Journal of Heuristics, 7: 261–304 (2001)

Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial

RONALD L. RARDIN AND REHA UZSOY
School of Industrial Engineering, Purdue University, West Lafayette, IN 47907-1287, USA

Abstract

Heuristic optimization algorithms seek good feasible solutions to optimization problems in circumstances where the complexities of the problem or the limited time available for solution do not allow exact solution. Although worst case and probabilistic analysis of algorithms have produced insight on some classic models, most of the heuristics developed for large optimization problem must be evaluated empirically—by applying procedures to a collection of specific instances and comparing the observed solution quality and computational burden.

This paper focuses on the methodological issues that must be confronted by researchers undertaking such experimental evaluations of heuristics, including experimental design, sources of test instances, measures of algorithmic performance, analysis of results, and presentation in papers and talks. The questions are difficult, and there are no clear right answers. We seek only to highlight the main issues, present alternative ways of addressing them under different circumstances, and caution about pitfalls to avoid.

Key Words: Heuristic optimization, computational experiments

1. Introduction

Heuristic optimization algorithms (heuristisics for short) seek good feasible solutions to optimization problems in circumstances where the complexity of the problem or the limited time available for its solution do not allow exact solution. The formal intractability, in the sense of *NP*-hardness (Garey and Johnson, 1979), of many commonly encountered optimization problems and the growing use of real-time control have made the development of heuristics a major area within the field of operations research.

Unlike exact algorithms, where time-efficiency is the main measure of success, there are two burning issues in evaluating heuristics: how fast can solutions be obtained and how close do they come to being optimal. One body of research, beginning with Graham's investigations of heuristics for parallel processor scheduling problems (Graham, 1969), demands polynomially bounded time and seeks provable limits on the worst-case or maximum deviation of a heuristic from the optimal solution value. Another, pioneered by Karp (1977), tries to compute the expected deviation from optimality, assuming polynomial time and a probabilistic structure for the problem instances considered.

Both these approaches have yielded significant insights into a number of heuristics and problems, and both have the appeal of mathematical certainty. Still, their analytical difficulty, which makes it hard to obtain results for most realistic problems and algorithms, severely limits their range of application. Also, a worst-case result, which is by definition

pathological, may not give meaningful information on how a heuristic will perform on more representative instances. Stochastic analyses attempt to discount such low probability pathologies, but very simple underlying distributions on instances usually have to be assumed to get results.

As a consequence of these difficulties, most of the many heuristics developed for large optimization problems are evaluated *empirically*—by applying the procedures to a collection of specific instances and comparing the observed solution quality and computational burden. While this has been done in the literature one way and another for decades, learning by experimentation does not come as naturally to algorithm designers, who are trained in the mathematics culture, as it does to investigators in other branches of engineering and the sciences. It is still comparatively rare, given the large volume of literature on heuristics, to encounter well designed computational experiments that produce real insights for either practitioners or researchers.

There has been a long series of papers trying to set out guidelines for empirical research on algorithms (Ahuja and Orlin, 1996; Crowder et al., 1979; Lin and Rardin, 1980; Barr et al., 1994; Golden and Stewart, 1985; Golden et al., 1986; Greenberg, 1990; Jackson et al., 1991; Lee et al., 1993; Hooker, 1994 and 1995; McGeogh, 1996), most implicitly addressed to the testing of exact optimization methods. Although many of the issues that arise are similar to those encountered in evaluating exact procedures, the nature of heuristic optimization presents a number of unique challenges. In particular, we must often evaluate the quality of solutions when an exact optimum, or even a credible estimate of one, is unavailable.

Our intent with this paper is to focus squarely on empirical evaluation of heuristic optimization methods. The questions are difficult, and there are no clear right answers. We seek only to highlight the main issues and present a number of alternative ways of addressing them under different circumstances, as well as a number of pitfalls to avoid.

We begin in Section 2 by differentiating contexts within which one may perform experimentation on heuristics that we think should shape how the investigation is conducted. Next, Section 3 outlines the basic issues of computational experiment design. Sections 4 and 5 follow with detailed treatment of the two special challenges that confront heuristic investigators: obtaining suitable test instances and constructing performance measures. We offer a case study in Section 6 to to illustrate some of the issues already discussed and provide a setting for later analysis. Sections 7 and 8 then discuss the analysis and presentation of experimental results, including when and how we think statistical analysis can contribute. Finally, Section 9 provides some brief conclusions.

2. Evaluation contexts

Experimental investigations of heuristics are conducted for many reasons, and the manner in which they should be carried out depends on the context. Before beginning our discussion of design and analysis issues, it is useful to dwell on three of the most important contextual distinctions: research vs. development, design vs. planning vs. control applications, and the life cycle of the problem under study.

2.1. Research vs. development

Heuristics are tested for two rather distinct purposes, depending upon the goals of the investigator. *Research* testing is primarily addressed to discovering new technologies for existing problems or applying existing technology in creative ways to new problems. For example, investigators may be exploring a new approach to vehicle routing or an innovative scheme for facilities layout. The goal should be to gain insight into what works, what does not, and why this is so, at a level where the insights are transferable beyond the domain of the specific problem under study.

Testing in the *development* phase of heuristic development is quite different. The goal is to evolve the most efficient solution procedure for a specific environment. For example, experimenters may be adapting a well-known machine scheduling algorithm to a particular manufacturing setting, or refining a new piece of commercial software using a standard library of benchmark problems from the literature. Software implementation details and specifics of the application domain become much more central because the focus is on how to implement the algorithm efficiently and how to tune its parameters to yield the best results in the intended application.

Most of the algorithm testing done for scholarly publication is of the research type. Only rarely do such experiments lead immediately to implementation or commercialization. Still, researchers too often spend an enormous amount of effort on detailed issues of computer implementation. We agree with Hooker (1994) that excessive emphasis on such development issues during the research phase has diverted time and effort much better spent on the real goal of research—learning about the problem and algorithm under study.

2.2. Design, planning and control applications

Optimization tasks differ dramatically in the time available for solution and importance of getting the answer right. At one extreme are what we will call *design* problems which are solved relatively infrequently and seek answers that cover an extended time period. This definition encompasses a host of long-term problems such as telecommunication network design, group technology cell formation, and capacity expansion. With each, we will be spending a lot of money and living with our mistakes for a long time to come.

With such problems, exact algorithms are almost never available for instances of realistic size. However, there is usually a significant amount of time available to develop a good heuristic solution, and computation times up to several hours or even days are manageable. Time is just an order of magnitude question in evaluating corresponding algorithms. Quality is critical.

At the other extreme are *control* problems that must be solved frequently and involve decisions over a relatively short horizon. Examples are shop-floor scheduling, dispatching of automated guided vehicles in a manufacturing system, and transmitting multimedia data through a computer network. Heuristics are usually used for these problems even if exact algorithms are available because answers must be obtained in near real time. Algorithms must run in fractions of a second, but quality matters somewhat less.

Planning problems, such as aggregate production planning, shift scheduling and meeting timetabling, occupy an intermediate position in terms of frequency of solution and available solution time. An answer must be obtained in a short time for it to be useful, which often renders available exact procedures impractical. Still, quality is much more important than time in evaluating heuristic performance.

2.3. Problem life cycle

Problems studied by optimization researchers tend to follow a certain life cycle, and a different level of experimentation is appropriate at different stages. For many problems, consistently being able to construct a feasible solution is a challenge in itself, and the development of heuristics that are able to do this is a significant achievement. This was particularly true in the early days of computers when the amount of computational experimentation that could be performed was severely limited by available hardware and software. Early papers on heuristics, such as those of Fisher and Thompson (1963) on dispatching rules for job shop scheduling, King (1980) on group technology cell formation, Kilbridge and Wester (1966) on assembly line balancing and Lin and Kernighan (1973) on the Traveling Salesman Problem fall into this category. Presenting the logic behind the heuristic as well as one or two illustrative applications on small data sets was quite adequate. The contribution lies in developing a workable algorithm for a problem for which there was no practical algorithm before.

Once a body of algorithms for the problem has emerged, simply demonstrating that an algorithm is possible is no longer a research question. This is particularly true in the case of metaheuristics like tabu search and simulated annealing. It is possible to adapt these methods to a wide variety of problems relatively trivially, making the demonstration that such an algorithm is feasible an uninteresting question. At this point in the problem life cycle, a useful contribution must either produce an insight that was not gleaned from preceding approaches, or exhibit an algorithm that outperforms existing methods on some relevant performance measure. Experimentation needs to be much more comprehensive, exposing the proposed heuristic to a wide variety of instances to test its performance under different conditions and in comparison to other algorithms.

New models and problems are continually introduced from a variety of application domains as well as more abstract research. In contrast to problems like the Traveling Salesman Problem, Job Shop Scheduling, the Knapsack Problem and Assembly Line Balancing, which have been studied for decades, new problems such as scheduling of multimedia data through distributed computer networks are continually emerging for which developing a reliable algorithm to obtain near-optimal solutions is a challenge. The life cycle begins anew.

3. Design of computational experiments

Throughout this paper we follow the usual convention of defining a *problem* to be a generic form such as the Traveling Salesman Problem or the Job Shop Scheduling Problem and an *instance* to be a particular numerical case. Every computational experiment involves running

	Algorithm 1	Algorithm 2	···	Algorithm n
Instance 1				
Instance 2				
⋮				
Instance m				

Figure 1. Basic instances vs. algorithms design.

one or more algorithms on a chosen population of instances and measuring responses. We begin here with the easiest design issues of experimental *format* or *layout*—what cases to run—leaving for Sections 4 and 5 the more vexing questions of how to obtain instances and how to measure performance when the experiment involves heuristics.

3.1. The basics: Instances vs. algorithms

We can start thinking about virtually any computational experiment by visualizing it as a simple matrix like figure 1. Rows correspond to problem instances and columns to algorithms being tested. Each cell represents the running of a particular algorithm on a specific instance. The instances may be completely independent, but more commonly they are grouped by structural characteristics such as size. The algorithms may be truly different procedures, but they often include variations on a single idea.

In order to construct such a conceptual experimental design, one starts with a set of questions about the heuristics under study that need to be answered. These are almost always related to how different problem characteristics (e.g., problem size, number and nature of constraints), and algorithm components or parameters (e.g., stopping criteria, search neighborhoods and move selection) affect the performance of the heuristics being tested. In development experiments the instances may consist of a fairly fixed list of library test problems. Only the algorithm dimension requires careful design. In research investigations, far more thought needs to be given to what problem characteristics should be tested. One does not want to be misled by experimenting in only a small part of the problem domain. Discovering how problem characteristics affect performance should also be an important goal of the research.

Suppose, for example, that we are working on a genetic algorithms-based heuristic for designing a telecommunications, wastewater or distribution network. An instance consists of a graph of candidate links, each of which has a fixed cost if it is used at all, plus a variable or congestion cost applied to flow. Specified supplies and demands detail requirements at sources and sinks. Test instances may be characterized by any number of parameters: number of nodes, number of links, relative size of fixed and variable costs, distribution of source, sink and transshipment nodes, size and variability of supplies and demands, and many more.

Combinations of any or all such parameters would be arrayed along the instance dimension of an experiment to see how the subject algorithms perform on various types of input. But each combination is a choice of problem parameter values, not a specific instance. We

would normally wish to test several distinct instances or *replicates* sharing each parameter combination to avoid being misled by an aberrant case.

The genetic algorithms in our example operate by maintaining a "population" of solution vectors. At each step a randomly chosen pair of solutions is "crossed over" by cutting the solutions at one or more randomly chosen points and reassembling two "children" with parts taken alternately from the two "parents". If parents are chosen in a way that favors solutions with the best objective value, replacing current members of the population with one or both children tends to improve the overall quality of the population.

Completely different procedures could be included in the experiment as baselines, but the algorithms dimension of our computational experiment on genetic algorithms might consist of nothing more than combinations of alternatives for implementing the concept. How big should the population be? How are parents selected for crossover? How many places should solutions be cut? How many and which children should replace members of the current population? A complete implementation also requires a mechanism for maintaining diversity in the population, a rule for terminating the search, and much more.

There is an extra consideration when testing on randomized procedures like genetic algorithms. Even for a single instance and a single algorithm implementation, successive trials can produce quite different outcomes because of the impact of random choices along the way. Several *runs* will usually need to be made, with different random number seeds controlling the evolution of computation, to get a sense of the robustness of the procedure.

Often experimental factors in both the instance and the algorithm dimension are run at only two levels: "high" and "low". This at least allows the researcher to determine whether the factor makes a difference, but results obviously depend on what levels are chosen and how far apart they are. Also, it is only possible to determine whether an effect is linear or nonlinear if at least three levels are included.

In our genetic network design example, we could easily have 8 problem parameter factors, which yields $2^8 = 256$ combinations, even if we have only 2 levels per factor. Using 3 replicates sharing each parameter combination leads to $256 \cdot 3 = 768$ instances. Even three algorithm factors at 2 levels would produce $2^3 = 8$ variations on the algorithm dimension, and with a randomized procedure, we would run each instance-algorithm combination say 5 times with different random seeds. The result is $768 \cdot 8 \cdot 5 = 30,720$ runs. Clearly, the innocuous looking instance-algorithm conceptualization of figure 1 carries the potential for an explosively growing experiment size.

If the problem of interest is a design problem such as our network example, it is likely that the algorithms are quite time-consuming, and the execution of a large experiment with so many cases would be impractical. Something must be done to limit experiment size. On the other hand, for control problems such as short-term scheduling of jobs on a single machine, a heuristic must necessarily be very fast in order to be useful. Thousands of runs may take only a few minutes to complete.

3.2. Refining experimental designs

The approach to experimental design we have outlined above is an extremely simplistic one, which consists of identifying all possible problem and algorithm characteristics which might conceivably be of interest, running all combinations and inspecting the results to see

what we can learn. We believe laying out factors in this way can be a useful basis for initial thinking about the design of a computational experiment, but a great deal of refinement is usually necessary before it is time to actually make the runs.

Exploratory or *pilot* studies over a few well chosen instances can be invaluable in honing a final experimental design. Such preliminary tests often demonstrate that some of the factors initially thought important have minimal effect, or at least that there is a single best level that can be fixed in all main runs. They may also suggest how two or more initial factors can be collapsed into a single one without missing effects that really matter. Where two or more levels of a factor do seem to be required, pilot testing is the main tool for deciding how many and what levels to use in determining whether the factor has practical significance. Finally, pilot testing give us an idea of how much variability we can expect in outcomes, i.e., how much replication and repeat running of randomized algorithms will be necessary to obtain reliable results.

Pilot studies also offer the opportunity to engineer many details of the algorithm dimension of an experiment as factors and levels are being refined. Candidate stopping rules, for example, often emerge from ad hoc testing on an early list of instances. Preliminary tests may also show that, say, the size of a genetic algorithm population p should grow with instance size n. Rather than formal testing over all combinations p and n, the algorithm can simply be redesigned to use a varying population related parametrically to instance size (e.g., $p = 10\sqrt{n}$).

Of course there are dangers in tuning an algorithm to work well on a specific set of instances. Results may not generalize to the full problem population. One of the advantages of not testing over all instances from the start is that algorithm engineering at early stages will not unduly compromise this validity of final results.

Even when the formal testing stage is reached, an experiment need not be a single monolith. *Sequential* or *multiple* designs may be much more efficient. For example, a first round of experiments might include all problem and algorithm factors at two levels with little or no replication. Analysis of these first results may suggest that many of the problem parameters and algorithms can be discounted in a second, focused around employing heavy replication on a few algorithms and factors that have emerged as central to the goals of the investigation. Similarly, if early experiments show some algorithms are simply not competitive in certain parts of the problem domain, separate experiments can be conducted in different problem categories, with only the appropriate algorithms tested in each.

With the cost of computer experiments being low in comparison to those on physical and biological systems, most formal computational experiments, whether single or multiple, run all algorithms on all instances. In the terminology of experimental design, these are called *full factorial* designs.

There is a wide literature of *fractional factorial* designs in statistics which seek to assess the same effects without running all combinations (see for example, Montgomery (1991)). A carefully chosen subset of factors and levels balances observations to avoid confusing main effects. When no other technique is available to squeeze the size of an experiment within available resources, such fractional designs can be helpful. However, many experiments on heuristics require replication—not one but several observations in each cell. Fractional designs are not appropriate in such settings.

3.3. Blocking on instances

Most computational experiments, at least most research-type experiments, use problem factors like size and cost structure, rather than specific instances along the instance dimension of figure 1. For each combination of problem factors and each algorithm, one or more specific instances must be chosen for testing.

This poses a dilemma with regard to the *randomization* standard in statistical experimental design which posits that all factors not explicitly controlled in the experiment should be set randomly (a classic example is the sequence in which observations are taken). Strict application of this randomization principle would mean different instances should be randomly chosen for each algorithm.

Common sense argues for the alternative of *blocking*, i.e., running all algorithms against the same collection of instances. Sound theoretical support of this approach can be found in Lin and Rardin (1980), which demonstrates that differences among algorithms are more likely to be detected statistically if the same instances are solved by all. For these reasons, well designed computational experiments almost always block on instances. However, some adjustments must be made in any statistical analyses of the results to account for the effect of blocks (see Section 7.4).

3.4. Balancing time and quality

We have noted earlier in this paper that one of the unique features of experiments on heuristics is the need to address tradeoffs between time to obtain solutions and how near they are to optimal. A bit of creativity is often required to deal with such tradeoffs in experimental designs.

Consider, for example, an experiment comparing a straight-forward local search to a much more complicated tabu or simulated annealing algorithm. The latter maintain the neighborhood structure of the local search but allow long sequences of non-improving moves. Not surprisingly, they can obtain much higher quality solutions at the cost of a dramatically longer run times.

In such a circumstance the experiment can often be made fairer by allowing the simpler method to be run more than once. That is, the local search is repeated from different random starts until a fixed limit on run time or number of objective function evaluations is reached. The tabu or simulated annealing algorithm is then run to the same limit, and results compared.

The principle is that all algorithms are allowed to consume the same amount of computational resources, with distinctions being based on the quality of solutions obtained. This is often the most effective way to fairly compare dramatically different heuristic algorithms.

4. Sources of test instances

One of the greatest practical challenges in conducting any computational experiment is assembling the required test instances. No matter how carefully we structure the experimental design, it cannot be implemented without sufficient quantities of realistic test instances that have the size and variety to span all problem characteristics of interest.

4.1. Real world data sets

At least in development experiments, the best test instances are probably those taken from real application. A host of hidden patterns and relationships between the details of the instance are automatically represented. No parametric description of problem characteristics can go as far to assure that inferences drawn from experiments will hold up in the real application of interest.

Sadly, it is rarely possible to obtain more than a few real data sets for any computational experiment. Proprietary considerations lead organizations that have the data to be reluctant to make them public, although they can sometimes be convinced if information is sanitized by changing names and rescaling constants. Also, collecting a real data set is a time consuming and tedious effort that may take longer than experimenters can wait. (See Fargher and Smith (1994) for an interesting discussion of development testing of a complex heuristic using both artificial and real data.)

Especially in the context of research-oriented experiments, real world data also have severe methodological limitations. They necessarily represent systems now in place. Tests on concepts not yet implemented may require information that does not exist in any actual application. Similarly, real world data sets rarely span all problem characteristics of interest. If experimenters are to test the limits of viability of a heuristic, they need instances of every size and type.

4.2. Random variants of real data sets

An alternative, which we believe is too little used, expands the power of a few available real data sources by testing on random variations. The macro-structure of the actual application is preserved, but details are randomly changed to produce new instances.

In its most elementary form, this approach fixes structural characteristics of a known instance and varies some of the numerical constants. For example, testing of a vehicle routing algorithm might leave unchanged the locations of depots and customer sites. However, many new test instances could be obtained from a single topology by randomly varying customer demands within ranges observed in practice or artificially introducing transportation delays along some highway links.

Ovacik and Uzsoy (1997) employed a more extensive implementation of the same idea to develop a random problem generator for scheduling instances in a semiconductor testing facility. Extensive data was collected from the shop floor of an actual test facility. This data included lot sizes, due date characteristics and processing times from the actual machinery. After statistical distributions were fit to the lot sizes, the distributions were used to generate the lot processing times and due dates for an experiment on heuristics.

4.3. Published and online libraries

As mentioned above, problems pass through a life cycle from new ones for which there is no viable algorithm to mature and well studied topics with sometimes hundreds of alternative methods proposed and tested. Investigations done early in the life cycle often contain

relatively small instances used to support the main contribution of the paper—construction of an algorithm capable of generating at least feasible solutions where none could be had before.

As research in these areas continues, the problem sets used by pioneering researchers tend to coalesce into collections of classic benchmarks used by all investigators working on the same problem. In a number of cases, early benchmarks have also been supplemented by a variety of larger instances assembled by different researchers along the way.

With the advent of the Internet, this ad hoc collecting of benchmark problems has blossomed into a host of online libraries which are readily available to interested researchers. For example, Bixby and Reinelt (1990) offer a *TSPLIB* of Traveling Salesman Problem test instances on the Internet, and Demirkol, Mehta and Uzsoy (1998) do something similar for Job Shop Scheduling problems. Such libraries are an invaluable tool in heuristic testing. Still, they introduce subtle biases that need to be recognized.

- Although the earliest of these benchmarks were usually based on real applications, the lineage of more recent additions is often much less clear. There is no particular reason to believe some of the posted instances closely model any real world environment.
- Some of the test instances found in texts and pioneering papers are not intended to be representative of applications at all. They may serve merely to put an algorithm through all of its steps, or worse, they may be specifically designed to illustrate pathological behavior.
- Researchers who post test instances can be expected to favor those on which their algorithms have been successful. There is nothing unethical about this; the experiments most likely to justify formal documentation and publication are those that the researcher considers a success. Still, this may produce a hidden bias against alternative algorithms better adapted to different problem characteristics.
- Availability of standard libraries may draw too much effort to making algorithms perform well on the specific benchmark problems available. This blurs the distinction between research and development studies to the deteriment of both (Hooker, 1994, 1995). Research effort is better redirected to learning more about the various algorithms and how their performance is impacted by problem characteristics.

4.4. Randomly generated instances

When none of the other sources discussed above provide an adequate supply of test instances, or even when they do yield enough data sets but we wonder about hidden biases, the remaining alternative is pure random generation. Instances are synthesized entirely artificially, although their properties may be controlled by broad parameters and their general form inspired by applications.

Random generation is certainly the quickest and easiest way to obtain a rich supply of test instances, but is also the most controversial. Some critics rightly argue that testing on casually generated random instances may seriously mislead investigators. (We will deal with some of those pitfalls shortly.)

Harder to understand is a general hesitation among optimizers, raised in certainty of the mathematics culture, to credit tests on synthetic data at all. If the same sort of thinking were

applied, say in biological experimentation, medical science would still be in the Dark Ages. Investigations there that are ultimately aimed at developing drugs or insights for the human population almost always begin on fruit flies, mice or bacteria. This is not because results on these laboratory models provide the most accurate indication of how things will work with humans, but because tests on fruit flies, mice and bacteria are dramatically cheaper, quicker and freer of ethical limitations than experiments on human subjects. Just as important, long experience has shown that insights gleaned from tests on such laboratory models often hold up in later trials on humans. In short, the science advances more quickly and relatively little validity is lost. We see no reason why computational experiments should be any different.

The conveniences of randomly generated instances in computational testing of heuristics are many:

- Problem characteristics are explicitly under the researcher's control if the generator is properly designed to produce instances with specified parametric characteristics (e.g., size, density, variability, etc.). This allows a good random generator to produce an extremely diverse population of instances, encompassing parts of the problem space that may not be represented in available real data or library benchmarks.
- If a generator is properly documented, the characteristics of the instances it generates are completely known to all future users. This is in contrast to many library data sets, whose origins may be somewhat obscure.
- Once a suitable generator has been written, the supply of instances it can provide is unlimited. Each new random seed produces a new data set with the same parametric characteristics. This is particularly valuable in high-variance situations where researchers need heavy replication to understand the behavior of an algorithm.
- Instances produced by random generators are extremely portable. Only the code and the parametric input need to be known to recreate a possibly enormous data set.
- For some classic problems, generators have been developed that yield instances for which an optimal solution is known upon completion of the generation process. (See Section 5.1 below.) With a known optimum in hand, a researcher can evaluate precisely how close heuristics come to optimality on instances of virtually any size.

4.5. Pitfalls of random generation

The very freedom to produce data sets with arbitrary characteristics with random generators also raises a number of interesting and potentially disturbing concerns. Are the problems we have generated suitably difficult or representative, or is there some hidden structure that makes them especially easy (or hard) for specific algorithms? Given that a procedure performs well on the experimental data sets, how likely is it to perform well in another environment?

These issues need to be addressed systematically by every researcher in the design of random generators. Failure to do so may introduce confusion into the experimental results which can, in extreme cases, invalidate all the results of the study and cause its authors and subsequent researchers considerable wasted time.

One dilemma posed by the power to generate instances of any specified parametric characteristics is that we have to decide what parameter values to try. Often, corresponding

values found in real data will be unknown or undocumented. The only choice is extended pilot testing followed by an expanded experimental design where poorly understood characteristics are vaired systematically.

Another concern is that random generators do not always control what they seem to, which leads to faulty analysis of the effects of different problem parameters. A good example is the issue of generating precedence graphs, which are acyclic digraphs with arcs fixing what nodes must come before what others.

A standard descriptor of a precedence graph is the density of the graph, i.e., the fraction of all possible node pairs that actually are linked by an arc. At first sight, if we want to generate a precedence graph with a given density, we could proceed by simply assigning a probability to each node pair and creating an arc between nodes of that pair with the given probability. However, Hall and Posner (1996) point out that this approach ignores the transitivity of precedence relationships; if A precedes B, and B precedes C, then A implicitly precedes C. Hence, the density value suggested by the probability of having an arc between nodes of a given pair severely underestimates the true precedence density in the generated graph.

In our experience, the most subtle and insidious of random generation pitfalls arise from the very randomness of the instance creation process. We can illustrate with a simple example. Suppose we want to randomly generate instances of the n-point Traveling Salesman Problem (TSP), i.e., n by n symmetric matrices of point-to-point distances/times/costs $c_{i,j}$. Almost everyone's first idea of how to do the generation would be something like the following:

> *Fill the upper triangle of an n by n cost matrix with $c_{i,j}$ generated randomly* (*independently and uniformly*) *between 0 and 20. Then complete the instance by making $c_{j,i} = c_{i,j}$ in the lower triangle and setting $c_{i,i} = \infty$ along the diagonal.*

Think carefully about at the instances we would produce by such a generator. Each cell of the matrix is an independent realization of the *Uniform* (0, 20) distribution, which has mean $\mu = (20-0)/2 = 10$ and standard deviation $\sigma = (20-0)/\sqrt{12} \approx 5.774$. The length of any feasible tour will be the sum of n of these independent cell values. Thus, the mean tour length will grow linearly in n as μn, and the standard deviation of tour length will increase with $\sqrt{n}$ as $\sigma\sqrt{n}$.

Suppose we decide to test our heuristics on really big instances from this generator, say $n = 5000$ points. Tours will average $10(5000) = 50{,}000$ in length with standard deviation $5.774\sqrt{5000} \approx 408$. Using the central limit theorem, this means nearly every feasible solution will have length within $\pm 3(408)$ of 50,000. These are hardly typical TSP's! Almost any random guess will yield a good solution.

The difficulty here is too much independence. Constants in a randomly generated problem must have enough internal consistency, or correlation to present something like the same challenge to a heuristic that a real data set would produce. In the case of our TSP's, we would be much better off to generate costs out of some sense of spacial distance: randomly place points in space, and let the $c_{i,j}$ be the implied distance from point i to point j. Points need not be spread uniformly—we can introduce random clusters—and distance metrics need not be as simple as Euclidean. Still, the resulting $c_{i,j}$ will yield much more useful test

results because values in different cells of the matrix are correlated in something like the way they would be in real data.

The manner in which different problem parameters are correlated with each other (or uncorrelated) has been shown to have a critical effect on the performance of algorithms in many other cases. For example, Coffman et al. (1988) discuss a variety of settings where simple algorithms are asymptotically optimal with increasing size if the problem population is sufficiently random. In processor scheduling problems, job release times and due dates correlated so that a job which is released earlier is due earlier (agreeable release times and due dates) render a normally *NP*-Hard problem polynomially solvable (Lenstra, 1977). Hill and Reilly's (1996b) studies of two-dimensional knapsack problems show that instances where the objective function coefficients and constraint coefficients are correlated in certain ways are much harder, both for exact solution algorithms and for a heuristic, than those where these coefficients are independent. Furthermore, different types of correlation affect different algorithms in different ways.

It seems to us that given the many difficulties of obtaining and analyzing large numbers of problem instances based on any other source, randomly generated test problems will, *faute de mieux*, be with us for a long time to come. While there are risks, both obvious and subtle, they can be avoided to a great extent by thoughtful design of problem generators. Test instances cannot be an incidental concern addressed only at the last stage of research. Instead, investigators should expect to devote nearly as much time and energy to their experimental methods as they do to the algorithms themselves.

5. Performance measurement

The choice of performance measures for experiments on heuristics necessarily involves both solution quality and computation time. Time has actually received far more attention in the literature than quality, due to the need to reconcile results obtained from different computers using different hardware and software technology. Extensive discussions of factors to be considered while evaluating computation time and the need to describe them in detail can be found in Barr et al. (1995), Jackson et al. (1991), and Crowder et al. (1979). The arrival on the computing scene of different parallel computing architectures has added another dimension of complexity here, as discussed by Barr and Hickman (1993).

In this paper we have taken the position that highly accurate measurements of computation time are required chiefly in the development stage of heuristic development, when the basic algorithms have been designed and one is aiming at effective implementations for a specific setting. Running time is an order-of-magnitude question in most other experiments on heuristics. In addition, the issues in tracking CPU times differ little between exact and approximate algorithms.

5.1. Solution quality

The real challenge is evaluating solution quality. Except in control applications, where heuristics may still be used when orders-of-magnitude slower exact algorithms are available,

the problems to which heuristics are addressed are almost always hard, usually *NP*-Hard. Exponential algorithms like branch and bound may be available for tiny cases, but the instances on which we should be using heuristics are beyond any practical hope of computing an exact solution in the time available for experimentation. Even available bounds on optimal solution values are not likely to be very sharp. If they were, a satisfactory branch and bound algorithm would probably make heuristics unnecessary.

Timetabling problems, which abound in scheduling the offerings of classes in every kind of teaching institution or large conference, illustrate what heuristic researchers usually confront. Details vary, but most such problems are variants of Quadratic Assignment—choosing room/time slots for each event under quadratic costs that can only be assessed after pairs of events are scheduled. Timetabling problems are ideal targets for heuristic research because of their wide application, intractability to exact algorithms, and natural interchange neighborhoods for improving search. Still, there is no hope of computing a guaranteed optimum on instances of even moderate size, and there are no good lower bounds available for large instances because of the quadratic nature of the objective.

How are experimenters to evaluate heuristic solution quality on problems like these? The truth is that there is no satisfactory answer. Still, some are better than others. The next several subsections review the possibilities.

Exact solution of small instances: One of the most commonly used approaches is our least favorite (except in testing high speed control heuristics). Exact optimal solutions to a number of small instances are obtained, usually at the cost of high computation times, and the solutions from heuristics are compared. Then, the heuristic is applied to the dramatically larger instances of practical interest, and the performance measured on exact-solvable cases is assumed to carry over. Put simply, we look for results where they are easy to get, and hope they turn out to be representative.

We think there are many good reasons to doubt the underlying premise that heuristic behavior on small instances extends to more realistic cases. First, asymptotically optimal heuristics are known for some problems—the larger the instance becomes, the better the chances of the heuristic producing a good solution. An extreme example is the Quadratic Assignment Problem, which is the prototypical hard combinatorial problem at small and medium size. Burkard and Fincke (1983) prove it becomes easy for every heuristic as the size approaches infinity.

For small instances of any combinatorial problem, there are likely to be a limited number of feasible solutions, which can produce erratic heuristic performance. A tabu search or a genetic algorithm is able to search a large fraction of the solution space in a short period of time, making it very likely to find a high quality solution. Results are misleadingly good. On the other hand, one or two wrong decisions by a constructive, one-pass heuristic may result in a solution differing dramatically from the optimum of a small case. Outcomes are atypically bad.

All these large and small case phenomena suggest that if algorithms are evaluated on the basis of exactly-solvable small instances, researchers may be seriously misled. Effort spent developing and testing the exact method for small examples would be more productively used to find better heuristics.

Bounds on optimal values: To an investigator schooled in discrete optimization, bounds on the optimal solution value are a natural retreat from exact optimal values. We compute a bound on the optimal value and compare with the solution produce by the heuristic.

The advantage of this approach is that it gives us a provable upper bound on the deviation from optimality of the heuristic solution. Still, there are obvious disadvantages. If the bound is loose, it is not clear how much of any measured deviation from optimal is due to the poor performance of the heuristic and how much to the inadequacy of the bound. On the other hand, computing tight bounds is often an intellectually complex and computationally intense (sometimes even *NP*-hard) exercise in itself. After all, if we could compute sharp bounds easily, we could probably use branch and bound methods in the first place. As a result, a considerable part of the investigators time and effort may be diverted to implementing bound computations, all to more accurately assess heuristic performance.

Clearly, it is also possible to combine the last two approaches—compute optimal solutions for small instances, compare these to the bounds, and assume the measured bound error carries over to larger cases. This approach has the same disadvantages as generalizing heuristic performance from results on small instances. Gaps between bound and optimum may disappear as the number of feasible solutions explodes with instance size, or they may become worse.

Built in optimal solutions: A number of researchers have shown how instances of classic combinatorial problems can be randomly generated so that an optimal solution is known upon completion of the generation process. These include Pilcher and Rardin (1992), Arthur and Frendewey (1988), and Moscato and Norman (1998) for Traveling Salesman problems; Krishnamurthy (1987) for Graph Partitioning problems; Khoury, Pardalos and Du (1993) for Steiner Tree problems; and Sanchis (1994, 1995) for Vertex Packing, Maximum Clique and other graph problems. With a known optimum in hand, a researcher can evaluate precisely how close heuristics come to optimal on instances of arbitrary size.

Obviously, randomly generated instances with known optimal solutions must contain hidden structure that assures optimality of the reported solution. For example, the Pilcher and Rardin generators use polyhedral combinatorics, constructing a solution that is linear-programming-optimal over the original model plus a collection of randomly selected extra linear constraints.

To see the idea, suppose the problem of interest can be written as the integer linear program

$$\begin{aligned} \min \quad & \mathbf{cx} \\ \text{s.t.} \quad & \mathbf{Ax} = \mathrm{b} \\ & \mathbf{x} \geq \mathbf{0} \text{ and integer} \end{aligned}$$

and that a large family of other valid inequalities

$$\mathbf{Gx} \geq \mathbf{h}$$

are known to hold for every integer-feasible solution. Such families have been documented for many classic combinatorial problems, e.g., Grötschel and Padberg (1979, 1985) for the Symmetric Traveling Salesman Problem.

We construct an instance with a built in optimum as follows:

1. Randomly generate constraint coefficients **A** and **b**, along with an integer feasible solution $\mathbf{x}^*$
2. Randomly select a sample $\bar{\mathbf{G}}\mathbf{x} \geq \bar{\mathbf{h}}$ of the valid inequalities, being careful to include only ones satisfied as equality at $\mathbf{x}^*$
3. Randomly generate nonnegative dual multipliers $\mathbf{u} \geq \mathbf{0}$ on the main constraints, $\mathbf{v} \geq \mathbf{0}$ on the sampled valid inequalities, and $\mathbf{w} \geq \mathbf{0}$ on the primal variables, with $w_j = 0$ for all j with $x_j^* > 0$
4. Compute cost vector $\mathbf{c} = \mathbf{uA} + \mathbf{v}\bar{\mathbf{G}} + \mathbf{w}$
5. Discard the duals and valid inequalities before returning instance (**A, b, c**) with provable optimal solution $\mathbf{x}^*$

The constructed integer solution is optimal because it is optimal in the linear-programming relaxation with the main constraints and valid inequalities. However, there is no particular reason to believe it will be easy for heuristics. Outputs have been shown to be valid for evaluating non-polyhedral heuristics in Pilcher and Rardin (1987), and the full population of obtainable instances can be *NP*-Hard (Rardin, Tovey and Pilcher, 1993; Sanchis, 1990).

What limits wider application of these approaches to building in optimal solutions is their deep dependence on problem-specific information. For example, the polyhedral approaches outlined above depend on availability of both a rich class of known valid inequalities for the problem and a procedure for sampling from among those that are active at a particular solution. Both might be within reach for many classic combinatorial problems, but it is unrealistic to expect such techniques to extend to more applied models.

Statistical estimation of optimal values: Combinatorial optimization problems usually have an enormous number of feasible solutions. The idea of statistical estimation techniques for optimal values (Klein, 1975; Dannenbring, 1977; Golden, 1977, 1978; Golden and Alt, 1979; Los and Lardinois, 1982; Derigs, 1985) is to use a sample of solutions to predict where the true optimum may lie.

Some research attempts to work with assumed distributions of the full set of feasible solution values (e.g., Boender et al., 1982), but the foundation for the most popular estimation methods is a classic result due to Fisher and Tippett (1928) about the distribution of least values:

> *The least of m random variables with a common distribution on real numbers greater than or equal to a is, under mild assumptions, asymptotically distributed* (*as* $m \to \infty$) *Weibull with cummulative distribution function* $F(z) = 1 - \exp(-[(z-a)/b]^c)$

Here, the distribution lower limit *a* is the location parameter of the *Weibull* distribution, *b* is the scale parameter, and *c* characterizes its shape. Quite a large *m* is required for the result to hold with any precision.

Optimal solution prediction methods (for minimization problems) exploit this result by thinking of objective function values for feasible solutions to an optimization model as

points from an objective-value distribution. Assuming we can equate the continuous random variables of the Fisher-Tippett result with this many-valued, but discrete distribution, the minimum of any large collection of m such solution values should be approximately Weibull. If we take n independent samples of this sort from the same instance, they can be combined to estimate the parameters of the asymptotic distribution. In particular, they can approximate the location parameter a, which is the optimal solution value of the instance we need to evaluate heuristic performance.

There are many alternative ways to estimate the Weibull parameters (see Zanakis, 1977, 1979; Zanakis and Mann, 1982), but satisfactory results have been obtained with a very simple one. Let $z_1, z_2, \ldots, z_n$ be the n independent group minima, and sort them so that

$$z_{[1]} \leq z_{[2]} \leq \cdots \leq z_{[n]}$$

Then the location parameter or optimal value a can be estimated as

$$\hat{a} = \frac{z_{[1]}z_{[n]} - (z_{[2]})^2}{z_{[1]} + z_{[n]} - 2z_{[2]}} \tag{1}$$

A reliable estimate of the true optimum would be valuable in itself, but Golden and Alt (1979) introduce more certainty by computing a lower confidence limit z_ℓ which should be less than or equal to the optimal value z^* with high probability. Specifically, they estimate Weibull scale parameter b from $\hat{a}$ as

$$\hat{b} = z_{[\lfloor 0.63n+1 \rfloor]} - \hat{a} \tag{2}$$

and then compute

$$z_\ell = \hat{a} - \hat{b} \tag{3}$$

In theory, this interval should cover the optimal value z^* with probability approximately $(1 - e^{-n})$.

A common way of obtaining the sample minima z_i required in such estimates is to select n random initial solutions to the combinatorial problem under consideration and pursue each to a local optimum with some naive improving search. These n local minima, each of which is the best of many solutions in its neighborhood, are then used as the z_i. Random starts keep the z_i independent.

Many other implementations are possible. Ovacik, Rajagopalan and Uzsoy (2000) use z_i that are minima of long sequences of solutions visited by a simulated annealing heuristic. Hypothesis tests establish that these sample points are approximately independent. Ghashghai and Rardin (1998) follow an even simpler approach. The thousands of random solutions evaluated as part of their genetic algorithms search are divided into large groups, and group minima used as the z_i.

We believe these estimation techniques have considerable promise and warrant much more research. The effort required to compute optimal value estimates and confidence limits is relatively modest, and the techniques offer real hope of obtaining reliable information on optimal solution values that is independent of problem domain.

Still, the few tests that have been published to date show rather mixed results. For example Derig's (1985) experiments on published Traveling Salesman Problems gave good estimates $\hat{a}$ and confidence intervals that always covered the true optimum. However, tests in the same study on Quadratic Assignment instances were somewhat less satisfactory. Our sample experience in Section 6.7 is similar, showing both good and bad outcomes. Estimation must be viewed as promising work in progress.

Best known solution: When other methods cannot be applied, or as a supplement to them, almost every computational experiment on heuristics compares performance to the best known feasible solution for each test instance. Sometimes, as in the case of published benchmark libraries, the best known solution comes from other researcher's work. Many may have tried to solve the same instance and kept track of their best results. When this is the case, it is not unreasonable to assume the best known solution is optimal or nearly so.

Another source of best known solutions is long runs. A local search heuristic might be applied many times from different starts to obtain a good approximation to the optimal value, or a continuing method like Tabu Search might be run long beyond the stopping point that would be implemented in main experiments. Very large computation times can be justified if they are needed only once on each instance, and long runs can usually be obtained without many modifications in the algorithms of interest in the research.

A final way to find a best known solution is to record the best found by any of the algorithms in the current experiment. When the design solves the same instances by many procedures, especially when the tested methods follow fundamentally different strategies, these best solution value encountered by any algorithm may also be close to optimal. Still, we should be skeptical. If the instance was solved only a few times by substantially the same method, there is real reason to doubt whether we have any real idea how good an answer can be obtained.

5.2. Adjusted ratios

Even if a reasonable approximation to the optimal solution values of test instances can be obtained, the results of a computational study can often be influenced by exactly how heuristic and optimal solutions are converted into a quality measure. Usually, we wish to form the ratio of heuristic to optimal solution values, or equivalently, the fraction error obtained by dividing the difference between optimal and heuristic by the optimum. However, Zemel (1981) shows that details of the computation can change the result.

In many cases, the confusion has to do with adding a constant to both the numerator and denominator of the ratio. For example, adding a constant of 100 to every edge cost of a Traveling Salesman Problem instance on n cities has no effect on the ranking of feasible solutions; every solution value will be increased by the same $100n$. Still, the change adds $100n$ to both the numerator and the denominator of the performance ratio, which makes the two more alike and diminishes the measured error.

The cynical will see that one can achieve any given level of measured precision by simply adding a sufficiently large constant to all test instances objectives. Often the confusion arises from much more innocent thinking.

Consider the problem of scheduling n jobs on a single machine to minimize total completion time subject to job arrivals over time. Each job i has a processing time p_i and becomes available for processing at release timer r_i. Then for any job i, completion time is given by

$$C_i = r_i + p_i + W_i,$$

where W_i denotes the time the job spends waiting in the queue before it begins processing.

Notice that computed total completion time $\sum_i C_i$ for any job sequence includes constant $\sum_i (r_i + p_i)$. We will measure a considerably smaller percent error if we form ratios of this inflated $\sum_i C_i$ as opposed to the truly controllable $\sum_i W_i$.

Even more difficult issues arise when the objective function may have a negative value. For example, Ovacik and Uzsoy (1997) evaluated heuristics for scheduling a semiconductor testing facility in terms of maximum lateness

$$L_{\max} = \max_i \{C_i - d_i\}$$

where C_i is the completion time and d_i the due date of job i.

If due dates are sufficiently loose, $L_{\max}$ can easily be negative, and heuristic-to-optimal ratios make no sense. This anomaly can be remedied by adding a sufficiently large constant (e.g., $\min\{d_i\}$) to all $L_{\max}$ values. But then the size of the added constant changes the performance error measured.

Each of these ratio formation difficulties must be dealt with the in context of the particular problem being studied. An obvious rule of thumb, however, is to be conservative: form the ratios that will show the biggest percent error.

6. One machine scheduling case

To make some of the issues treated so far more concrete, as well as to provide a setting for illustrating analysis methods to come, we now include a brief case example. The problem is the scheduling of a single machine to minimize maximum lateness (completion time minus due date) for all jobs. Jobs may be taken up in any order, but sequence-dependent setup times add to processing times according to which job comes immediately before.

Like so many other combinatorial problems, most of what we can learn about the problem must derive from computational experiments. Still, the single processor maximum lateness problem without setups has been extensively examined, and earliest due date sequence produces an optimal schedule (Pinedo, 1995). Uzsoy, Lee and Martin-Vega (1992) investigate many theoretical aspects of the extension with sequence dependent setups, noting that the problem in *NP*-hard in the strong sense. They also exhibit a one-pass heuristic for the case where setup times do not exceed process times and prove it has worst case performance $3L^* + 2d_{\max}$ where L^* is the optimal lateness when setups are taken as zero, and $d_{\max}$ is the greatest due date.

Our case will deal with instances where jobs to be processed are organized into product groups. Setup times are applicable only upon changeovers between product groups. Recent work by Baker (1997), Baker and Magazine (1997), and Hariri and Potts (1997) has proposed and tested new heuristic algorithms for this setting.

Table 1. Algorithms tested in case.

Identifier	Strategy
EDD plus	Earliest due date followed by local search
Multistart	EDD Plus followed by repeated local search from random starts
Tabu 6	Tabu search prohibiting change of a swapped job for 6 moves
Tabu 12	Tabu search prohibiting change of a swapped job for 12 moves
Tabu 24	Tabu search prohibiting change of a swapped job for 24 moves

6.1. Heuristic algorithms

We began our thinking about algorithms with three possibilities:

- A one-pass, earliest due date (EDD) heuristic which is optimal when there are no setups.
- A local search starting from the EDD solution and making the best swap of two adjacent jobs in the sequence until no such move improves the solution value.
- A tabu search starting from the EDD solution and paralleling the local search in adopting the best available swap of adjacent jobs in the sequence. However, chosen moves need not be improving, and the most recently swapped jobs are excluded from consideration for a fixed number of moves to prevent immediate backtracking unless the forbidden move would produce the best solution found so far.

This initial list of three strategies evolved into the five algorithms of Table 1 through a combination of adding detail and pilot testing. First, we decided from pilot results like figure 2 to omit the pure EDD method in main testing. Its performance was much worse than all

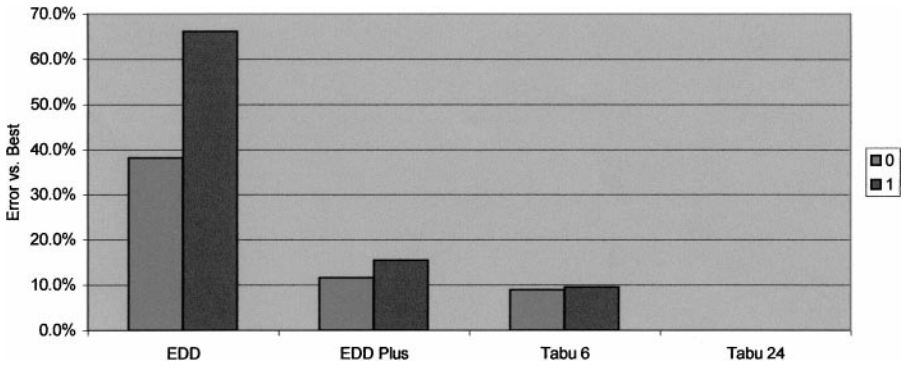

Figure 2. Pilot results on EDD.

competitors, and its erratic behavior confused analysis of problem parameter effects. For example, bars in figure 2 compare average performance (percent error versus the best known solution) on instances generated under different setup time assumptions 0 and 1. Only EDD showed much impact.

The EDD followed by local search alternative (EDD Plus) posed a different issue. Since it runs so quickly, it is plausible (see Section 3.4) that repeated use of the search from random starts would do better than the tabu methods if run for the same amount of time. We added the Multistart algorithm of Table 1 to test this idea. Local searches were restarted from random starts until the move limit of tabu alternatives was exhausted.

The tabu strategy has many specific implementations depending on the stopping rule employed and the number of moves a job stays on the tabu list (is forbidden to swap). Pilot testing evolved a stopping rule of terminating after 60 times the number of jobs moves with no improvement in the best known solution which seemed to work well in all cases. The effect of the number of moves a job remains on the tabu list was less clear in preliminary experimentation. Thus, three alternatives were formally tested, yielding the three tabu variations of Table 1.

6.2. Instance generation

Although some randomly generated instances of our problem could be obtained from other studies, there are essentially no reference benchmarks available. Thus we developed our own random instance generator for this study.

The list of controlled parameters in the generator is detailed in Table 2. Generation begins by choosing the process time for each of the n jobs uniformly over the interval $[p_\ell, p_u]$ and then picking setup times uniformly between multiples s_ℓ and s_u of process time.

To avoid the tendency to have too much independence in data sets, we imposed two structuring elements on setups. First, setup option δ changes the process time base from which individual setups are chosen. With $\delta = 0$ that base is the same midpoint of the process time range for all jobs; individual job setup times are independent of corresponding process times. With $\delta = 1$ the base is the actual process time for the job, which should introduce a correlation.

Table 2. Parameters controlled by the random generator.

	Parameter	Levels
n	The number of jobs	Fixed = 80
p_l, p_u	Lower and upper limits on process times	Fixed = 1 and 500
s_l, s_u	Lower and upper fractions of process times for setup times	Fixed = .25 and .75
δ	Setup option	Fixed = 1
g	Setup group fraction	Levels 0.1, 0.3 and 0.9
τ	Tardiness factor	Levels 0.3 and 0.6
w	Due date spread parameter	Levels 0.5 and 2.5

More important are the setup groups controlled by group fraction g. Many applications have product groups that require the same machine setup, so that no changeover time is needed when one member of a group is followed on the machine by another member of the same group. Furthermore, it can be shown (Monma and Potts, 1989) that the optimal schedule subsequence for members of any such group is EDD, which links grouping to our selection of algorithms. Group fraction g is the number of such groups generated as a fraction of the total number of jobs, so that $g = 1$ corresponds to no grouping and $g = 0$ to no setups. For $0 < g < 1$, jobs are randomly assigned to the specified number of groups and setup times set $= 0$ between members of the same group.

The other required data element for instances of our problem is due dates. They could be chosen completely randomly, but this would be another case of too much independence and leave due date tightness beyond our control. Proceeding instead on the assumption that due dates should be scaled to the time work can actually be finished, we begin their generation by computing a typical *makespan* or final completion time as

$$(\textit{makespan}) \approx (\textit{total processing time}) + \sqrt{g}(\textit{total average setuptime})$$

Here setup times are averaged over all possible predecessors and a $\sqrt{g}$ reduction (suggested by preliminary testing) is applied to account for the nonlinear saving realized with setup groups. Then actual due dates are randomly chosen between $(1-\tau)(1-w/2)$ and $(1-\tau)(1+w/2)$ times the estimated makespan. Varying τ controls the tightness of due dates, and changing w scales their variance.

6.3. *Response variable*

As in most investigations of heuristics, our main interest in this experiment is how close the algorithms come to producing an optimal schedule (percent error versus the optimal solution value). Unfortunately, no satisfactory lower bounds are available for the problem being investigated, and the random generator does not produce guaranteed optima.

This left us with two reasonable options from the list in Section 5.1 for approximating the optimal solution value: statistical estimation and using the best value encountered by any algorithm. We replied most heavily on the best known solution, but estimators (1) were computed for comparison from local minima of the Multistart algorithm alternative.

The lateness measure of our study also illustrates the difficulty discussed in Section 5.2 of expressing solution quality ratios, or equivalently percent errors, when an objective value can be negative. We dealt with this concern by adding the maximum due date to lateness values in both numerator and denominator of ratios. Since no job can ever be more early that the magnitude of its due date, this eliminates negative values. Still, resulting rations are skewed toward 1.000 by the transformation, thus understating the percent error of heuristic solutions.

6.4. *Picking levels of problem parameters*

We chose to try two problem sizes in our illustrative experiment, with numbers of jobs $n = 80$ and $n = 40$ respectively. Also, we wanted significant variability in process times,

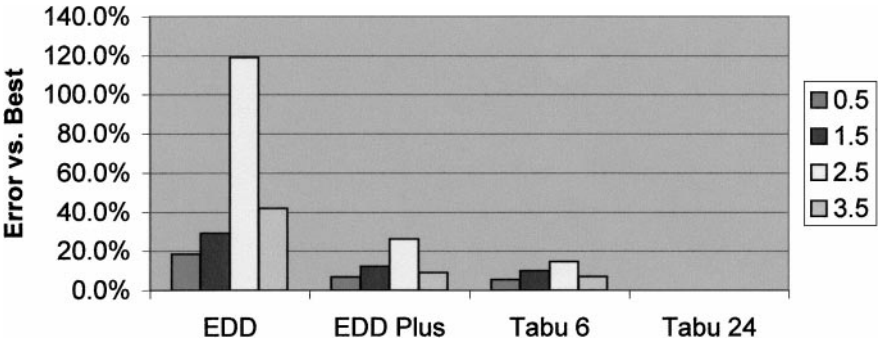

Figure 3. Pilot study of spread factor w.

so their range was fixed from $p_\ell = 1$ to $p_u = 500$. Experience with real applications in Uzsoy et al. (1992) suggested setup times between $s_\ell = .25$ and $s_u = .75$ of process.

The impacts of other parameters in Table 2 were less predictable, particularly the numerical ones like tardiness and spread which have a wide array of possible values. This is a circumstance where pilot studies are appropriate to help choose factor levels. EDD, EDD-Plus and two variants of tabu were tested on 5 replicates each at $n = 40$ jobs, setup option $\delta = 0$ and 1, group fraction $g = 0.1$ and 0.9, tardiness factor $\tau = 0.6$, and spread factor $w = 0.5, 1.5, 2.5, 3.5$.

Figure 2 shows average results for algorithms versus setup option δ. After deciding to delete the EDD procedure from later experiments, setup option could be fixed at $\delta = 1$.

Figure 3 shows similar results for due date spread factor w. Results show a clear effect, especially for the EDD algorithm, but it dissipates after $w = 2.5$. Thus, $w = 0.5$ and $w = 2.5$ were chosen as the two levels to be used in the main experiment. Similar pilot testing and experience in Ovacik et al. (1994) settled the levels of the tardiness factor at $\tau = 0.3$ and $\tau = 0.6$.

The impact of setup groups was a major interest of the study and its effect was expected to be nonlinear, but pilot results were inconclusive. Thus we chose to try more than two levels in the main experiment. Values 0.1, 0.3 and 0.9 were selected to produce equal spacing on a logarithmic scale and span the possible range from 0.0 to 1.0.

6.5. Blocking and replication

Consistent with our discussion of blocking in Section 3.3, we solved each instance by all five algorithms of the main experiment. None of the algorithms make random choices, but we were still confronted with a decision on how many instances to run at each combination of problem factor levels. At least three such replicates are needed if results are to give any idea of algorithm robustness, since one tells us nothing about reliability and two gives us no clue which is more typical. Since run times were modest, on the order of one minute or less, we chose to test ten instances at each combination of problem characteristics.

Table 3. Average percent errors for main experiment.

Number jobs	Group fraction	Tardiness factor	Spread factor	EDD plus	Multistart	Tabu 6	Tabu 12	Tabu 24
40	0.1	0.3	0.5	20.7	0.0	16.6	11.4	16.2
			2.5	11.5	11.0	6.8	0.5	7.6
		0.6	0.5	18.4	0.0	15.6	10.7	14.5
			2.5	11.2	3.9	7.1	1.7	6.4
	0.3	0.3	0.5	16.5	0.1	12.9	7.8	15.5
			2.5	34.4	34.4	13.9	0.0	18.4
		0.6	0.5	24.2	0.0	20.1	14.7	20.6
			2.5	16.7	16.7	11.7	0.0	14.4
	0.9	0.3	0.5	10.5	7.3	7.0	0.6	7.4
			2.5	1.9	1.9	0.3	0.0	0.0
		0.6	0.5	10.3	1.3	6.9	1.1	7.5
			2.5	53.3	53.3	5.9	2.7	46.9
80	0.1	0.3	0.5	27.2	0.0	25.4	23.8	17.4
			2.5	17.7	17.7	15.5	11.5	0.0
		0.6	0.5	27.0	0.0	26.1	22.7	18.5
			2.5	14.9	2.0	12.6	10.9	3.6
	0.3	0.3	0.5	23.0	0.0	20.8	18.3	12.0
			2.5	118.7	118.7	92.1	26.0	0.0
		0.6	0.5	26.4	0.0	24.7	22.0	17.7
			2.5	14.3	14.3	12.2	8.3	0.0
	0.9	0.3	0.5	8.4	8.4	6.0	3.0	0.1
			2.5	0.0	0.0	0.0	0.0	0.0
		0.6	0.5	10.2	0.0	9.3	6.9	4.0
			2.5	36.3	36.3	23.2	11.3	3.8
Overall average				23.1	13.6	16.4	9.0	10.5
Overall worst				909.0	909.0	658.2	204.7	290.8
Overall best				0.0	0.0	0.0	0.0	0.0

6.6. Main experiment results

Tables 3 and 4 summarize results for our main case experiment. In all

$$(3g's)(2\tau's)(2w's)(10 \text{ replicates}) = 120 \text{ instances}$$

were solved by each of the 5 algorithms.

Each cell of Table 3 is an average over the ten instances tried. Table 4 provides more detail on the effects of each problem factor versus algorithms. Average errors are

Table 4. Percent errors for problem factors versus algorithms.

		EDD plus	Multistart	Tabu 6	Tabu 12	Tabu 24	Overall
Number jobs							
40	Average	19.1	10.8	10.4	4.3	14.6	11.8
	Worst	290.8	290.8	30.6	18.2	290.8	290.8
	Best	0.0	0.0	0.0	0.0	0.0	0.0
80	Average	27.0	16.5	22.3	13.7	6.4	17.2
	Worst	909.0	909.0	658.2	204.7	34.7	909.0
	Best	0.0	0.0	0.0	0.0	0.0	0.0
Group fraction							
0.1	Average	18.6	4.3	15.7	11.6	10.5	12.2
	Worst	34.2	34.2	31.3	28.0	22.4	34.2
	Best	0.0	0.0	0.0	0.0	0.0	0.0
0.3	Average	34.3	23.0	26.1	12.1	12.3	21.6
	Worst	909.0	909.0	658.2	204.7	65.2	909.0
	Best	0.0	0.0	0.0	0.0	0.0	0.0
0.9	Average	16.4	13.6	7.3	3.2	8.7	9.8
	Worst	290.8	290.8	112.4	72.9	290.8	290.8
	Best	0.0	0.0	0.0	0.0	0.0	0.0
Tardiness factor							
0.3	Average	24.2	16.6	18.1	8.6	7.9	15.1
	Worst	909.0	909.0	658.2	204.7	65.2	909.0
	Best	0.0	0.0	0.0	0.0	0.0	0.0
0.6	Average	21.9	10.7	14.6	9.4	13.2	14.0
	Worst	290.8	290.8	112.4	72.9	290.8	290.8
	Best	0.0	0.0	0.0	0.0	0.0	0.0
Spread factor							
0.5	Average	18.6	1.4	16.0	11.9	12.6	12.1
	Worst	31.5	11.4	31.1	28.0	26.0	31.5
	Best	5.1	0.0	1.0	0.0	0.0	0.0
2.5	Average	27.6	25.9	16.8	6.1	8.4	16.9
	Worst	909.0	909.0	658.2	204.7	290.8	909.8
	Best	0.0	0.0	0.0	0.0	0.0	0.0

provided for each level of each problem factor, both overall and by algorithm. The single worst and best performance observed on any instance is also given for each combination. Most of the rest of this paper will focus on analyzing and presenting such results.

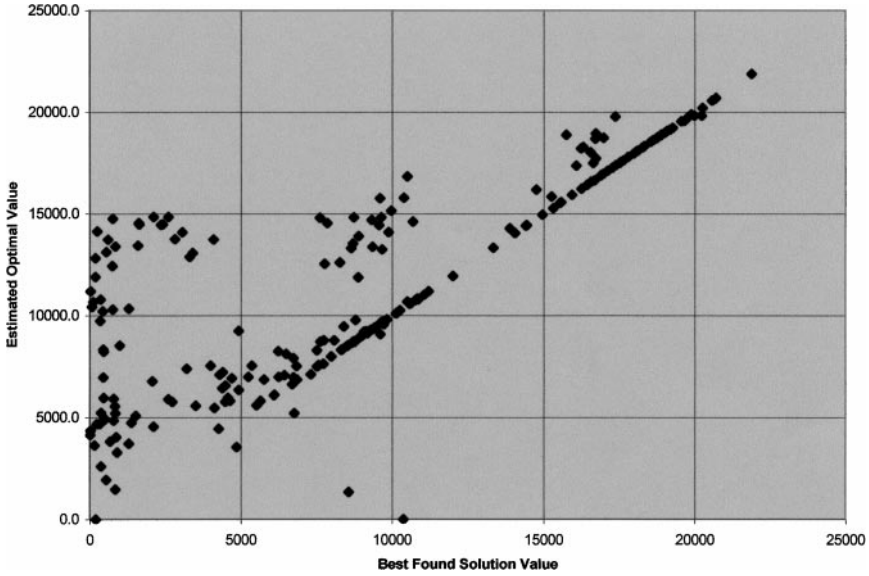

Figure 4. Comparison of estimated and best found solution values.

6.7. Estimated optimal values

The repeated local minima produced by the Multistart algorithm in our experiment afford an opportunity to test Golden-Alt optimal solution value estimators of equation (1). Figure 4 arrays results for the 240 instances, comparing best found and estimated optimal values.

The large number of points near the diagonal in figure 4 demonstrates promise. For these instances, the estimate was very close to the best solution found. Of course, the best solution found is only an upper bound on a true optimum.

Unfortunately, many of the points in figure 4 fall above the diagonal, indicating instances for which the estimated optimal value was not even as small as the best known solution, which may itself be higher than the true optimal value. These results certainly raise concerns about the hazards of estimating optimal values.

We conjecture that a part of the difficulties may be a consequence of our problem having a minimax rather than a minisum objective. All prior tests of the estimators with which we are familiar dealt with the minisum form. Minimax problems may be more problematic because it is common that large numbers of solutions have exactly the same objective value. Thus underlying assumptions that the distribution of optimal values is essentially continuous may not hold. (See Garfinkel and Gilbert (1978) for some evidence.)

7. Analysis

After one or more computational experiments have been conducted, the typical researcher is confronted with a daunting array of numbers—performance measures computed while solving a usually large number of instances by a variety of algorithms. Now what?

We think it is useful to distinguish two stages of the processing that must follow. *Analysis*, the stage we consider in this section, is highly exploratory. Guesses and false starts are fine; pages are unlimited. We seek only to discover as much as we can about the stories the numbers have to tell while taking precautions to be sure we are not deceived by apparent phenomena that may not prove real.

The *presentation* stage, which is the topic of Section 8, aims to illuminate and assess the importance of major results in a concise way suitable for reporting and publication. Details matter; balance and completeness are essential.

7.1. Randomness in experiments on heuristics

Statistics, as opposed to statistical methods, are nothing more than aggregate numbers that summarize the central tendencies and variability of sets of numbers. If, for example, we are testing a variety of (nonrandom) heuristics on a fixed set of benchmark instances, we might calculate the average and standard deviation of the resulting measures of solution quality for each algorithm to help compare performance.

Notice that there is no error in these statistics. The population of instances is fixed, and we have data for all its members. There might be some measurement error in solution times, which can vary with the competing load on the computer, but for the solution quality measures central to research on heuristics, results would be unchanged if we ran the algorithms again on the same instances. None of the parts of statistical methods addressed to sampling uncertainty are either needed or appropriate to analyze our results.

When does uncertainty have to be confronted in analyzing computational experiments: whenever results can be viewed as a random sample from a much larger, often essentially infinite population. Without being able to test the whole population, we can benefit from statistical methods to get an idea how much the results we have can be trusted.

Such randomness definitely arises (i) when instances are produced by a random generator, and/or (ii) when algorithms make random decisions as they search. In our case study, for example, results for the 10 instances tested at each level of the problem factors constitute a random sample of the population of outcomes that could arise as we change the random number seeds. Similarly, any collection of measured performances on a given instance by genetic or simulated annealing algorithms, which make random decisions at each step, are only a sample of the infinite population of outcomes for each algorithm-instance combination.

The notion of a sample can sometimes be stretched to encompass data sets taken from real application. If the instances were collected in a sufficiently random way, we might be willing to assume they constitute a random sample of all possible inputs. Still, statistical methods that account for randomness have to be applied cautiously in such cases because any biases in the sampling are likely to be reflected in the results.

7.2. Statistical vs. practical significance

When computational results can be taken as a random sample from a larger population, any analysis must be alert to the possibility that apparent effects in the results are nothing more than statistical accidents. Could another sample or a bigger one yield different conclusions?

This is the issue of *statistical significance*—determining whether an observed affect is likely to be due to sampling error. Most authors who have written recently on computational experimentation (Hooker, 1994 and 1995; Golden and Stewart, 1985; Barr et al., 1995; McGeoch, 1996) advocate more use of formal tests of statistical significance as a way of introducing more scientific precision into empirical investigations of algorithms.

We certainly agree that a part of every analysis of data drawn from a sample should be some assessment of whether apparent effects could be due to sampling error. Still, we believe it is critical to keep in mind the limitations of formal tests of statistical significance.

First, every such test is built on a series of assumptions about the underlying distributions and the sampling process. As we will illustrate in our discussion of some of the more familiar methods below, such assumptions can often be questioned when dealing with experiments on heuristics. Certainly, formal statistics cannot yield much useful edification unless they are done carefully and correctly. Researchers, especially those like us who are much more expert in optimization than in statistical methods, are well advised to seek guidance from colleagues with more sophisticated knowledge to avoid novice errors.

A more fundamental concern is that statistical significance is too often confused with *practical significance*—whether an observed effect is large enough to matter. The tiniest of effects can be statistically significant if enough points are sampled to make the likely random error even tinier. That does not mean the effect is important, or even worth reporting in a formal presentation of experimental results. A practically significant effect should not be claimed unless it is statistically significant, because it may be a random accident, but a demonstration of statistical significance does not prove practical importance. Analysts must take care not to let formal statistical tests dominate their investigations or substitute for their judgment and insight about the problems and algorithms addressed.

7.3. Two-way tables of sample means

Whether or not randomness is present in experimental results, almost any analysis of heuristic tests will begin by examining displays of average results like Table 3. Just by looking at overall results we can draw a preliminary conclusion that tabu algorithms usually produce better quality results than the others tested, although all methods do very poorly on some instances. Similarly, Multistart does seem to improve on average over the much quicker EDD Plus.

Two-way aggregations like Table 4 provide a first look at how various problem parameters affects performance. With algorithms our primary interest, each other factor is displayed versus algorithms. We can immediately see, for example, that tardiness factor τ seems to have little impact on average, but others make considerable difference.

Two-way tabulations also make it easy to isolate cases where algorithm effectiveness is affected in disparate ways at different levels of some problem characteristics. For example, we can readily conclude that the gain from multiple local searches after EDD Plus is dramatic (18.6% to 1.4%) when due dates are similar (spread $w = 0.5$) but becomes much less pronounced with more variable due dates ($w = 2.5$)

7.4. Analysis of variance (ANOVA)

Although we have already warned against relying too heavily on formal statistical methods, there is a convenient tool for more comprehensive screening of effects that is widely available in standard statistical software: the *Analysis of Variance* or *ANOVA*. We believe ANOVA calculations can both help sort out the stories our results have to tell and provide some protection against being misled by effects probably due to sampling error.

The underlying notion of ANOVA (see for example Montgomery, 1991) is to assume that all the nonrandom variation in experimental observations is due to differences in mean performance at alternative levels of the experimental factors. For example, a computational experiment with problem parameter levels $p = 1, \ldots, P$ and algorithms $a = 1, \ldots, A$ might begin by assuming responses can be expressed

$$y_{pak} = \mu + \tau_p + \beta_a + \epsilon_{pak} \tag{4}$$

where y_{pak} is the measured response on replicate $k \in \{1, \ldots, K\}$ of tests at level p of the problem factor and level a of the algorithms. Here μ is the overall mean; τ_p is the incremental effect of the problem factor at level p assuming $\sum_p \tau_p = 0$; β_a is the corresponding incremental effect of algorithm a assuming $\sum_a \beta_a = 0$; and ϵ_{pak} is the random error in the *pak* observation. (For the moment we are ignoring the issue of blocking by solving the same instances with all algorithms.)

ANOVA proceeds by estimating each of the various means and partitioning the total *sum of squares*, or squared deviation from the sample global mean into separate parts due to each experimental factor and to error. For example in simple model (4)

$$\begin{aligned} SS_{\text{total}} &= \sum_{p=1}^{P}\sum_{a=1}^{A}\sum_{k=1}^{K}(y_{pak} - \bar{y}_{\ldots})^2 \\ &= A\sum_{p=1}^{P}(\bar{y}_{p\cdot\cdot} - \bar{y}_{\ldots})^2 + P\sum_{a=1}^{A}(\bar{y}_{\cdot a\cdot} - \bar{y}_{\ldots})^2 \\ &\quad + \sum_{p=1}^{P}\sum_{a=1}^{A}\sum_{k=1}^{K}(y_{pak} - \bar{y}_{p\cdot\cdot} - \bar{y}_{\cdot a\cdot} + \bar{y}_{\ldots})^2 \\ &= SS_{\text{problems}} + SS_{\text{algorithms}} + SS_{\text{error}} \end{aligned}$$

where bars denote sample means and dots indicate the subscripts over which averages were computed. After dividing by the number of degrees of freedom in each effect, the resulting *mean squares* indicate the relative importance of the various factors in explaining experimental outcomes.

Besides the fact that most computational experiments will have more than one problem parameter factor, two extensions arise in many cases. First, it is not unusual to find *interactions* between factors. That is, the effect of say algorithm a in our simple example may vary with the level of problem factor p. Neither problem nor algorithm means by themselves tell the whole story. Such interactions are accommodated in ANOVA model (4) by introducing

new offsets γ_{pa} for each *pa* combination to obtain

$$y_{pak} = \mu + \tau_p + \beta_a + \gamma_{pa} + \epsilon_{pak} \tag{5}$$

with the added conventions that $\sum_p \gamma_{pa} = 0$ for each a and $\sum_a \gamma_{pa} = 0$ for each p.

The other extension almost always needed in analyzing experiments on heuristics is an adjustment for *blocking* on problem instances. As noted in Section 3.2, there are both common sense and theoretical reasons to test all algorithms on the same instances, which is a form of blocking.

Blocking is accommodated in ANOVA models by introducing yet another linear term ϕ_{pk} for each instance k at each problem parameter level p. For example (5), the result is

$$y_{pak} = \mu + \tau_p + \beta_a + \gamma_{pa} + \phi_{pk} + \epsilon_{pak} \tag{6}$$

with added requirements $\sum_k \phi_{pk} = 0$ for all p. Instances k become a new "instance" factor in the experiment which is *nested*, in the terminology of experimental design, because it cannot be compared across levels of problem factors p.

Up to this point in our discussion of the Analysis of Variance, we have presumed merely that nonrandom variation is due solely to differences in means and that effects add as in Eqs. (4)–(6). The normality and common error variance assumptions usually associated with ANOVA come into play only when we try to assign statistical significance to whether a particular mean square is large relative to the error mean square.

There is sometimes reason to doubt at least the equal variance assumption of ANOVA significance testing in the heuristic experimentation context. For example, problem size is often a factor in tests of heuristics. We have already discussed in Section 5.1 how heuristic results can become either more consistent or more variable as size grows. Either way the error variance is not constant over different size levels.

When researchers are determined to make formal tests of significance, but doubt the common variance assumption, the remedy most often recommended (Montgomery, 1991; Conover, 1980) is to transform experimental responses. One effective way to accomplish this is using *ranks*, i.e., replacing the best response value by 1, the next best by 2, etc., with duplicate ranks allowed in the case of identical responses.

In the heuristic research context of this paper, ranking would typically be based on solution quality ratios or percents error. Results for each problem instance would be ranked in increasing (for a minimize model) ratio sequence, with the most effective algorithm on the instance scored 1, the next most effective given a 2, etc. Since the variation among a sequence of consecutive integers is predictable, the effect is to stabilize the variation within instance blocks. Then running ANOVA on the rank numbers may produce more reliable indications of factor significance.

7.5. Exploratory analysis using ANOVA

Having reviewed the main ideas of the Analysis of Variance, we are ready to demonstrate how we think it can be productively employed in computational experiments: helping to

Table 5. Analysis of variance for case study.

Source	DF	AnovaSS	MeanSquare	FValue	Pr>F
Alg	4	29700	7425	6.93	0.0001
Jobs	1	8576	8576	8.00	0.0048
Grpfrac	2	30852	15426	14.39	0.0001
Tardy	1	375	375	0.35	0.5541
Spread	1	7048	7048	6.58	0.0105
Inst(Jobs*Grpfrac*Tardy*Spread)	225	1411572	6273	5.85	0.0001
Alg*Jobs	4	15000	3750	3.50	0.0076
Alg*Grpfrac	8	17008	2126	1.98	0.0455
Alg*Tardy	4	4502	1125	1.05	0.3800
Alg*Spread	4	36777	9194	8.58	0.0001
Jobs*Grpfrac	2	14083	7041	6.57	0.0015
Jobs*Tardy	1	8376	8376	7.82	0.0053
Jobs*Spread	1	1512	1512	1.41	0.2351
Grpfrac*Tardy	2	40988	20494	19.12	0.0001
Grpfrac*Spread	2	22101	11050	10.31	0.0001
Tardy*Spread	1	1674	1674	1.56	0.2116
Error	936	1003134	1072		

structure the exploratory, analysis phase of investigation. Table 5 displays the ANOVA for our case study of Section 6.

Values in Table 5 assess all the main effects in the 1200 observations (240 instances against 5 algorithms) included in our experiment, including algorithms, number of jobs, group setup fraction, tardiness factor and spread, as well as potential two-way interactions among all these factors. The most important information is the rightmost column which estimates the probability that the factor has no effect. We do not have to accept these values as exact to believe that smallest values indicate the more interesting source of variation.

Notice that an instance factor has also been included as a nested effect within all the problem factors to account for blocking on problem instances. Such a nested factor cannot interact with problem characteristics, because it is not comparable across different problem levels, and we adopt the usual assumption that it is also not interacted with algorithms in order to have an estimate of random error.

Main effects showing significance: The most obvious clues in ANOVA results are the main experimental factors that seem to have greatest statistical significance. In Table 5 there are several: algorithms, problem size, group fraction, and possibly due date spread. These are the ones likely to consume most of our future analytic attention.

An indication of statistical significance is only an initial signal, not a final conclusion of the investigation. We need to see where the effects are occurring, and whether they have practical as well as statistical significance.

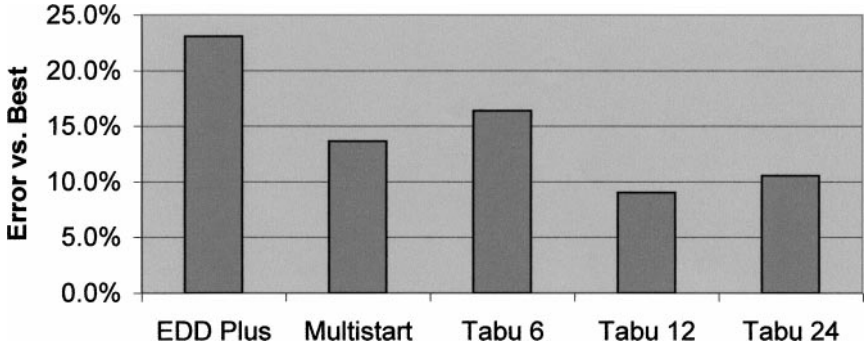

Figure 5. Algorithm means bar chart.

First we might produce a simple bar chart like figure 5. Differences averaging several percent do seem to be large enough to be practically important, but these are only means.

A different view comes from looking more closely at the 240 actual observations for each algorithm. Figure 6 provides histograms of the frequencies of different levels of error experienced for each of the algorithms. Multistart and the last two tabus stand out, but it is much less clear which we would prefer.

Negligible effects: For factors not involved in significant interactions, the next thing we can tell from ANOVA results is whether they seem to be negligible. A low level of factor significance suggests differences among the various levels are not even statistically distinguishable, let alone practically important. Any factors falling into this category can be aggregated (summed over) to simplify later presentation.

In our experiment of Table 5, the tardiness factor regulating tightness of due dates seems to have little importance. Before neglecting tardiness altogether, however, it might be appropriate to take a look at a frequency plot like figure 6.

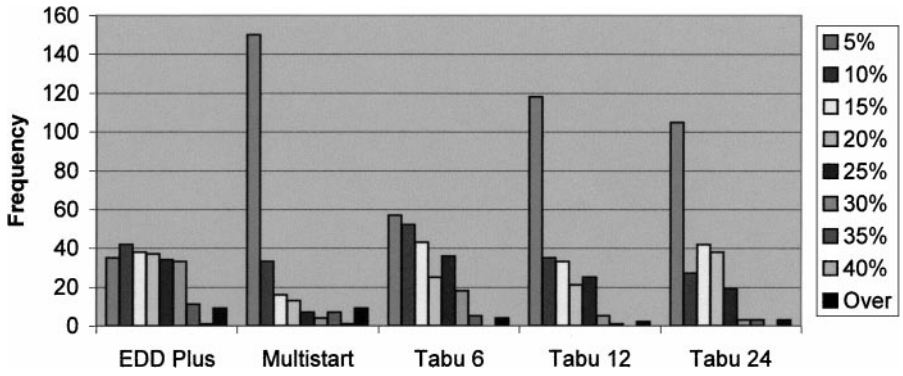

Figure 6. Algorithm error histograms.

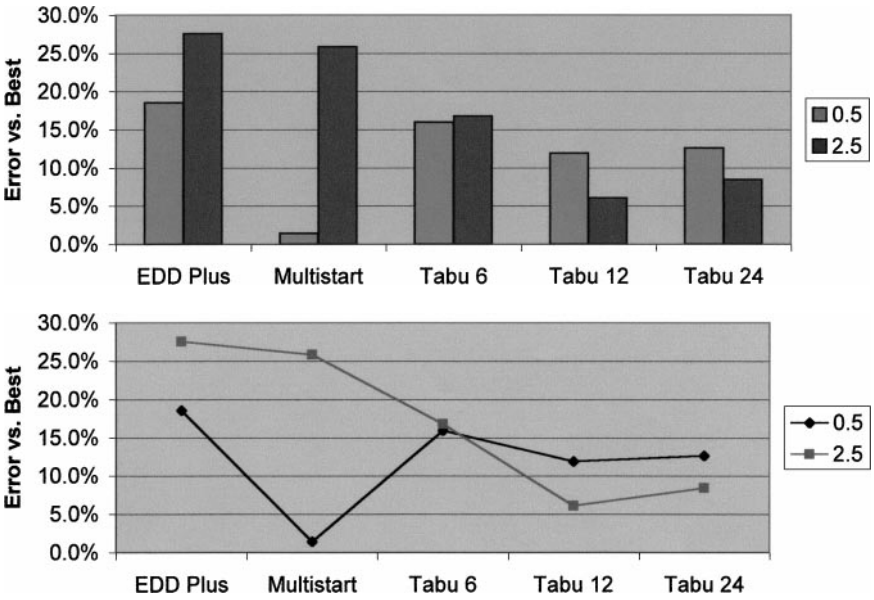

Figure 7. Graphical investigation of an interaction.

Interactions: One of the most useful sources of information in an ANOVA like Table 5 is the interactions which mean the impact of a first factor depends on the level of the second. The ANOVA setting, which makes it easy to estimate and examine such interactions, helps us to avoid overlooking phenomena that could easily be lost in a display of means like Table 3.

Table 5 lists several interactions that seem to be important, and each should be examined. Again, this is often best done graphically. Figure 7 illustrates two most obvious choices. The first compares results in bars, and the second in lines. Both show the impact of spread is quite different for some algorithms than others. Multistart does best when spread is small (due dates are similar). The tabu algorithms seem to prefer highly variable due dates.

7.6. Comparison of means

If there are only two levels of an experimental factor and an ANOVA on original or transformed responses shows the factor relatively significant, the two levels probably have distinguishable means. But what if there are three or more levels, as for example, is usually the case with the algorithms dimension of a computational experiment? An indication of factor significance merely says something is going on that does not seem to be due to chance. What we really want to know is which means are distinct, e.g., which algorithms are distinguishable from which others.

Graphic analysis like figures 5 and 6 can provide insight, but there is a standard extension of ANOVA, Duncan's Multiple Range Test, that adds statistical precision. As with all ANOVA tools it is an element of most commercial statistical software.

Table 6. Duncan's multiple range output for case study.

Duncan	Grouping	Mean	N	Algorithm
	A	23.066	240	EDD Plus
	B	16.371	240	Tabu 6
C	B	13.638	240	Multistart
C	B	10.523	240	Tabu 24
C		8.993	240	Tabu 12

The Duncan output for the algorithms factor in our experiment is displayed in Table 6. The results show that the EDD-Plus mean performance can be distinguished from the others, and the best tabu (list length 12) from the worst (length 6). Otherwise, differences seem to be within the noise of random variation.

Being built on ANOVA's normality and common variance assumptions, Duncan's test should probably not be taken as definitive in an exploration of computational experiment results. Still, the test is more precise than pairwise analyses because it explicitly accounts for simultaneous comparisons of several sample means. The consequence is that effects are more likely to be seen as indistinguishable, which is probably wise in experiments where the underlying assumptions are somewhat in doubt.

Goldsman and Nelson (1998) develop a whole series of alternative methods for comparing means from Monte Carlo simulation experiments working outside the ANOVA context. Unlike the Duncan goal of just determining which means are distinct, they try to pick the best of several alternatives, or at least a subset containing the best, based on experimental results.

These methods can accommodate blocking on problem instances (common random numbers in the simulation context), so they could be employed to say pick the best of several algorithms based on results for a collection of instances solved by all procedures. Unfortunately, they explicitly assume that all results for each algorithm are drawn from a common probability distribution. This limits their application in the heuristic testing context to identifying the best algorithm under a particular combination of problem parameter factors rather than the best overall.

7.7. Exploring time quality tradeoffs

To this point our discussion of exploratory analysis has presumed the main interest is solution quality—how close the heuristics come to optimal. However, tradeoffs between solution time and solution quality are sometimes relevant, especially when some of the subject algorithms run dramatically more rapidly than others (see Section 3.4).

Figure 8 illustrates the simple graphic display we find most instructive in studying such behavior. The horizontal axis is time, or equivalently, objective function evaluations. The vertical is solution value. A plot line tracks the sequence of objective function values visited as the search proceeds. In this example, which is one of the tabu searches in our case, we can clearly see how the algorithm wanders, but continues to find better and better solutions.

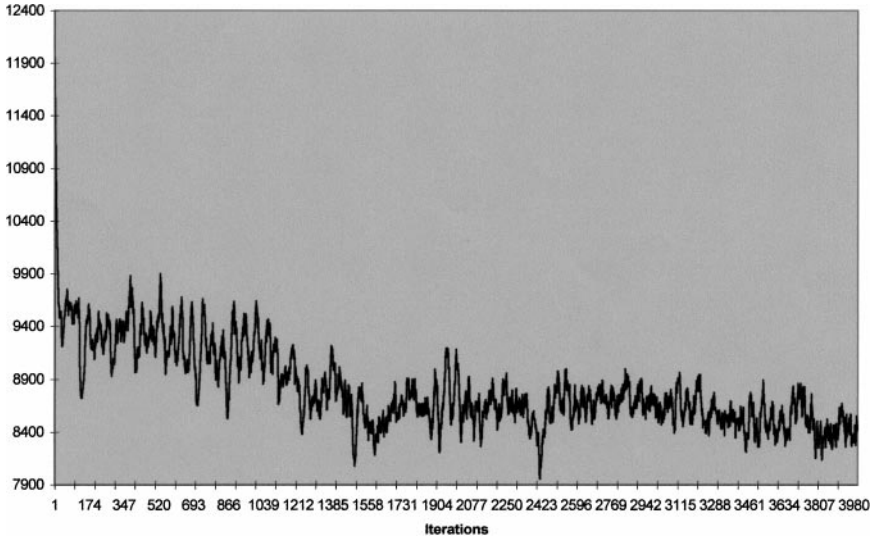

Figure 8. Plotting time quality tradeoffs.

The example in figure 8 displays only one search of one particular instance. However, multiple algorithms can easily be included if different line symbols are used. Also, multiple instances can be collected in a single graph if we standardize by making the vertical axis the ratio of current to optimal solution value.

8. Presentation of results

Unlike the highly exploratory analysis phase of examining experimental results, the presentation stage must get right to the point. Time and pages do not allow endless digressions, and we want significant findings to be communicated clearly.

8.1. Documentation and reproducibility

One unexciting but necessary part of every presentation should be background on how the experiment was conducted. It is essential that enough information be provided for a reader/listener to independently judge whether the experiment was conducted correctly and results are to be believed.

Great controversy has raged over how detailed such background needs to be. Standards proposed by Crowder, Dembo and Mulvey (1978), and later by Jackson et al. (1991), call for great precision about the computer environment in which the experiment was conducted and similar details that might affect performance. Others argue for a less formal discussion like our Section 6 (with some of the tutorial elements deleted).

At the heart of the controversy is the question of *reproducibility*. Should we demand that enough detail be provided for another researcher to reproduce the results presented in a published computational study?

We are inclined to say both "yes" and "no". There is no reason for published reports to include all the detail needed to reproduce a study. Still, those details should be documented somewhere. A working paper, a research memorandum, or a graduate thesis should have all the details needed to recover results. This means, for example,

- Results for every individual run, even though only summaries and averages may be published
- Details on every instance tested, either in the form of the actual data sets or of the parameter settings (including random number seeds) used in random instance generation
- Specification of what options, tolerances and other controls of algorithms were employed in experimentation
- Computer codes for both random instance generation and all tested algorithms

Some of these detail items may very well be proprietary, so we are not demanding that they be made public. But we do support the Crowder et al. (1978) minimum standard that at least the researchers themselves should be able to replicate the results.

We would also advocate at least including some comprehensive tabulation like Table 3 in the published report of the experiment. Discussion may quickly lead to discounting of the factors that have proved uninteresting, and some averaging has already been performed. Still, the reader is entitled to see as many details as can be captured in one or two summary tables before results are narrowed too much.

8.2. Table and figure readability

With most tables and graphs included in a report of empirical research, investigators will be using every idea they can come up with to get more information into limited space. Too often, readability suffers.

To illustrate, we invite the reader to compare our Table 3 with bad example Table 7. Some of the common flaws the latter illustrates include the following:

- *Tiny print*. Especially when a table is going to be used in an oral presentation, it is abundantly obvious that the numbers must be large enough to be read. Too often the quest to include more and more results leads to a vanishing font size.
- *Cryptic titling*. Far too frequently labels in tables and graphs are so cryptic that they cannot be understood without either a code book or constant reference back to the preceding text. Why not keep them as self-explanatory as possible? A good rule is that exhibits should be able to stand alone, independent of the surrounding text.
- *Lack of highlighting*. Not all numbers in an exhibit are of equal importance, and modern spreadsheets offer many tools for focusing on the most significant.

8.3. Significant digits

The bad example of Table 7 also illustrates another flaw often encountered when researchers do not think enough about their exhibits: meaningless digits. Modern calculators

Table 7. Bad example of a large table of results.

n	g	τ	w	A1	A2	A3	A4	A5
40	0.1	0.3	0.5	0.20660502	0	0.166305605	0.113796015	0.161847745
			2.5	0.114575438	0.109884701	0.067820095	0.004981718	0.075661086
		0.6	0.5	0.183687679	0	0.155879886	0.107167513	0.144702577
			2.5	0.111863398	0.039172852	0.071041119	0.016784564	0.064287894
	0.3	0.3	0.5	0.165178582	0.001061966	0.12898659	0.077529698	0.155110865
			2.5	0.343673654	0.343673654	0.138760641	0	0.18420094
		0.6	0.5	0.242220079	0	0.20128551	0.146797204	0.205816344
			2.5	0.166634557	0.166634557	0.116904669	0	0.144438247
	0.9	0.3	0.5	0.104667556	0.072595897	0.070278664	0.005824373	0.074009393
			2.5	0.019031543	0.019031543	0.003256142	0	0.000204499
		0.6	0.5	0.103421393	0.013312008	0.069346525	0.010594223	0.075072388
			2.5	0.533276251	0.533276251	0.059187526	0.026848865	0.468598607
80	0.1	0.3	0.5	0.271848878	0	0.254081079	0.23794587	0.173537096
			2.5	0.176902243	0.176902243	0.155389401	0.115310772	0
		0.6	0.5	0.269993431	0	0.260633789	0.226987261	0.185020519
			2.5	0.148636471	0.020357532	0.12613682	0.108566216	0.36446661
	0.3	0.3	0.5	0.230256466	0	0.208331871	0.182938675	0.120443426
			2.5	1.186935702	1.186935702	0.921054089	0.260031409	0
		0.6	0.5	0.26419497	0	0.247264753	0.220447992	0.176865614
			2.5	0.143135805	0.143135805	0.12169825	0.082827913	0
	0.9	0.3	0.5	0.083524671	0.083524671	0.059956387	0.030398271	0.001168031
			2.5	0.000101215	0.000101215	0.000101215	0	0.000101215
		0.6	0.5	0.101919132	0	0.092893654	0.069396146	0.040039464
			2.5	0.363450146	0.363450146	0.232377383	0113188118	0.03801273

and spreadsheets love digits, lots of digits. In the interest of both readability and scientific validity, researchers need think about the number of digits they should present.

We suggest that one of the simplest ways to focus attention on distinctions that matter, while at the same time beginning to deal with any sampling errors, is to systematically round such averages to a defensible number of significant digits before presenting results.

The first issue to consider is *practical significance*—how much of a difference in average response would be consider important enough to investigate further. For example, if responses are ratios of heuristic to optimal solution values (of a minimize problem), raw averages will be numbers like 27.349682. This particular number represents about a 27.3% error. One could argue whether another algorithm that averaged 27.2% is really any better, but one-hundredths of a percent certainly make no practical difference. Thus we would begin by rounding this and all other averages to at most one decimal place.

When computed averages involve random sampling, Song and Schmeiser (1994) offer a scheme for dealing with *statistical significance* or sampling error, in a similar digit-based manner. Their notion is elegantly simple: do not report any *random digits*, i.e., ones that could plausibly take on any of the ten possible values. To be more precise, they construct an approximate confidence interval around each sample mean and eliminate the 10^p digit if the width of the confidence interval is at least 10^{p+1}. (See Yoneda (1996) for an extension).

The easiest case is cell sample means that average only across the runs or the replicates of a given algorithm on problems with a fixed set of characteristics. In our case of Section 6,

that could be the heuristic errors for replicates $k = 1, \ldots, 10$ (different random instances) with say $n = 80$ jobs, group fraction $g = 0.3$, tardiness $= 0.6$, spread $w = 2.5$, solved by the EDD Plus algorithm. The actual values observed for that cell, which we denote y_k, are

10.0, 16.5, 17.1, 24.1, 8.8, 18.4, 17.8, 10.9, 10.7, 8.7

The sample mean and standard deviation for these $K = 10$ replicates can be computed as

$$\hat{\mu} = \frac{1}{K}\sum_{k=1}^{K} y_k = 14.3$$

$$\hat{\sigma} = \sqrt{\frac{1}{k-1}\sum_{k=1}^{K}(y_k - \hat{\mu})^2} = 5.17$$

Thus, if we are willing to assume $\hat{\mu}$ is approximately normally distributed, elementary statistics tells us a confidence interval that would cover the true mean approximately 95% of the time is

$$\hat{\mu} \pm t_{K-1,.975}\frac{\hat{\sigma}}{\sqrt{K}} = 14.3 \pm 3.70$$

where $t_{K-1,.975}$ is the classic t-statistic with $\alpha = .025$ and $K - 1$ degrees of freedom. With $10 \geq 2(3.70) \geq 1$, we would conclude that all decimal places of the cell mean are random. This average would best be reported simply as 14%.

The same idea can be applied with a bit more effort to sample means formed across cells of the experiment, or within cells when we have both multiple instances and multiple runs. For example, we might be interested in the mean solution quality yielded by the EDD Plus algorithm across all problem characteristics. The difference here arises from the fact that, if problem characteristics have any effect, the numbers being averaged are independent but from distributions with distinct means. This makes it less straightforward to estimate the standard deviation $\hat{\sigma}$ needed for a confidence interval.

If we are willing to assume the true error variance is the same for all observations in the experiment, such an estimate can be obtained from the Mean Square for Error (MS_E) of a standard Analysis of Variance (see Section 7.4). Then an approximate confidence interval is

$$\hat{\mu} \pm t_{df_E,.975}\sqrt{\frac{MS_E}{N}}$$

where df_E is the number of degrees of freedom in MS_E, N is the number of values averaged in $\hat{\mu}$, and $t_{df_E,.975}$ is the usual t-statistic. For the EDD Plus average of 23.1% in Table 4, $N = 240$, and (from ANOVA Table 5) $df_E = 936$ and $MS_E = 1072$. This produces a confidence interval half length of 4.15 which again indicates that only whole percents should be reported in the average.

8.4. Reporting variability

Sample means like those of Table 3 do a good job of showing central tendencies, but nearly every such average hides some degree of variability among the individual observations summarized. We believe the investigator is just as obligated to report such variability in a published summary of the experiment as he or she is to include a comprehensive table of averages like Table 3. But how can it be done succinctly?

Table 8 illustrates the four most common alternatives for a row of our experimental results. The first two approaches focus on the individual observations themselves. Part (a) uses standard deviations of the numbers averaged, and part (b) presents the best and worst single value encountered. Both alternatives still omit most of the detail. Still, we think either is adequate in a summary of results.

Parts (c) and (d) take a different tack. Instead of merely summarizing the actual data points observed, they attempt to depict how much sampling error may be present in the stated average. Alternative (c) reports the standard error of the average (standard deviation divided by $\sqrt{N}$), and (d) shows a confidence interval.

Methods described in the above discussion of significant digits could be used to obtain either form when data points are a sample from a much larger distribution (see Section 7.1). Given our skepticism about formal statistics, however, we would prefer the more straightforward presentations of parts (a) and (b), combined with rounding away random digits.

When a result is going to be one of the main findings of the experiment, we would argue for more. None of the methods of Table 8 can really do an adequate job of depicting the variability. In such cases, we suggest a more detailed graph. One way to do this is the frequency diagram of figure 6.

Table 8. Including variability in tables.

	EDD Plus	Multistart	Tabu 6	Tabu 12	Tabu 24
(a) Standard Deviations					
Sample Mean	11.5%	11.0%	6.8%	0.5%	7.6%
Standard Deviation	7.0%	7.2%	4.5%	1.6%	7.1%
(b) Ranges					
Sample Mean	11.5%	11.0%	6.8%	0.5%	7.6%
Worst	26.8%	26.8%	16.1%	5.0%	22.4%
Best	0.0%	0.0%	0.0%	0.0%	0.0%
(c) Standard Errors of Means					
Sample Mean	11.5%	11.0%	6.8%	0.5%	7.6%
Standard Error of Mean	2.2%	2.3%	1.4%	0.5%	2.3%
(d) Confidence Intervals					
Sample Mean	11.5%	11.0%	6.8%	0.5%	7.6%
95% Confidence Interval	$11.5 \pm 5.0\%$	$11.0 \pm 5.1\%$	$6.8 \pm 3.2\%$	$0.5 \pm 1.1\%$	$7.6 \pm 5.1\%$

8.5. Plots and graphs

One reason formal statistics may not be as central to presentation of experiments today as they were in earlier times is that so many options are available in modern spreadsheets and statistical software for displaying results graphically. We would argue that virtually every finding in a formal presentation of experimental results should be accompanied by a graphic display that makes it easy for readers or listeners to grasp what is being said. The whole purpose of empirical research is to discern patterns in numerical outcomes that are clouded by various forms of noise, and well-conceived graphic displays do more than any other presentation form to accentuate what matters.

We have consciously aimed at using a number of different types of simple plots and graphs through the last several sections of this paper to emphasize their effectiveness in presenting the results of empirical studies. Line graphs are good for illustrating how a response variable varies as a function of an experimental factor. Multiple lines can be used as in figure 7(b) to include the effects of a second factor. Bar graphs dramatize comparisons such as the average errors for different algorithms depicted in figure 5. An extremely useful type, which we think has been underutilized, is the frequency histogram illustrated in figure 6. They succinctly depict much more of the details for individual instances, and thus can reveal many effects that are obscured when aggregate statistics like averages and standard deviations are used alone. Scatter diagrams such as figure 4 do the same, although with less concentration and focus.

There are a number of excellent works on presenting quantitative information in graphical form, such as those by Tukey (1977), Tufte (1983, 1990), Jones (1996) and Wolff and Yaeger (1993). Still, the most practical approach to developing simple graphs and tables after running an extensive experiment is to download the results into a spreadsheet such as *Microsoft Excel*, which has easy-to-use statistical and graphing tools built in. While there are more sophisticated graphics packages available, such as *Delta Graph* and *MathCad*, the resources in *Excel* should be enough for all but the most sophisticated experimenter.

9. Conclusions

No one who has ever tried it thinks conducting empirical research on heuristics is easy, but it cannot be avoided. In all but the most structured of cases, experimental evaluation is the only tool available.

We do not claim to have the best answers to every challenge, or even to have listed all the possibilities. Still, we hope we have put the emphasis where it belongs. The toughest technical challenges are finding (or generating) suitable test instances, and assessing how close heuristic algorithms come to optimal. No investigation can hope to produce useful results if these matters are not addressed early and intensively.

We believe the rest of the procedures for conducting, analyzing and reporting empirical results can usually be boiled down to fairness and common sense. Formal statistics may help some analyses, but a well-conceived graph can do just as well. If adequate thought and effort is devoted to understanding what the research is about, what results are hidden in the experimental outcomes, and how these findings can be fairly and effectively communicated,

every study can contribute something to the critical but still underdeveloped science of heuristic algorithms.

Acknowledgments

We gratefully acknowledge the enormous assistance of Ms. Ebru Demirkol, who did most of the computational work for the experiment of Section 6. This research was supported in part by National Science Foundation grant number DMI-96-13708 and by the Intel Corporation.

References

Ahuja, R.K. and J.B. Orlin. (1996). "Use of Representative Operation Counts in Computational Testing of Algorithms." *INFORMS Journal on Computing* 8, 318–330.

Arthur, J.L. and J.O. Frendewey. (1988). "Generating Travelling Salesman Problems with Known Optimal Tour Length." *Journal of the Operational Research Society* 39, 153–159.

Baker, K.R. (1997). "Heuristic Procedures for Scheduling Job Families with Setups and Due Dates." Working paper, Amos Tuck School of Business Administration, Dartmouth College.

Baker, K.R. and M.J. Magazine. (1997). "Minimizing Maximum Lateness with Job Families." Working paper, Amos Tuck School of Business Administration, Dartmouth College.

Barr, R.S., B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. (1995). "Designing and Reporting on Computational Experiments with Heuristic Methods." *Journal of Heuristics* 1, 9–32.

Barr, R.S. and B.L. Hickman. (1993). "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures and Experts Opinions." *ORSA Journal on Computing* 5, 2–18.

Bixby, B. and G. Reinelt. (1990). *TSPLIB*, Software Library, Rice University, Houston, Texas.

Boender, C.G.E., A.H.G. Rinnooy Kan, L. Stougie, and G.T. Timmer. (1982). "A Stochastic Method for Global Optimization." *Mathematical Programming* 22, 125–140.

Burkard, R.E. and U. Fincke. (1983). "The Asymptotic Probabilistc Behaviour of Quadratic Sum Assignment Problems." *Z. Opns. Res.* 27, 73–81.

Coffman, E.G., G.S. Lueker, and A.H.G. Rinnooy Kan. (1988). "Asymptotic Methods in the Probablistic Analysis of Sequencing and Packing Heuristics." *Management Science* 34, 266–290.

Conover, W.J. (1980). *Practical Nonparametric Statistics*. New York: John Wiley.

Crawford, J.M. and L.D. Auton. (1983). "Experimental Results on the Crossover Point in Satisfiability Problems." In *Proceedings of the Eleventh National Conference on Artificial Intelligence AAAI93*. pp. 21–27.

Crowder, H.P., R.S. Dembo, and J.M. Mulvey. (1979). "On Reporting Computational Experiments with Mathematical Software." *ACM Transactions on Mathematical Software* 5, 193–203.

Dannenbring, D. (1977). "Estimating Optimal Solutions for Large Combinatorial Problems." *Management Science* 23, 1273–1283.

Demirkol, E., S.V. Mehta, and R. Uzsoy. (1998). "Benchmarks for Shop Scheduling Problems." *European Journal of Operational Research* 109, 137–141.

Derigs, U. (1985). "Using Confidence Limits for the Global Optimal in Combinatorial Optimization." *Operations Research* 33, 1024–1049.

Fargher, H.E. and R.A. Smith. (1994). "Planning in a Flexible Semiconductor Manufacturing Environment." In M. Zweben and M. Fox (eds.), *Intelligent Scheduling*. Morgan Kaufman.

Fisher, H. and G.L. Thompson. (1963). "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules." In J.F. Muth and G.L. Thompson (eds.), *Industrial Scheduling*. New Jersey: Prentice-Hall, Englewood Cliffs.

Fisher, R. and L. Tippett. (1928). "Limiting Forms of the Frequency Distribution of the Largest or Smallest Member of a Sample." *Proceedings of the Cambridge Philosophical Society* 24, 180–190.

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

Garfinkel, R.S. and K.C. Gilbert. (1978). "The Bottleneck Traveling Salesman Problem: Algorithms and Probabilistic Analysis." *Journal of the ACM* 25, 435–448.

Ghashghai, E. and R.L. Rardin. (1998). "Using a Hybrid of Exact and Genetic Algorithms to Design Survivable Networks." Working paper, School of Industrial Engineering, Purdue University.

Golden, B.L. (1977). "A Statistical Approach to the TSP." *Networks* 7, 209–225.

Golden, B.L. (1978). "Point Estimation of a Global Optimum for Large Combinatorial Problems." *Communications in Statistics* B7, 361–367.

Golden, B.L. and F.B. Alt. (1979). "Interval Estimation of a Global Optimum for Large Combinatorial Problems." *Naval Research Logistics Quarterly* 26, 69–77.

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

Golden, B.L. and W.R. Stewart. (1985). "Empirical Evaluation of Heuristics." In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), *The Traveling Salesman problem: A Guided Tour of Combinatorial Optimization.* New York: John Wiley.

Goldsman, D. and B.L. Nelson. (1999). "Comparing Systems via Simulation." In J. Banks (ed.), *Handbook of Stimulation.* New York: John Wiley.

Graham, R.L. (1969). "Bounds on Multiprocessor Timing Anomalies." *SIAM Journal of Applied Mathematics* 17, 416–429.

Greenberg, H.J. (1990). "Computational Testing: Why, How and How Much." *ORSA Journal on Computing* 2, 94–97.

Grötchel, M. and M.W. Padberg (1979). "On the Symmetric Traveling Salesman Problem I: Inequalities." *Mathematical Programming* 16.

Grötchel, M. and M.W. Padberg. (1985). "Polyhedral Aspects of the Traveling Salesman Problem I: Theory." In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), *The Traveling Salesman problem: A Guided Tour of Combinatorial Optimization.* New York: John Wiley.

Hall, N.G. and M. Posner. (1996). "Generating Experimental Data for Scheduling Problems." Research report, College of Business, Ohio State University.

Hariri, A.M.A. and C.N. Potts. (1997). "Single Machine Scheduling with Batch Set-up Times to Minimize Maximum Lateness." *Annals of Operations Research* 70, 75–92.

Hill, R.R. and C.H. Reilly. (1996a). "Multivariate Composite Distributions for Co-efficients in Synthetic Optimization Problems." Research report, Department of Industrial, Welding and Systems Engineering, Ohio State University.

Hill, R.R. and C.H. Reilly. (1996b). "The Effects of Coefficient Correlation Structure in Two-Dimensional Knapsack Problems on Solution Procedure Performance." Research report, Department of Industrial, Welding and Systems Engineering, Ohio State University.

Hooker, J. (1994). "Needed: An Empirical Science of Algorithms." *Operations Research* 42, 201–212.

Hooker, J. (1995). "Testing Heuristics: We Have It All Wrong." *Journal of Heuristics* 1, 33–42.

Jackson, R.H.F., 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.

Jones, C.V. (1996). *Visualization and Optimization*. Boston: Kluwer Academic Publishers. Boston.

Karp, R.M. (1977). "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane." *Mathematics of Operations Research* 2, 209–224.

Khoury, B.N., P.M. Paradalos, and D.Z. Zhu. (1993). "A Test Problem Generator for the Steiner Problem in Graphs." *ACM Transactions on Mathematical Software* 19, 509–522.

Kilbridge, M.D. and L. Wester. (1996). "A Heuristic Method of Assembly Line Balancing." *Journal of Industrial Engineering* 12, 394–398.

King, J.R. (1980). "Machine-Component Grouping in Production Flow Analysis: An Approach Using a Rank Order Clustering Algorithm." *International Journal of Production Research* 18, 213–232.

Klein, S. (1975). "Monte Carlo Estimation in Complex Optimization Problems." Ph.D. Dissertation, George Washington University, Washington, D.C.

Krishnamurthy, B. (1987). "Constructing Test Cases for Partitioning Heuristics." *IEEE Transactions on Computing* 36, 1112–1114.

Law, A.M. and W.D. Kelton. (1991). *Simulation Design and Analysis*, 2nd edn. New York: McGraw-Hill, Inc.

Lee, C.Y., J. Bard, M. Pinedo, and W.E. Wilhelm. (1993). "Guidelines for Reporting Computational Results in IIE Transactions." *IIE Transactions* 25, 71–73.

Lenstra, J.K. (1977). "Sequencing by Enumerative Methods." Mathematical Center Tract 69, Mathematisch Centrum, Amsterdam.

Lin, B.W. and R.L. Rardin. (1980). "Controlled Experimental Design for Statistical Comparison of Integer Programming Algorithms." *Management Science* 25, 1258–1271.

Lin, S. and B.W. Kernighan. (1973). "An Effective Heuristic Method for the Travelling Salesman Problem." *Operations Research* 21, 498–516.

Los, M. and C. Lardinois. (1982). "Combinatorial Programming, Satistical Optimization and the Optimal Transportation Network." *Transportation Research Board* 16B, 89–124.

McGeoch, C.C. (1996). "Towards an Experimental Method for Algorithm Simulation." *INFORMS Journal on Computing* 8, 1–15.

Mitchell, D., B. Selman, and H. Leveque. (1992). "Hard and Easy Distributions of SAT Problems." In *Proceedings of the Tenth National Conference on Artificial Intelligence AAA192*. 459–465.

Monma, C. and C.N. Potts. (1989). "On the Complexity of Scheduling with Batch Setup Times." *Operations Research* 37, 798–804.

Montgomery, D.C. (1991). *Design and Analysis of Experiments*, 3rd edn. New York: John Wiley.

Moscato, P. and M.G. Norman. (1998). "On the Performance of Heuristics on Finite and Infinite Fractal Instances of the Euclidean Traveling Salesman Problem." *INFORMS Journal on Computing* 10, 121–132.

Ovacik, I.M., S. Rajagopalan, and R. Uzsoy. (2000). "Integrating Interval Estimates of Global Optima and Local Search Methods for Combinatorial Optimization Problems." *Journal of Heuristics*, 6, 481–500.

Ovacik, I.M. and R. Uzsoy. (1997). *Decomposition Methods for Complex Factory Scheduling Problems*. Boston: Kluwer Academic Publishers.

Plicher, M.G. and R.L. Rardin. (1988). "Invariant Problem Statistics and Generated Data Validation: Symmetric Traveling Salesman Problems." ONR-URI Computational Combinatorics Report CC-87-16, Purdue University, West Lafayette, Indiana.

Pilcher, M.G. and R.L. Rardin (1992). "Partial Polyhedral Description and Generation of Discrete Optimization Problems with Known Optima." *Naval Research Logistics* 39, 839–858.

Pinedo, M. (1995). *Scheduling: Theory, Algorithms and Systems*. New York: Prentice-Hall.

Rardin, R.L. and B.W. Lin. (1982). "Test Problems for Computational Experiments-Issues and Techniques." In M. Beckmann and H. P. Kunzi (eds.), *Evaluating Mathematical Programming Techniques*, Berlin: Springer-Verlag.

Rardin, R.L., C.A. Tovey, and M.G. Pilcher. (1993). "Analysis of a Random Cut Test Instance Generator for the TSP." In P.M. Pardalos (ed.), *Complexity in Numerical Optimization*, World Scientific Publishing, pp. 387–405.

Sanchis, L.A. (1990). "On the Complexity of Test Case Generation for NP-Hard Problems." *Information Processing Letters* 36, 135–140.

Sanchis, L.A. (1994). "Test Case construction for the Vertex Cover Problem." In *Computational Support for Discrete Mathematics.* DIMACS Series in Discrete Mathematics and Theoretical American Mathematical Society. Computer Science, Providence, Rhode Island: vol. 15.

Sanchis, L.A. (1995). "Generating Hard and Diverse Test Sets for NP-Hard Graph Problems." *Discrete Applied Mathematics* 58, 35–66.

Song, W.T. and B.W. Schmeiser. (1994). "Reporting the Precision of Simulation Experiments." In S. Morito, S. Sakasegawa, H. Yoneda, M. Fushimi, and K. Nakano (eds.), *New Directions in Simulation for Manufacturing and Communications*. pp. 402–407.

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

Tufte, E. R. (1990). *Envisioning Information*. Cheshire, Connecticut: Graphics Press.

Tukey, J.W. (1977). *Exploratory Data Analysis*. Reading, Massachusetts: Addison-Wesley.

Uzsoy, R., C.Y. Lee, and L.A. Martin-Vega. (1992). "Scheduling Semiconductor Test Operations: Minimizing Maximum Lateness and Number of Tardy Jobs on a Single Machine." *Naval Research Logistics* 39, 369–388.

Wolff, R.S. and L. Yaeger. (1993). *Visualization of Natural Phenomena*. New York: Springer-Verlag.

Yoneda, K. (1996). "Optimal Number of Digits to Report." *Journal of the Operations Research Society of Japan* 33, 428–434.

Zanakis, S.H. (1977). "Computational Experience with Some Nonlinear Optimization Algorithms for Deriving

Maximum Likelihood Estimates for Three Parameter Weibull Distribution." *TIMS Studies in Management Science* 7, 63–77.

Zanakis, S.H. (1979). "A Simulation Study of Some Simple Estimators of the Three-Parameter Weibull Distribution." *Journal of Statistical Computing and Simulation* 9, 419–428.

Zanakis, S.H., J.R. Evans, and A.A. Vazacopoulos. (1989). "Heuristic Methods and Applications: A Categorized Survey." *European Journal of Operational Research* 43, 88–110.

Zanakis, S.H. and N. Mann. (1982). "A Good Simple Percentile Estimator of the Weibull Shape Parameter for Use When All Three Parameters Are Unknown." *Naval Research Logistics Quarterly* 29, 419–428.

Zemel, E. (1981). "Measuring the Quality of Approximate Solutions to Zero-One Programming Problems." *Mathematics of Operations Research* 6, 319–332.