuable points of reference, even in noncomparative settings. However, as observed in Hooker (1995), the definition of "standard" should be extended. We will comment on this more later.

5.2. Selecting Test Problems

The choice of test problems follows directly from the experimental design. Problem sets may be drawn from practical settings or created by a researcher with problem generation software.

Real-world problems reflect the ultimate purpose of heuristic methods, and are important for assessing the effectiveness of a given approach. Of particular value are instances that are representative, in some manner, of those encountered in a given problem domain. If the situation permits, problems should be collected with factors controlled per some statistical experimental design.

However, in some cases it can be important to develop special test problems to test particular performance features and particular response patterns of a method. Although "artificial" problems are occasionally criticized for being "unrealistic" and more difficult to solve than "real" ones (O'Neill, 1982; Rardin and Lin, 1982), problem-generation software has some definite advantages. Generators typically give the user control over problem characteristics (Arthur and Frendewey, 1994; Klingman, Napier, and Stutz, 1974), thereby allowing creation of instances with a specific configuration of experimental design factors (Klingman and Mote, 1987). Some codes supply an optimal solution or its value (Arthur and Frendewey, 1988; Lin and Rardin, 1977). If written for machine independence and portability, generators also provide an efficient means of distributing and reproducing problems.

If problems are created by the developer, then the generation process should be clearly described and the newly generated problems should be archived (or the process for generating them should be reproducible) for use by other researchers. A variety of different types of problems should be generated to reflect the diversity of factors that could be encountered. Generated problem instances should be representative of problems likely to be found in the field or designed to explore key hypotheses. The investigator should also attempt to assess the overall difficulty of the problems created since, in some cases, problem characteristics make problems easy to solve by just about any heuristic.

For many classes of problems [such as traveling salesman, bin packing, set covering, global optimization (Floudas and Pardalos, 1990)] well-studied test suites are available in the literature and in electronic form [such as TSPLIB (Reinelt, 1991)]. A new heuristic should be tested on all "standard" problems, whether generated or drawn from practice, for which it was designed. This permits comparison with other published results, even with descriptive experiments.

In general, the more test problems evaluated, the more informative the study. Of interest are instances that stress, or test the limits of, the software or cause it to fail.

5.3. Executing the Experiment

This is the data-collection step in the process, where specific computer runs are made. The investigator must ensure that the experimental design is being followed, and that the relevant process details are documented. Of particular concern are:

- Randomization—performing the tests in the designated random order,
- Uniform computing environment—maintaining as uniform a state of the test environment as possible, if it can have an effect on the performance measures, such as job execution times.

These elements help reduce variablity and the influence of unknown factors. For example, if solution times vary with the system's job mix, data should be collected during lightly loaded or dedicated periods.

6. Analyzing the Results and Drawing Conclusions

This phase of the experiment converts the collected data into information through analysis and interpretation. *Data analysis* refers to evaluating the recorded empirical data with statistical and nonstatistical techniques with respect to the experiment's purpose and goals. At this point, the requisite data should be available to test the hypotheses, estimate population parameters, uncover pertinent relationships, verify assumptions, and measure variability.

Analysts should address the experimental goals and the questions they posed, such as those in Section 4.1. In particular, consider the key tradeoffs (such as solution quality versus time, speed versus robustness) and attempt to identify the factors or factor combinations that seem to contribute to (are correlated to) performance. For example, the rate at which time grows with problem size, can be quite useful in assessing how a heuristic will perform on large problem instances. A regression model that relates run times to problem size can characterize this empirical growth (Stewart, 1987).

Data analysis tools include general statistical packages, DOE software (if a standard experimental design model was used), data visualization software, as well as manual methods and human pattern recognition. Graphical displays can provide especially persuasive insight into the data and the relationships that they contain. Graphics offer a means of visualizing all of the data, and greater understanding can come from examining the entire data set rather than just summary statistics.

Statistical methods should be employed wherever possible to indicate the strength of relationships between different factors and performance measures (e.g., see Amini and Barr, 1990; Golden et al., 1986; Lin and Rardin, 1979; McGeoch, 1995; Nance, Moose, and Foutz, 1987). While they cannot prove causality, these methods do indicate the reliability and validity of the results. If testing involved benchmark problems that were not part of a designed experiment, there is no standard statistical means of analyzing the data (see Barton and Ivey, Jr., 1996; Golden and Stewart, 1985, for representative approaches to this issue), although visual analysis can always be employed.

Once the data has been analyzed, the results are *interpreted* as a series of conclusions and inferences, deduced from the collected evidence. The statistical and practical significance of these conclusions should be addressed, their implications evaluated, and recommendations made. The recommendations often include further experiments to answer questions suggested by the data. Experimentation tends to be an iterative process, with each round expanding knowledge of the algorithm, but leading to new questions and, sometimes, the need for new performance measures.

The analysis and interpretation steps are the culmination of all the planning and implementation activities and, in the end, determine the overall merit of the work. The final reporting step is needed to document the experimental details and findings and communicate them to the research and practice communities.

7. Reporting on Computational Experiments with Heuristics: Guidelines for Investigators

What should be reported? Enough information to convince a reader that the experiment has scientific merit, is reproducible, and answers the important questions posed at its outset. In testing a new heuristic, the researcher often wishes to demonstrate that it makes a contribution as described in Section 3.

In this section, we provide guidelines to help authors report on their computational testing, and referees and editors evaluate the results. Table 2 summarizes the main points, which are detailed below. In a nutshell, we ask the conscientious researcher to thoroughly document the experimentation and all relevant factors, carefully analyze the resulting data, and present unbiased, supportable conclusions.

Table 2. Guidelines for reporting computational results.

1. *Reproducibility* is essential: document to allow replication of results.
2. *Specify* all influential factors in detail: heuristic, code, parameters, pseudo-random numbers, input data, nontrivial data structures, and computing environment.
3. Be precise about *timing*.
4. Show how algorithm *parameters* are set.
5. Use *statistical* experimental design techniques.
6. *Compare* the heuristic with other methods.
7. Reduce *variability* of results.
8. Produce a comprehensive *report* of the results.

7.1. Reproducibility

An essential ingredient of scientific research is reproducibility, and experimental results that cannot be independently verified are given little credence in the scientific community. To be reproduced, an experiment must be thoroughly documented. Hence, when reporting on heuristic testing, the algorithm and its implementation (the code) should be described in sufficient detail to allow replication, including any parameters, pseudo-random number streams, and nontrivial data structures employed. The sources and characteristics of problem instances should be documented—including all details of any generation scheme—and nonproprietary problems made available to other researchers. Making available to researchers any developed code will also enhance the scientific merit of the work.

7.2. Computing Environment

Many test-environment factors can influence the empirical performance of a heuristic method, and should be documented, including

- Model and make of the computer,
- Number, types, and speeds of processors,
- Size and configuration of main, cache, and swap memories,
- Interprocessor scheme and communication times,
- Operating system name, vendor, and version,
- Programming languages and compliers along with compiler settings and libraries linked to the load module,
- System load, where applicable.

Also detailed should be the computing resources—notably time and primary and secondary memory—required by the algorithm itself, expressed as a function of problems parameters, such as size. If the machine is not well-known, a rough comparison of its speed against popular systems is useful.

7.3. Timing

As mentioned in earlier sections, heuristic methods are used instead of exact methods because of their ability to produce usable, albeit nonoptimal, solutions considerably faster than exact methods. Therefore, accurate reporting of timings for experiments is of the utmost importance. How times are measured should be documented. Authors should differentiate between user, system, and real (wall clock) times, and report system and real times if system time is significant or real time is much greater than the sum of user and system times. The reader should understand at what points in the code the timings were recorded. Authors should keep in mind that system clock routines usually have a much coarser resolution than the internal processor clock.

When reporting times, authors should be clear about which operations are included and which, if any, are excluded. In general, times should be reported for

- "Overhead" operations required by the algorithm, such as problem setup and preprocessing of data: Many problems involve substantial computations (for example, distance matrix calculation) before the heuristic is started.
- Each stage of a heuristic when there are multiple phases such as occur when the heuristic calculates an initial feasible solution and then searches for local improvement.
- Total time and time to the best solution: When there is a probabilistic component to a heuristic (such as random starting points), the times for all runs, those that produced the solutions being reported along with all others, should be included as part of the time required to find the best solution. The total time that a complex heuristic is allowed to run before termination should be included with the time for the best solution.
- Calibration routines: If standard timing codes are available and appropriate, their results in the experimental computing environment should be documented.

7.4. Quality of Solution

Even if the main thrust of the experiment is descriptive, the author must, when possible, measure the closeness of the heuristic's solution to the optimal solution. While this can be demonstrated by a theoretical worst-case analysis (Fisher, 1980), the usual thrust of computational testing is the demonstration of an algorithm's typical or average solution quality, relative to the optimal solution value, and the consistency of its behavior:

- Where the optimal solution is known, the heuristic solution can be directly compared as a measure of the effectiveness of the heuristic.

- Where optimal solutions are unknown or unobtainable by current methods, some other benchmark of performance should be offered by the author, such as a comparison to a tight lower (upper) bound (see Johnson, 1990; Johnson et al., 1995), or comparison to published solution values on publicly available test problems.
- Where possible, some insight should be provided as to how the quality of solution holds up as problem instances grow in size or complexity. This is a complementary measure to the time and size characterization function discussed in Section 6. The performance of some heuristic procedures (such as percent above optimality) deteriorate as the problem size grows.

7.5. Parameter Selection

If the code allows for different choices of algorithm contol (tuning) parameters, the report should specify the parameter settings and how they were chosen. In particular, the values for any parameters associated with a stopping rule *must* be documented and justified. Other issues to be addressed include

- The parameter values, or computation rule, used in solving each reported problem instance: Fixing the values or rules is preferable to ad hoc tuning for each problem.
- The process by which the parameter values were selected: Generally, some experimentation and statistical analysis is appropriate in this undertaking.
- Where parameter values differ for different problem instances, the reason for these differences: If there is a rule for deriving parameter settings from the characteristics of a problem instance, that rule needs to be specified and justified. Where the solutions reported represent the best solution observed from runs with several different parameter settings, the run times reported for each problem instance should be either (1) the average of the run times for all parameter settings applied to that problem, or (2) the times from both "standard" and "hand-tuned" settings, along with the number of runs made.
- Evidence that the parameter values presented are generalizeable to other problem instances and are insensitive to minor changes in their values: This is used to assess the robustness of a parameter-driven heuristic. When more than one parameter is present, the reported analysis should assess the importance of each to the performance of the heuristic and their degree of interaction.
- An estimate of the time required to fine-tune the algorithm: This estimate should give an idea of the effort required to develop the heuristic versus the time needed to find robust parameter values. For example, some heuristics may be conceived or implemented relatively quickly (in one or two weeks) but require substantial time to fine-tune (one or two months).

The selection of parameter values that drive heuristics is itself a scientific endeavor and deserves more attention than it has received in the operations research literature. This is an area where the scientific method and statistical analysis could and should be employed.

7.6. Statistics

Statistical experimental design techniques are powerful, often neglected methodologies that can highlight those factors that contribute to the results, as well as those that do not. To help readers better understand the effect of these factors on the output measure (such as running time or solution quality), experiments should be well-planned and standard experimental design techniques adopted.

Where the main thrust of the research is to demonstrate that a particular heuristic outperforms another in one or more dimensions, a statistical comparison of results should be included. Where sample sizes are large enough, this may take the form of *t*-tests or an analysis of variance. When the assumptions for parametric tests fail to hold, there are a host of nonparametric techniques (for example, the sign test) that can and should be employed to make the author's arguments, even if the result is that there is no statistical difference between the quality of the solutions produced by the heuristics under study (e.g., Golden and Stewart, 1985).

Since more information can be conveyed if results obtained are contrasted with other methods, authors are encouraged to identify points of reference to make such comparisons. Whenever possible, the "best" competing codes should be used. Published results can be used, but if well-known (publicly released) codes are available, it is preferable to repeat the experiment, even if this requires conducting runs in different computing environments. If no publicly released code exists or can be obtained, authors are encouraged to implement another method, whether a previously described heuristic or a simple-minded heuristic (such as a greedy method) to make the contrast.

7.7. Variability

Experiments should be designed to reduce variability of the measured results, either by running longer experiments (producing more data points) or running the final experiments on a dedicated or lightly loaded machine. Experiments should be conducted on a variety of problem classes and on many instances in each class. If pseudo-random number generators are used, the code should be tested on each instance with different seed initializations.

7.8. Analysis and Interpretation

Reporting should not only present the experimental results, but give an analysis and state the investigator's conclusions. For example, it is not sufficient to present tables of running times for the reader to interpret. The analysis should describe the effects of factors, individually and in combination, on the chosen performance measures. The conclusions should address the questions posed and hypotheses tested, and indicate the significance and impact of the experimental outcomes.

In particular, the central tradeoffs should be explored, especially solution quality versus computational effort (see Figure 5). Other tradeoffs, such as time versus problem size (per Figure 6) and robustness versus quality, lend insight into the behavior of the algorithm(s)

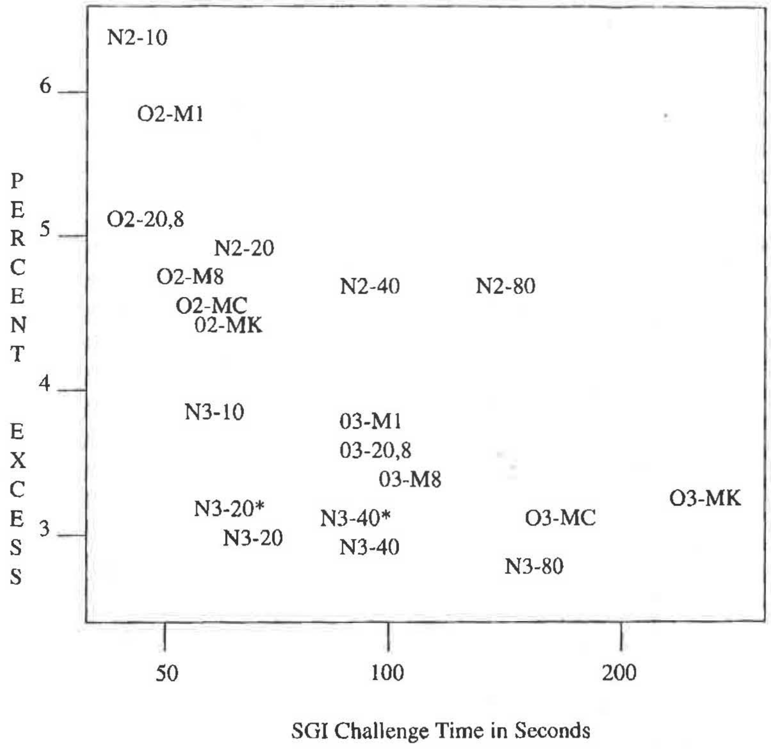

Figure 5. Average quality, in percent over lower bound, versus time for 20 heuristics on 10,000-city TSPs (from Johnson et al., 1995).

(see Golden and Stewart, 1985). Researchers are also interested in benefit-cost analyses, such as time and memory tradeoffs and the contribution of individual stages in a multi-stage heuristic.

In discussing their data, authors should highlight unexpected or anomalous results. Where possible they should attempt to explain these (ideally by more focused experimentation, perhaps based on other performance measures), otherwise they should be left as problems worthy of further investigation.

7.9. Report

A comprehensive report of the results of computational testing should include most of the aforementioned information. Results should be summarized using measures of central tendency and variability. Some attention should be given to a presentation of cost effectiveness of the heuristic method, such as a plot of the quality of solution versus either time or algorithmic iterations. The ratios mentioned in Section 4 offer numerical measures of the rate of convergence of a heuristic method.

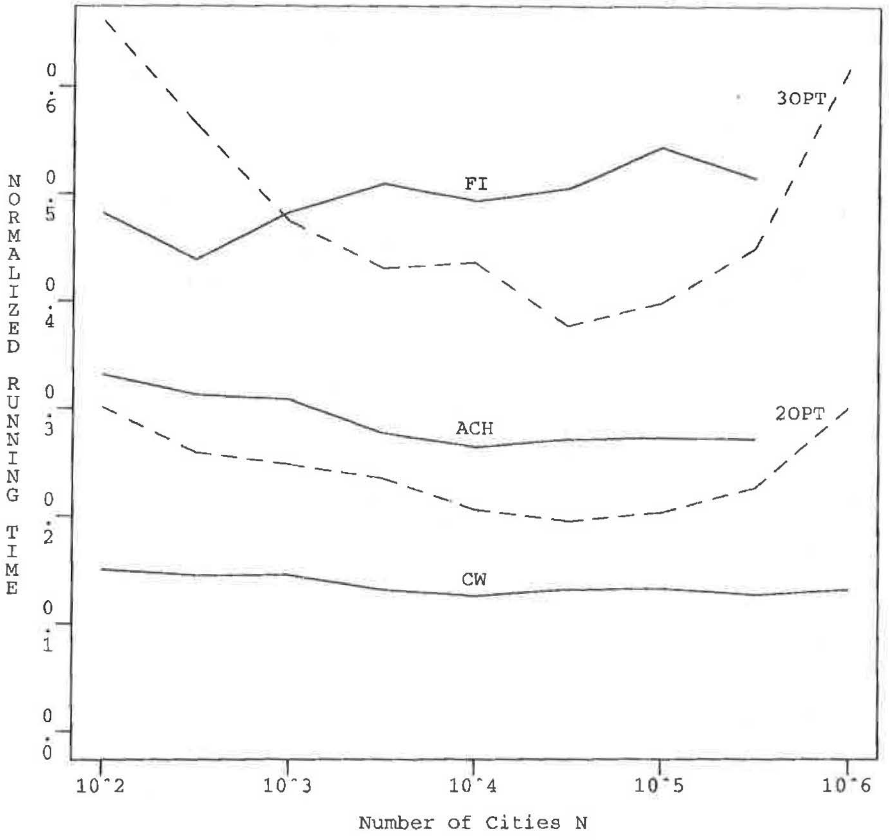

Figure 6, Running time (normalized by *N* log *N*) versus problem size, *N*, for five TSP heuristics (from Johnson et al., 1995).

In presenting results, graphic displays can be highly effective and inviting to the reader. Good graphics software with a wide range of output styles is available to most researchers. Be creative and consider new means of communicating the results (see Tufte, 1983, for inspiration).

In addition to including as much information as possible in the published paper, supplementary material should be made available to other researchers via working papers, preferably in machine-readable format via the Internet. This includes source codes, input and output files, and complete solutions to the problem instances reported in the paper, particularly when reporting on new problem classes or improved solutions for standard test problems[1] (see Gendreau, Hertz, and Laporte, 1994; Stewart, Kelly, and Laguna, 1995).

8. Conclusions and Future Directions

This article considered the issue of reporting experimental (that is, computational) research with heuristics. Rigorous reporting of experimental studies in computing and the mathematical sciences is still in its infancy and much progress is needed to achieve the level of quality commonly observed in more established experimental sciences. We have provided some thoughts on how such a path to rigorous experimental research and reporting can be established, listing a preliminary set of guidelines that may help authors carry out and report their research, and editors and referees evaluate it.

In recent years, a need for a rigorous empirical science of algorithms has been recognized (Bland et al., 1993; Hooker, 1994; McGeoch, 1986). The growth of the Internet has helped to promote the interchange of information and sharing of data and codes by researchers. On-line repositories of test problems have come into existence (such as TSPLIB, QAPLIB, Netlib, MIPLIB, libraries of the DIMACS Algorithm Implementation Challenges, and OR-Library). However, a need for larger, more accessible, computer-based archives of benchmark problems, test beds, and codes exists.

We encourage the *Journal of Heuristics* to provide authors with support for experimental research on heuristics. This support could consist of a World Wide Web site on which researchers could have direct or linked access to

- Libraries of standard test problems,
- Source codes of publicly released implementations of heuristics and other algorithms,
- Benchmark routines for system clock calibration.

The study of heuristic methods is a flourshing and fruitful area for algorithmic research (see Zanakis, Evans, and Vazacopoulos, 1989; Reeves, 1993b). We feel that conscientious reporting is at the heart of real progress and hope that the thoughts collected here will encourage all researchers to advance the practice of experimentation and reporting in this exciting field.

Acknowledgments

We appreciate the contributions of David Johnson and Manuel Laguna in the preparation of this article. We also thank the creators of the figures for allowing us to include their works and Fred Glover for the opportunity to be a part of this important new journal.

Note

1. This recommendation is based on the following. There are several heuristic results reported in the literature that researchers have been unable to replicate. Unfortunately, the solutions that purportedly produced these objective function values are unavailable, and there is no way of determining whether the reported values are correct or are simply the result of typographical errors.

References

Aarts, E., van Laarhoven, P., Lenstra, J., and Ulder, N. (1994). A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, 6(2), 118–125.

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice Hall.

Amini, M., and Barr, R. (1990). Network reoptimization algorithms: A statistically designed comparison. *ORSA Journal on Computing*, 5(4), 395–409.

Arthur, J., and Frendewey, J. (1988). Generating traveling salemen problems with known optimal tours. *Journal of the Operational Research Society*, 39(2), 153–159.

Arthur, J., and Frendewey, J. (1994). An algorithm for generating minimum cost network flow problems with specific structure and known optimal solutions. *Networks*, 24(8), 445–454.

Barr, R., and Hickman, B. (1993). Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinions. *ORSA Journal on Computing*, 5(1), 2–18.

Barr, R., and Hickman, B. (1994). Parallelization strategies for the network simplex algorithm. *Operations Research*, 42(1), 65–80.

Barton, R., and Ivey, Jr., J. (1996). Nelder-mead simplex modifications for simulation optimization. Tech. rep., Department of Industrial and Systems Engineering, Pennsylvania State University, University Park, PA. To appear in *Management Science*.

Battiti, R., and Tecchiolli, G. (1994). Simulated annealing and tabu search in the long run: A comparison on gap tasks. *Computers and Mathematics with Applications*, 28(6), 1–8.

Bland, R., Cheriyan, J., Jensen, D., and Ladányi, L. (1993). An empirical study of min cost flow algorithms. In D. Johnson, and C. McGeoch (Eds.), *Network Flows and Matching: First DIMACS Implementation Challenge*, Vol. 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (pp. 119–156). Providence, RI: American Mathematical Society.

Box, G., and Draper, N. (1969). *Evolutionary Operation, A Statistical Method for Process Improvement*. New York: John Wiley.

Bratley, P., Fox, B., and Schrage, L. (1983). *A Guide to Simulation*. New York: Springer-Verlag.

Cornuejols, G., Sridharan, R., and Thizy, J. (1991). A comparison of heuristics and relaxations for the capacititated plant location problem. *European Journal of Operational Research*, 50, 280–297.

Crowder, H., Dembo, R., and Mulvey, J. (1980). On reporting computational experiments with mathematical software. *ACM Transactions on Mathematical Software*, 5, 193–203.

Dyer, M., and Frieze, A. (1985). A simple heuristic for the *p*-centre problem. *Operations Research Letters*, 3(6), 285–288.

Feo, T., and Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, 109–133.

Fisher, M. (1980). Worst-case analysis of heuristic algorithms. *Management Science*, 26(1), 1–17.

Floudas, C., and Pardalos, P. (1990). *Collection of Test Problems for Constrained Global Optimization Algorithms*, Vol. 455 of *Lecture Notes in Computer Science*. Springer-Verlag.

Garey, M., and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10), 1276–1290.

Gilsinn, J., Hoffman, K., Jackson, R., Leyendecker, E., Saunders, P., and Shier, D. (1977). Methodology and analysis for comparing discrete linear l_1 approximation codes. *Communications in Statistics*, 136, 399–413.

Glover, F. (1989). Tabu search—part I. *ORSA Journal on Computing*, 1(3), 190–206.

Glover, F., Karney, D., Klingman, D., and Napier, A. (1974). A computational study on start procedures, basis change criteria, and solution algorithms for transportation problems. *Management Science*, 20, 793–813.

Golden, B., Assad, A., Wasil, E., and Baker, E. (1986). Experimentation in optimization. *European Journal of Operational Research*, 27, 1–16.

Golden, B., and Stewart, W. (1985). Empirical analysis of heuristics. In E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys (Eds.), *The Travelling Salesman Problem, a Guided Tour of Combinatorial Optimization* (pp. 207–249). Chichester (U.K.): Wiley.

Greenberg, H. (1990). Computational testing: Why, how and how much? *ORSA Journal on Computing*, 2, 7–11.

Held, M., and Karp, R. (1970). The travelling-salesman problem and minimum spanning trees. *Operations Research*, 18, 1138–1162.

Held, M., and Karp, R. (1971). The travelling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1, 6–25.

Hochbaum, D., and Shmoys, D. (1985). A best possible heuristic for the *k*-center problem. *Mathematics of Operations Research*, 10(2), 180–184.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

Hooker, J. (1994). Needed: An empirical science of algorithms. *Operations Research*, 42(2), 201–212.

Hooker, J. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1), 33–42.

Hopfield, J., and Tank, D. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52, 141.

Jackson, R., Boggs, P., Nash, S., and Powell, S. (1990). Report of the ad hoc committee to revise the guidelines for reporting computational experiments in mathematical programming. *Mathematical Programing*, 49, 413–425.

Jackson, R., and Mulvey, J. (1978). A critical review of comparisons of mathematical programming algorithms and software (1953–1977). *J. Research of the National Bureau of Standards*, 83, 563–584.

Johnson, D. (1990). Local optimization and the traveling salesman problem. In *Proceedings of the 17th Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 443 (pp. 446-461). Berlin: Springer-Verlag.

Johnson, D., Bentley, J., McGeoch, L., and Rothberg, E. (1995). Near-optimal solutions to very lage traveling salesman problems. Tech. rep., monograph, in preparation.

Johnson, D., and Papadimitrious, C. (1985). Performance guarantees for heuristics. In E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys (Eds.), *The Travelling Salesman Problem, A Guided Tour of Combinatorial Optimization* (pp. 145–180). Chichester (U.K.): Wiley.

Kelly, J., Golden, B., and Assad, A. (1992). Cell suppression: Disclosure protection for sensitive tabular data. *Networks*, 22(4), 397–417.

Kelly, J., Golden, B., and Assad, A. (1993). Large-scale controlled rounding using tabu search and strategic oscillation. *Annals of Operations Research*, 41, 69–84.

Klingman, D., and Mote, J. (1987). Computational analysis of large-scale pure networks. Presented at the Joint National Meeting of ORSA/TIMS, New Orleans.

Klingman, D., Napier, A., and Stutz, J. (1974). Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20, 814–821.

Knox, J. (1994). Tabu search performance on the symmetric travelling salesman problem. *Computers & Operations Research*, 21(8), 867–876.

Lawler, E., Lenstra, J., Kan, A. Rinnooy, and Shmoys, D. (1985). *The Travelling Salesman Problem, A Guided Tour of Combinatorial Optimization*. Chichester (U.K.): Wiley.

Lin, B., and Rardin, R. (1977). Development of a parametric generating procedure for integer programming test problems. *Journal of the ACM*, 24, 465–472.

Lin, B., and Rardin, R. (1979). Controlled experimental design for statistical comparison of integer programming algorithms. *Management Science*, 25(12), 33–43.

Lin, S., and Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2), 498–516.

Martello, S., and Toth, P. (1990). *Knapsack Problems*. Chichester (U.K.): Wiley.

Mason, R., Gunst, R., and Hess, J. (1989). *Statistical Design and Analysis of Experiments*. New York: Wiley.

McGeoch, C. (1986). *Experimental Analysis of Algorithms*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

McGeoch, C. (1995). Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, to appear.

McGeoch, C. (1992). Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups. *ACM Computing Surveys*, 24(5), 195–212.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21, 1087–1091.

Montgomery, D. (1984). *Design and Analysis of Experiments*. New York: Wiley.

Mulvey, J. (1982). *Evaluating Mathematical Programming Techniques*. Berlin: Springer-Verlag.

Nance, R., Moose, Jr., R., and Foutz, R. (1987). A statistical technique for comparing heuristics: An example from capacity assignment strategies in computer network design. *Communications of the ACM*, 30(5), 430–442.

Nelder, J., and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7, 308–313.

Nygard, K., Juell, P., and Kadaba, N. (1990). Neural networks for selecting vehicle routing heuristics. *ORSA Journal on Computing*, 2(4), 353–364.

O'Neill, R. (1982). A comparison of real-world linear programs and their randomly generated analogs. In J. Mulvey (Ed.), *Evaluating Mathematical Programming Techniques* (pp. 44–59). Berlin: Springer-Verlag.

Rardin, R., and Lin, B. (1982). Test problems for computational experiments—issues and techniques. In J. Mulvey (Ed.), *Evaluating Mathematical Programming Techniques* (pp. 8–15). Berlin: Springer-Verlag.

Reeves, C. (1993a). Evaluation of heuristic performance. In C. Reeves, (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. New York: Wiley.

Reeves, C. (1993b). *Evaluation of Heuristic Performance*. New York: Wiley.

Reinelt, G. (1991). TSPLIB—a travelling salesman problem library. *ORSA Journal on Computing*, 3(4), 376–384.

Resende, M., and Ribeiro, C. (1995). A GRASP for graph planarization. Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ.

Rothfarb, B., Frank, H., Rosebaum, D., Steiglitz, K., and Kleitman, D. (1970). Optimal design of offshore natural gas pipeline systems. *Operations Research*, 18, 992–1020.

Stewart, W. (1987). An accelerated branch exchange heuristic for the traveling salesman problem. *Networks*, 17, 423–437.

Stewart, W., Kelly, J., and Laguna, M. (1995). Solving vehicle routing problems using generalized assignments and tabu search. Tech. rep., Graduate School of Business, College of William and Mary, Williamsburg, VA.

Taguchi, G., and Wu, Y.-I. (1979). *Introduction to Off-Line Quality Control*. Central Japan Quality Control Association, Meieki Nakamura-Ku Magaya, Japan.

Tufte, E. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press.

Zanakis, S., Evans, J., and Vazacopoulos, A. (1989). Heuristic methods and applications: A categorized survey. *European Journal of Operational Research*, 43, 88–110.