Ordinal Comparison of Heuristic Algorithms Using Stochastic Optimization

Chun-Hung Chen

Dept. of Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104-6315
chchen@seas.upenn.edu

S. David Wu

Dept. of Industrial and Manufacturing Systems Eng.
Lehigh University
Bethlehem, PA 18015
sdw1@lehigh.edu

Liyi Dai

Dept. of Systems Science and Mathematics
Washington University
St. Louis, MO 63130
liyi@zach.wustl.edu

Abstract

The performance of heuristic algorithms for combinatorial optimization is oftentimes sensitive to problem instances. In extreme cases, a specialized heuristic algorithm may perform exceptionally well on a particular set of instances while fail to produce acceptable solutions on others. Such a problem-sensitive nature is most evident in algorithms for combinatorial optimization problems such as job shop scheduling, vehicle routing, and cluster analysis. This paper proposes a formal method for comparing and selecting heuristic algorithms (or equivalently, different settings of a same algorithm) given a desired confidence level and a particular set of problem instances. We formulate this algorithm comparison problem as a stochastic optimization problem. Two approaches for stochastic optimization, Ordinal Optimization and Optimal Computing Budget Allocation are applied to solve this algorithm selection problem. Computational testing on a set of statistical clustering algorithms in the IMSL library is conducted. The results demonstrate that our method can determine the relative performance of heuristic algorithms with high confidence probability while using a small fraction of computer times that conventional methods require.

1. Introduction

Many specialized algorithms and heuristics have been developed for combinatorial problems such as production scheduling and vehicle routing. The performance of these specialized algorithms is often sensitive to problem instances. As a common practice, researchers "tune" their algorithm to its best performance on the test set reported. This causes problem when the algorithms are to be used in industry applications where the algorithm configured for today's problem may perform poorly for tomorrow's instances. It is not practical to conduct massive experiments in a regular basis for the selection of a more effective algorithm setting. The same situation applies when a number of alternative algorithms are available to the decision-maker for selection.

The testing and comparison of heuristic algorithms have been a subject of much discussion in recent years. A familiar approach of algorithmic testing is to show that a proposed algorithm is better, at least in some aspect, than the current incumbent using either standard benchmark problems or randomly generated ones. Shortcomings of this approach are highlighted in a 1995 article by Hooker who argued that "Most experimental studies of heuristic algorithm resemble track meets more than scientific endeavors (pg. 33, Hooker 1995)." Among other problems of this "track meets" approach is how much should an algorithm developer tune his/her own algorithm vs. the competing algorithm, and whether such comparison is possible to generalize. He suggested that the approach of controlled experimentation is the way to alleviate potential biases and unfairness in algorithm comparison.

Empirical testing of algorithms has been the focus of research in a variety of contexts. In a recent article, Barr et al. (1995) provide a comprehensive view to the computational experiments for heuristic algorithms. Ahuja and Orlin (1992) suggest methods for comparing computational efficiency of network algorithms using the concept of operation counts. In the context of mathematical programming, the issue of conducting computational experiments has been addressed since the late 70's. Crowder et al. (1979), Jackson and Mulvey (1978) and Jackson et al. (1989) have provided important guidelines for computational experiments. Crowder and Sunders (1980) and Greenberg (1990) discuss performance measures such as algorithm robustness, reliability, and solution accuracy when comparing algorithms. Most relevant to this

paper are statistical methodologies for the design and analysis of computational experiments. McGeoch (1986), Bentley (1990), Golden et. al. (1986), Amini and Racer (1992) and Barton (1987) suggest various statistical techniques for this purpose. Of special interest is the approach suggested by McGeoch (1992) McGeoch and Tygar (1991) which uses variance reduction techniques to reduce the size of the computational experiments when analyzing sorting algorithms.

In this paper, we focus our attention on heuristic algorithms whose performance is sensitive to special structures of different problem instances while the algorithm performance can be improved through parametric tuning. We believe this represents a large class of algorithms including exact methods whose efficiency relies on particular search heuristics (e.g., branching rules in a branch and bound algorithm). This problem-sensitive nature is most evident in algorithms for combinatorial optimization problems such as job shop scheduling, vehicle routing, and cluster analysis. Clearly, given a set of problem instances one can find a parameter setting of a particular algorithm that is at least as good as all others under consideration. The ability to make this decision effectively has great practical importance. For example, scheduling problems in a manufacturing plant may vary in a frequent basis due to its highly dynamic nature. Due-date tightness, seasonality of order volumes and mixes, machine or tooling status, and changing operating policies may all alter the structure of the scheduling problem to be solved. As a result, the "best" algorithm or algorithm setting for solving a preconceived static problem may perform poorly in a day-to-day basis.

In this paper, we propose a stochastic optimization method designed to compare algorithms or algorithm settings in a timely and efficient fashion when a new set of problem instances arise. The purpose of this method is two fold: to provide a means for algorithm comparison and to provide a “self-tuning” mechanism for heuristic algorithms, i.e., to identify appropriate parameter settings of an algorithm given problem instances at hand.

In the next section, we formulate algorithm comparison as a stochastic optimization problem. In Sections 3 and 4, we present two optimization techniques, *ordinal optimization* and *optimal computing budget allocation*, for the solution of the problem. Section 5 gives two numerical examples for computational testing. Section 6 address several implementation issues and Section 7 concludes the paper.

2. Problem Statement

Suppose we wish to compare several different heuristic algorithms and each algorithm has different parameter settings. There are a total of k different algorithm-parameter combinations. For convenience we will call these combinations k different algorithms indexed by i, where $i = 1, 2, ..., k$. Our objective is to find an algorithm (or more accurately, an algorithm with a particular parameter setting) which performs the best over a particular problem instance as well as a specified range of variations for that instance. We further assume that there exists *a priori* statistical information regarding the variations. Thus, a particular problem instance and its variations form the set of problem instances under consideration. We define a *best algorithm* as one that provides the best expected performance for the current set of problem instances. Denote $h_i(w)$ as the result of applying algorithm i given the variations of the current problem instances characterized by w. $h_i(w)$ is a random variable characterized by the variation of the current problem instances. Specifically,

$$h_i(w_j) = \mathrm{E}_w[h_i(w)] + e_i(w_j). \qquad (1)$$

where $g_i(w_j)$ can be viewed as an estimation uncertainty or noise. (1) implies that $e_i(w_j)$ has zero mean. A good example for $e_i(w_j)$ could be a Gaussian noise, i.e., $N(0, s_i^2)$. Thus a best algorithm i^* can be chosen based on the expected performance measure $\mathrm{E}_w[h_i(w)]$, i.e., $i^* = \arg\min_i \mathrm{E}_w[h_i(w)]$.

For most real-life problems, neither the closed-form expression of $h_i(w)$ nor that of $\mathrm{E}_w[h_i(w)]$ exists. To estimate $\mathrm{E}_w[h_i(w)]$, one may take a sample of w, say w_j, and apply algorithm i to solve the problem based on this sample w_j. Then this is repeated for n samples. Thus, $\mathrm{E}_w[h_i(w)]$ is approximated by the value:

$$\hat{\mathrm{E}}_w[h_i(w)] \equiv \frac{1}{n}\sum_{j=1}^{n} h_i(w_j)$$

If we conduct independent sampling and the variance is finite, as the strong law of large numbers dictates, the following property holds with probability 1:

$$\frac{1}{n}\sum_{j=1}^{n} h_i(w_j) \rightarrow \mathrm{E}_w[h_i(w)], \text{ as } n \rightarrow \infty.$$

Since it is not possible to get an infinite number of test samples, the best algorithm must be chosen without knowing the exact value of the performance measure. The main difficulty is that with traditional sampling methods the estimate $\frac{1}{n}\sum_{j=1}^{n} h_i(w_j)$ converges slowly. In general, the rate of convergence for such a value estimate is at best $O(1/\sqrt{n})$ (Fabian 1971, Kushner and Clark 1978). The large n required for a good approximation implies that each algorithm must be repeated with a large number of samples, which translate to long computer time. In this paper we present a new approach for algorithm comparison using the notion of Ordinal Optimization and Optimal Computing Budget Allocation. Given a specified confidence interval, our method seeks to identify the best algorithm among a group of algorithms using a fraction of the computing effort required for traditional methods.

3. Ordinal Optimization

Although the estimate $\hat{\mathrm{E}}_w[h_i(w)]$ converges very slowly as n goes to infinity, recent research has shown that comparing *relative orders* of performances measures converges much faster than the performance measures themselves do. This is the basic idea of *ordinal comparison*. (Dai 1996) showed that under certain conditions the rate of convergence for ordinal comparison can be exponential. This result has important implications as it means that in many cases we could have a good estimate on the relative performance of algorithms while the value estimate on the actual algorithm performance is still poor. *Ordinal optimization* refers to the general approach that selects a subset of alternatives from the design space based on a certain criteria and a specified confidence level (Barnhart et al. 1994, Ho 1994a, Ho et al. 1992). *Ordinal comparison* can be used as a means for solving *ordinal optimization* if our goal is to find a good alternative in a group rather than to find an accurate estimate of the performance value. This idea is applicable not only to problems with discrete design space, but also to problems over a continuous design space (Cassandras and Bao 1994, Cassandras and Julka 1994, Chen and Kumar 1996, Chen et al. 1997a, Gong et al. 1994, Patsis et al. 1996, Yan and Mukai 1993). If our goal is to find the best or a subset of good designs rather than to find an accurate estimate of the best performance value

(as is true in many practical situations) it is advantageous to use ordinal comparison for selecting the best design.

Suppose we select an algorithm b using the following criterion:

$$b \equiv \arg \min_i \hat{\mathrm{E}}_w[h_i(w)] \left(\equiv \frac{1}{n}\sum_{j=1}^{n} h_i(w_j)\right).$$

Given the fact that we use only a finite number of testing samples, n, $\hat{\mathrm{E}}_w[h_i(w)]$ is an approximation to the true expected performance $\mathrm{E}_w[h_i(w)]$. An algorithm b with the smallest value of $\hat{\mathrm{E}}_w[h_i(w)]$ is not necessarily the true best algorithm (i.e., its true expected performance $\mathrm{E}_b[h_i(w)]$ may not be the best).

Definition 1. Define *correct selection* (CS) as the event that the selected algorithm b is actually the best algorithm. Define the *confidence probability* $P\{\mathrm{CS}\} \equiv P\{$ The current top-raking algorithm b is actually the best algorithm $\}$.

Based on the results from *ordinal comparison*, it is possible to establish the *relative order* of $\hat{\mathrm{E}}_w[h_i(w)]$ efficiently (i.e., to make the probability $P\{\mathrm{CS}\}$ sufficiently high) although the value of $\hat{\mathrm{E}}_w[h_i(w)]$ may converge slowly.

Theorem 1. Suppose the testing samples for each algorithm are i.i.d. and the testing samples between any two algorithms are independent. Assume that $h_i(w)$ (or $e_i(w)$) has a finite moment generating function. The ordinal comparison confidence probability converges to 1 exponentially. More specifically, there are $a > 0$, $b > 0$ such that

$$P\{\mathrm{CS}\} \geq 1 - ae^{-bn}.$$

Proof. Dai (1996), Theorem 5.1.

Since most statistical distributions (for example, normal, exponential, Erlang, and uniform distributions) have finite moment generating functions and therefore, Theorem 1 is applicable to most cases.

While the *confidence probability* $P\{\mathrm{CS}\}$ could converge at an exponential rate in *ordinal comparison*, a critical issue in applying it to *ordinal optimization* is the estimation of the $P\{\mathrm{CS}\}$

itself. Goldsman and Nelson (1994) provide an excellent survey on available approaches (e.g., Goldsman et al. (1991), Gupta and Panchapakesan (1979), and Law and Kelton (1991)) to estimating simulation confidence level. Bechhofer et al. (1995) give a systematic and more detailed discussion on this issue. Unfortunately most of these approaches are only suitable for problems with a small number of designs (e.g., Goldsman and Nelson (1994) suggest 2 to 20 designs). For real-life problems, the number of designs under consideration can be quite large. Using a Bayesian model, Chen (1996) developed an estimation technique to quantify the confidence level for ordinal comparison when the number of designs is large. In addition to the confidence probability, this approach also provides sensitivity information for each algorithm. The sensitivity information is useful if incremental computing effort is to be allocated during the comparison. We will make use of this particular feature in Section 4 to develop a computing budget allocation algorithm. The computation of $P\{\text{CS}\}$ in this paper is a special case discussed in Chen (1996).

Theorem 2. Let $\tilde{J}_i$, $i \in \{1, 2, .., b\text{-}1, b, b\text{+}1, .., k\}$, denote the random variable whose probability distribution is the posterior distribution of the expected performance for algorithm i under a Bayesian model. For a minimization problem,

$$P\{\text{CS}\} \geq \prod_{i=1, i\neq b}^{k} P\{\tilde{J}_b < \tilde{J}_i\} \equiv \textit{Approximate Probability of Correct Selection} \ (\textit{APCS}) \qquad (2.1)$$

For a maximization problem,

$$P\{\text{CS}\} \geq \prod_{i=1, i\neq b}^{k} P\{\tilde{J}_b > \tilde{J}_i\}. \qquad (2.2)$$

Proof of Theorem 2 is given in the Appendix. Note that the computation of *APCS* is simply a product of pairwise comparison probabilities, which is much easier to compute.

Under the Bayesian model, the posterior distribution $p(\tilde{J}_i)$ consists of information from both prior distribution and the test samples $\{h_i(w_j), j = 1, 2, .., n\}$. In other words, $p(\tilde{J}_i)$ summarizes the statistical properties of algorithm i's performance given the prior knowledge and the test results. To give an explicit model for $\tilde{J}_i$, we consider a Gaussian estimation noise $e_i(w_j) \sim \text{N}(0, s_i^2)$. Then the testing output is also normally distributed with mean $\text{E}_w[h_i(w)]$ and variances s_i^2.

In this paper, we consider non-informative prior distributions. This implies that no priori knowledge is given about the performance of any algorithm before the comparison starts. If s_i^2 is known (Bernardo and Smith 1984),

$$\tilde{J}_i \sim \mathrm{N}(\frac{1}{n}\sum_{j=1}^{n} h_i(w_j), \frac{s_i^2}{n}), \qquad \text{for } i = 1, 2, .., k.$$

If the variance is unknown, s_i^2 can be replaced by the sample variance

$$S_i^2 = \frac{1}{n-1}\sum_{j=1}^{n} \{h_i(w_j) - [\frac{1}{n}\sum_{s=1}^{n} h_i(w_s)]\}^2$$

and $\tilde{J}_i$ becomes *t*-distributed with *n* -1 degrees of freedom (Inoue and Chick 1998).

In Theorem 2 we establish the lower bound of $P\{\mathrm{CS}\}$ (in (2)) as the *Approximate Probability of Correct Selection* (*APCS*). While $P\{\mathrm{CS}\}$ is very difficult to obtain, *APCS* can be computed easily, for instance, in the case that variances are known and for a minimization problem,

$$APCS = \prod_{i=1, i\neq b}^{k} P\{\tilde{J}_b - \tilde{J}_i < 0\} = \prod_{i=1, i\neq b}^{k} \Phi\left(\frac{\frac{1}{n}\sum_{s=1}^{n} h_i(w_s) - \frac{1}{n}\sum_{s=1}^{n} h_b(w_s)}{\sqrt{\frac{s_i^2}{n} + \frac{s_b^2}{n}}}\right),$$

where is Φ is the standard normal cumulative distribution. Numerical testing in Chen (1996) shows that *APCS* provides a good approximation to $P\{CS\}$. We will therefore use *APCS* to approximate $P\{\mathrm{CS}\}$ in this paper.

Intuitively, *APCS* provides a convenient stopping criterion for the process of algorithm comparison using *ordinal optimization*. As the number of test samples *n* increases, the variance $\frac{s_i^2}{n}$ decreases and more confidence can be given to the sample mean. Using the *APCS* measure and the basic property of $\tilde{J}_i$, we design an iterative algorithm comparison experiment as follows:

Consider a set of algorithms for comparison. Allocate a small number of test samples for each algorithm, then rank the algorithms according to their estimated relative performance. Select the "best" (highest ranked) algorithm. Compute the approximate probability of correct

selection (*APCS*) for the current ordinal comparison. If the current selection reaches the desired level of confidence, stop; otherwise, allocate more test samples to the algorithms and repeat the process. Continue until the desired level of confidence is reached.

The algorithm comparison procedure is stated more formally as follows:

Algorithm Selection using Ordinal Optimization (OO)

Step 0. Specify a satisfactory confidence level P^*
Perform n_0 testing samples for all algorithms,
$l \leftarrow 0$,
$N_1^l = N_2^l = \cdots = N_k^l = n_0$.

Step 1. Calculate $APCS(N_1^l, N_2^l, \cdots, N_k^l)$.
If $APCS(N_1^l, N_2^l, \cdots, N_k^l) \geq P^*$, stop; Otherwise, go to Step 2.

Step 2. Perform additional t testing samples for algorithm i, $i = 1, ..., k$.
$N_i^{l+1} = N_i^l + t$, for $i = 1, ..., k$.
$l \leftarrow l + 1$, Go to Step 1.

The key to the above algorithm is the *APCS* measure which takes advantage of the exponential convergence property of ordinal comparison. As the number of test samples increases, the probability of selecting the correct top-ranking algorithm increases rapidly. In Section 5, we will demonstrate computationally the performance of this algorithm selection procedure using ordinal optimization.

To apply the algorithm, we need to select the initial number of testing samples, n_0, and the one-time increment, t. It is well understood that n_0 cannot be too small as the estimates of the mean and the variance may be very poor, resulting in terminating the comparison too early. A

While *ordinal optimization* could significantly reduce the computational cost for algorithm selection, there is potential to further improve its performance by adjusting in each iteration the amount of additional samples (i.e., t) based on the relative performance of algorithm i. In this section, we present a technique called *optimal computing budget allocation* (OCBA) which makes use of this idea. The OCBA approach can be summarized as follows: Start the algorithm selection procedure using *ordinal optimization.* In each iteration, compute a "promising index" for each algorithm under consideration, allocate incremental computing budgets to "more promising" algorithms while a subset of algorithms may be declared inferior and allocated no additional budget. As the experiment continues, the relative-performance estimations improve and promising algorithms are re-determined for further testing. This procedure continues until a pre-specified confidence level is obtained for the algorithm ranking.

Suppose we could find the allocation of testing samples to all algorithms, which minimizes total computation cost while obtaining the desired confidence level. Then we can optimally decide which algorithm will receive how many computing budgets in each iteration of the experiment. Let N_i be the number of testing samples of algorithm i. If comparison is performed on a sequential computer and the difference of computation costs using different algorithms is negligible, the total computation cost can be approximated by $N_1 + N_2 + \cdots + N_k$. The goal is to choose N_i for all i such that the total computation cost is minimized, subject to the restriction that the confidence level defined by *APCS* is greater than some satisfactory level. This optimization problem in its simplest form can be stated as follows.

$$\min_{N_1,\cdots,N_k} \{ N_1 + N_2 + \cdots + N_k \}$$

$$\text{s.t. } APCS \geq P^*. \qquad (3)$$

where P^* is a user-defined confidence level requirement.

Some difficulties in solving (3) include the following:

a) There is no closed-form expression for the confidence level *APCS*.

b) The confidence level $APCS(N_1, N_2, .., N_k)$ can be computed only after all N_1, N_2, .., N_k testing samples for algorithm 1 through k, respectively, are performed.

c) $N_1, N_2, .., N_k$ are integers and the number of combinations for $N_1, N_2, .., N_k$ is large even for moderate k.

In general, solving (3) as an *a priori* optimization problem is difficult due to a typically large k and the fact that the information required for calculating *APCS* is poor. Since the very purpose of OCBA is to reduce the computation cost of algorithm comparison, there is little incentive to exert too much effort in solving (3) itself. The additional cost of solving (3) must be properly balanced with the benefits of budget allocation.

As a heuristic for the solution to OCBA (i.e., to find the best $N_1, N_2, .., N_k$), we sequentially select a subset of "promising" algorithms in each iteration of the computational experiment. This procedure continues until $APCS \geq P^*$. We define *promising algorithms* as those which maximize the estimated improvement of *APCS*. A critical issue in this approach is the determination of a set of promising algorithms, or more specifically, the estimation of the new $P\{\text{CS}\}$ if additional t testing samples are performed on algorithm s.

Definition 2. Define $EPCS(N_1, N_2, .., N_{s-1}, N_s+t, N_{s+1}, .., N_k)$ as an estimated $P\{\text{CS}\}$ if additional t testing samples are performed on algorithm s. *EPCS* is computed using the statistical information after $N_1, N_2, .., N_k$ testing are completed for algorithms 1, .., k, respectively.

To minimize the efforts of reaching a desired confidence level, in each iteration we test the algorithm that has a maximum *promising index* (*PI*) defined as follows:

$$PI(s) \equiv EPCS(N_1, N_2, .., N_{s-1}, N_s+t, N_{s+1}, .., N_k) - APCS(N_1, N_2, .., N_{s-1}, N_s, N_{s+1}, .., N_k).$$

Chen, et al. (1997b) suggests a simple and effective way to estimate the *EPCS* (without loss of generality we only consider minimization problems here):

If $s \neq b$,

$$EPCS(N_1, N_2, .., N_{s-1}, N_s+t, N_{s+1}, .., N_k) = P\{\tilde{J}_b < \hat{J}_s\} \cdot \prod_{i=1, i\neq b, i\neq s}^{k} P\{\tilde{J}_b < \tilde{J}_i\},$$

$$\text{where } \tilde{J}_i \sim \text{N}\left(\frac{1}{N_i}\sum_{j=1}^{N_i} h_i(w_j), \frac{s_i^2}{N_i}\right),\ \hat{J}_s \sim \text{N}\left(\frac{1}{N_s}\sum_{j=1}^{N_s} h_s(w_j), \frac{s_s^2}{N_s+t}\right).$$

And if $s = b$,

$$EPCS(N_1, N_2, .., N_{b-1}, N_b+t, N_{b+1}, .., N_k) = \prod_{i=1, i\neq b}^{k} P\{\hat{J}_b < \tilde{J}_i\},$$

where $\tilde{J}_i \sim \mathrm{N}(\frac{1}{N_i}\sum_{j=1}^{N_i} h_i(w_j), \frac{s_i^2}{N_i})$, $\hat{J}_b \sim \mathrm{N}(\frac{1}{N_b}\sum_{j=1}^{N_b} h_b(w_j), \frac{s_b^2}{N_b + t})$.

Note that the expression of *EPCS* is similar to that of *APCS* in Eq. (2). In fact, *EPCS* can be obtained by substituting $\hat{J}_s$ with $\tilde{J}_s$ in *APCS*. The only difference between the two is their variances, i.e., the former is $\frac{s_s^2}{N_s + t}$ and the latter is $\frac{s_s^2}{N_s}$. In effect, we use the statistical information on N_s to estimate *APCS* at $N_s + t$ by decreasing the sample variance from $\frac{s_s^2}{N_s}$ to $\frac{s_s^2}{N_s + t}$. Thus, *EPCS* provides sensitivity information about how *APCS* will change if additional t testing samples are performed on algorithm *s*.

Under the above framework, "promising" refers to high improvement of the overall comparison confidence level. Since we intend to minimize the total number of testing samples, we select and test a subset of most promising algorithms in each iteration, then repeat the process until *APCS* achieves the desired level, P^*.

It is worthwhile to compare our OCBA with the well-known computing budget allocation

designs, we heuristically determine the value of m, attempting to balance the extra computation with the potential saving in overall computer time. The procedure is summarized as follows:

A Sequential Approach for Optimal Computing Budget Allocation (OCBA)

Step 0. Perform n_0 testing samples for all algorithms,
$l \leftarrow 0$,
$N_1^l = N_2^l = \cdots = N_k^l = n_0$.

Step 1. If $APCS(N_1^l, N_2^l, \cdots, N_k^l) \geq P^*$, stop, otherwise, go to Step 2.

Step 2. Calculate $PI(s)$ for all algorithms $s = 1, 2, .., k$.

Step 3. Find the set $S(m) \equiv \{ s : PI(s)$ is among the highest $m\}$

Step 4. Perform additional τ testing samples for algorithm i, $i \in S(m)$.
Set $N_i^{l+1} \leftarrow N_i^l + \tau$, for $i \in S(m)$, and $N_i^{l+1} \leftarrow N_i$

implementation differences in terms of memory management, data structures and coding techniques are minimal. Given the task of selecting a best heuristic algorithm for the problem instances under consideration and a specified confidence level, we focus our analysis on the actual savings in computer time over traditional methods.

Cluster analysis has been used to solve machine-grouping problems in manufacturing environments (Askin and Standridge 1993). Many cluster analysis approaches have been proposed over the years. Among them are hierarchical clustering techniques (Anderberg 1973), optimization algorithms (Everitt 1993), fuzzy logic approaches and neural network based algorithms. Without complicating the issues in algorithm comparison, we will compare only hierarchical clustering algorithms in this study.

Hierarchical clustering utilizes a machine-part incidence matrix with 0-1 entries. The incidence matrix provides information about which parts are processed on which machines. Table 1.1 (on page 16) gives an example of such machine-part incidence matrix. Conventionally, rows of the incidence matrix correspond to machines while columns correspond to parts. In a matrix X, element x_{ij} is equal to 1 if part j visits machine i at some point of its process, and 0 otherwise. The objective of a clustering algorithm is to reorder the rows and columns of the matrix such that blocks of 1's appears, as much as possible, along the diagonal direction of the matrix. In this case, adjacent parts in the resulting matrix tend to use the same set of machines (i.e., the matrix defines machine cells). Ideally we would like to transform the matrix in Table 1.1 into a block diagonal matrix, in which 1's are located only in the diagonal blocks. In practice, this may not be possible to achieve, i.e., some of the 1's may not belong to any blocks (as shown in Table 1.2 on page 16).

Before starting the hierarchical clustering algorithm, the machine-part incidence matrix is transformed into a similarity matrix so that the clustering algorithms can be used for the machine-grouping problems. A similarity matrix contains information about the degree to which each machine is related to the other machines according to the parts they process. More specifically, each entry to the similarity matrix is a *similarity coefficient*, one for each machine pair. Let n_i be the number of parts that visit machine i and n_{ij} be the number of parts that visit machine i and machine j. We define three different *similarity coefficients* s_{ij}, as follows:

L_1 norm: $s_{ij} = \frac{n_{ij}}{n_i} + \frac{n_{ij}}{n_j}$,

L_2 norm: $s_{ij} = \sqrt{\left(\frac{n_{ij}}{n_i}\right)^2 + \left(\frac{n_{ij}}{n_j}\right)^2}$,

L_∞ norm: $s_{ij} = \max\left(\frac{n_{ij}}{n_i}, \frac{n_{ij}}{n_j}\right)$,

Hierarchical clustering algorithm forms machine groups by processing the similarity matrix $[s_{ij}]$ obtained from the incidence matrix. Initially, each machine belongs to a cluster of its own. In each succeeding iteration, the algorithm combines individual machine or groups of machines into clusters based on some (heuristic) criterion. The columns and rows for the cluster members are removed from the matrix and replaced by similarity coefficients aggregated from the clusters. This forms a reduced similarity matrix among machine clusters. The clustering algorithm continues to reduce similarity matrix until all machines are in a specified number of clusters. This procedure produces a range of clustering solutions, which is summarized in a "dendogram." A dendogram is a tree in which the root and the leave levels represent the trivial one-cluster, and no-cluster solutions, respectively. The levels between the root and the leaves represent all non-trivial solutions found in the algorithm. Given a specified number of clusters, a solution can be found at a corresponding level of the dendogram. A detailed description of general hierarchical clustering algorithms can be found in (Askin and Standrige, 1993).

A number of hierarchical clustering algorithms has been developed and widely used. Main differences between these heuristic algorithms include the criterion used for combining clusters, or the way similarity coefficients are updated. To demonstrate our proposed algorithm comparison procedure, we implemented six well-known hierarchical clustering algorithms (Anderberg 1973) as follows:

i) Single Linkage (SGL): Combine a pair of *similarity coefficients* by saving the maximum for further iterations,

ii) Complete Linkage (CPL): Combine a pair of *similarity coefficients* by saving the minimum for further iterations,

iii) Average Between Linkage (ABL): Combine a pair of *similarity coefficients* by saving their average for further iterations,

iv) Average Within Linkage (AWL): Combine a pair of *similarity coefficients* by re-calculating the average of *similarity coefficients* of all machines within the merged cluster for further iterations,

v) The Centroid Method (CTD): Suppose clusters *x* and *y* are being merged as a new cluster *z*. The number of parts that visit machine group *z* is calculated by $n_z = (n_x+n_y)/2$. Then the *similarity coefficients* between the newly merged cluster and all other clusters are re-calculated.

vi) Ward's Method (WAR): Combines those machine groups whose merger produces the minimum increase in the total sum of squares of similarity coefficients within the merged cluster.

Since the three different ways of calculating similarity coefficient can be considered a parametric element of hierarchical clustering algorithm, we combine them with each of the above hierarchical clustering algorithm. This results in 18 distinct "heuristic algorithms" to be tested. We repeat the computational experiments on two distinctly different sets of problem instances.

To produce a more realistic set of problem instances we assume some portion, say γ, of the elements in this machine-part incidence matrix will be perturbed from "1" to "0", or from "0" to "1". Further, we assume that the exact instance of the matrix after all the changes is not known *a priori*. Thus, the objective is to find a clustering algorithm (out of the 18) which has the best expected performance given the base problem instance and its variations.

Performance Measures

To determine the performance of different clustering algorithms we use three performance measures: average similarity (*CR*), total number of outliners (*OL*), and a linear combination of *CR* and *OL*. *CR* measures the system-wide similarity or the similarity among all clusters. More specifically,

$$CR = (\sum_{i=1}^{L} R_i^2)^{1/2},$$

where L is the number of cluster,

T_i is the total number of members in cluster i,

A_i is the centroid of the cluster i,

$M_{ij} = \| A_i - A_j \|_2$, the distance between two centroids i and j.

$S_i = \frac{1}{T_i} \sum_{i=1}^{T_i} \| s_{ij} - A_j \|_2$, the dispersion of cluster i,

$R_{ij} = \frac{S_i + S_j}{M_{ij}}$, the similarity between clusters i and j,

$R_i = [\sum_{j=1}^{L} R_{ij}^2]^{1/2}$, the similarity between cluster i and the rest of clusters.

On the other hand, *OL* is a count of the total number of 1's which do not belong to any of the blocks along the main diagonal, i.e., outliners (an example is shown in Table 1.2). For the third performance measure, we consider a linear combination of *CR* and *OL* and assume both *CR* and *OL* are equally important. Since they are on different scales, we have to choose appropriate scaling factors. We first conduct a numerical experiment (with $\gamma = 5\%$) to estimate $E[CR_i]$ and $E[OL_i$

approximation to $E_w[h_i(w)]$ can be obtained when the relative standard error (i.e., the ratio of the standard deviation of $\hat{E}_w[h_i(w)]$ to $|\hat{E}_w[h_i(w)]|$) is less than 0.1%.

Experiment 1. Base testing on the first set of problem instances

We consider the matrix in Table 1.1 as a base for the first set of problem instances (from Groover 1987). 5% of the elements in this matrix will be perturbed from "1" to "0", or from "0" to "1" (i.e., γ=5%). The resulting rankings for each performance measure are shown in Table 2.

Table 1.1. Machine-part incidence matrix for Example 1. (Note that values not shown are 0's)

	Parts																			
Machines	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1		1	1	1	1		1				1	1	1	1			1	1	
2			1			1			1	1		1								
3	1		1			1												1	1	1
4					1								1							
5		1							1		1				1				1	
6										1								1		
7		1					1		1	1	1				1	1				
8							1			1										
9							1			1										
10	1				1	1			1			1	1					1		
11	1		1		1							1								
12				1				1							1	1				1
13		1					1											1		
14		1													1					
15	1	1	1		1	1		1				1	1	1				1	1	1
16								1		1				1						1
17				1						1										
18			1						1			1	1		1	1	1			
19									1						1		1			
20												1	1		1					
21	1			1					1						1					
22	1		1	1		1		1	1	1				1	1					1
23	1			1					1						1					

Table 1.2. A solution to the problem in Table 1.1 after reordering its rows and columns.

	Parts																			
Machines	6	3	12	18	1	13	5	11	17	7	16	2	10	19	8	14	20	4	15	9
1	1	1	1	1	1	1	1							1	1	1		1	1	
2	1	1	1										1							1
3				1	1									1	1		1	1		
4						1	1													
10	1		1	1	1	1													1	1
11		1	1		1		1													
15	1	1	1	1	1	1	1		1			1			1	1	1			
5								1			1	1		1						1

Table 2. Ranking of Algorithms Using Different Performance Measures ($\gamma = 5\%$)

	Ranking of Algorithms Using *CR*		Ranking of Algorithms Using *OL*		Ranking of Algorithms Using *Combination*	
		CR		*OL*		*Comb.*
1	AWL-L∞	32.18	ABL-L1	44.12	AWL-L∞	0.62
2	AWL-L2	34.55	ABL-L2	45.58	SGL-L∞	0.64
3	AWL-L1	34.78	ABL-L∞	45.87	AWL-L1	0.64
4	SGL-L∞	37.67	WAR-L∞	47.83	AWL-L2	0.64
5	SGL-L2	40.23	WAR-L1	49.30	SGL-L2	0.64
6	SGL-L1	42.20	WAR-L2	49.37	SGL-L1	0.65
7	CTD-L∞	43.58	SGL-L1	55.63	CTD-L∞	0.73
8	CTD-L2	61.05	SGL-L2	56.37	CTD-L2	0.79
9	CTD-L1	64.70	SGL-L∞	57.84	CTD-L1	0.81
10	WAR-L2	101.28	CTD-L1	60.03	WAR-L2	0.94
11	WAR-L∞	105.38	AWL-L1	60.27	WAR-L∞	0.95
12	WAR-L1	105.40	AWL-L2	60.64	WAR-L1	0.96
13	ABL-L2	135.91	CTD-L2	60.66	ABL-L2	1.10
14	ABL-L1	139.46	AWL-L∞	61.00	ABL-L1	1.11
15	ABL-L∞	143.03	CTD-L∞	65.51	ABL-L∞	1.15
16	CPL-L∞	152.31	CPL-L∞	130.56	CPL-L∞	1.82
17	CPL-L2	156.58	CPL-L2	136.28	CPL-L2	1.89
18	CPL-L1	157.44	CPL-L1	139.41	CPL-L1	1.92

Comparing different performance measures in Table 2, we see that the Average Within

Experiment 2. Testing the Efficiency of Ordinal Optimization and OCBA: Instance One

We consider three different percentages of variations: γ = 2%, 5%, and 15% in forming the set of test instances. We consider two ways of applying the stochastic optimization techniques. In the first configuration we apply *ordinal optimization* to the algorithm comparison process described above, and measures the degree of saving achieved. In the second configuration we apply *ordinal optimization* and *optimal computing budget allocation* (OCBA) in sequence. We stop the comparison procedure when the confidence probability *APCS* is no less than P^*, which means that the required confidence level is achieved. Different confidence level requirements are tested at 90%, 95%, 99% and 99.5%. We repeat this testing 100 times, each run uses a different random seed. Since the total numbers of test samples are different from one run to another run due to different random seeds, we compute their average as the computation cost. Tables 3, 4, and 5 contain the testing results for *CR*, *OL*, 0.0113*CR* + 0.0147*OL* performance measure, respectively. We compare the total numbers of test samples using different approaches for different cases.

From the three tables, we see that with the application of *ordinal optimization*, the "best algorithm" can be identified with high probability within a much shorter time when compared to the traditional method. The time savings factors range from 23 to 739. If the OCBA is used, the factors can be as high as 1656. The required computation cost of our approach depends heavily on how close the performance measure of the best algorithm is to that of other algorithms. Intuitively, the closer in performance the best algorithm is comparing to other algorithms, the harder it is to identify the best algorithm. Clearly, the confidence level requirement *P** affects the required computation as well. In general, a higher confidence level requirement requires longer computation time for ordinal optimization. However, the speedup of OCBA over ordinal optimization (i.e., S.F.O.O.) increases as *P** increases. This is because a higher computational requirement on ordinal optimization offers more opportunity for OCBA to manipulate the budget allocation.

Larger γ implies larger variation of testing problems. As a result, the performance measures in our experiments, *CR*, *OL*, and their combination, become larger as γ increases. Since the stopping criterion for the traditional approach is that the standard deviation of $\hat{\mathrm{E}}_w[h_i(w)]$ is less than 0.1% of $|\hat{\mathrm{E}}_w[h_i(w)]|$, the computational cost of the traditional approach is lower for larger γ.

However, when using this approach the confidence level of identifying the best algorithm can not be guaranteed. On the other hand, our approaches tend to take longer computation time as γ increases, since the variances of the performance measure becomes larger and it becomes more difficult to isolate the best algorithm.

Table 3. Computation cost for *CR* performance measure with different confidence level requirements.

	γ = 2%. The comp. cost using traditional approach is 75092.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost[1]	T.S.F.[2]	Comp. Cost[1]	T.S.F.[2]	S.F.O.O.[3]
90.0%	101.5	739.8	63.8	1177.0	1.6
95.0%	120.4	623.7	72.2	1040.0	1.7
99.0%	183.2	409.9	93.3	804.8	2.0
99.5%	220.3	340.9	106.5	705.1	2.1
	γ = 5%. The comp. cost using traditional approach is 72993.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	341.2	213.9	107.8	677.1	3.2
95.0%	469.7	155.4	141.9	514.4	3.3
99.0%	883.2	82.6	228.1	320.0	3.9
99.5%	1031.4	70.8	243.0	300.4	4.2
	γ = 15%. The comp. cost using traditional approach is 63327.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	326.8	193.8	98.1	645.5	3.3.
95.0%	428.3	147.9	130.3	486.0	3.3
99.0%	846.3	74.8	209.1	302.8	4.0
99.5%	1163.1	54.4	219.8	288.1	5.3

[1] The average of the total numbers of test samples over the 100 independent experiments.

[2] T.S.F. is the time saving factor as compared to the traditional method.

[3] S.F.O.O. is the speedup factor of OO+OCBA as compared to using OO only.

Table 4. Computation cost for *OL* performance measure with different confidence level requirements.

	γ = 2%. The comp. cost using traditional approach is 36231.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	175.1	206.9	74.0	489.6	2.4
95.0%	284.7	127.3	90.0	402.6	3.2
99.0%	466.7	77.6	130.2	278.3	3.6
99.5%	592.0	61.2	152.3	237.9	3.9
	γ = 5%. The comp. cost using traditional approach is 29126.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	408.9	71.2	113.5	256.6	3.6
95.0%	650.8	44.7	146.2	199.2	4.5
99.0%	1137.5	25.6	247.8	117.5	4.6
99.5%	1248.6	23.3	281.6	103.4	4.4
	γ = 15%. The comp. cost using traditional approach is 35941.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	241.3	148.9	87.6	410.3	2.8
95.0%	374.9	95.9	108.7	330.6	3.4
99.0%	677.8	53.0	150.2	239.3	4.5
99.5%	864.7	41.6	183.8	195.5	4.7

Table 5. Computation cost for $0.0113CR + 0.0147OL$ performance measure with different confidence level requirements.

	γ = 2%. The comp. cost using traditional approach is 176583.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	245.1	720.5	106.6	1656.5	2.3
95.0%	337.6	523.1	131.3	1344.9	2.6
99.0%	518.5	340.6	188.7	935.8	2.7
99.5%	628.0	281.2	209.2	844.1	3.0
	γ = 5%. The comp. cost using traditional approach is 159450.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	544.3	292.9	163.5	975.2	3.3
95.0%	647.6	246.2	203.9	782.0	3.2
99.0%	1071.9	148.8	291.6	546.8	3.7
99.5%	1189.0	134.1	331.7	480.7	3.6
	γ = 15%. The comp. cost using traditional approach is 103191.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	607.1	170.0	160.8	641.8	3.8
95.0%	873.0	118.2	204.0	505.9	4.3
99.0%	1502.6	68.7	305.5	337.8	4.9
99.5%	1837.9	56.1	387.2	266.5	4.7

Experiment 3. Base testing on the second set of problem instances

We test our approaches using another base instance shown in Table 6.1 (Chandrasekharan and Rajagopalan 1986). We again consider three performance measures: *CR*, *OL*, and a linear combination of *CR* and *OL*. Similarly, we conduct a pre-testing in order to obtain the appropriate scaling factor for the combination of *CR* and *OL*, which is $0.0068CR + 0.0411OL$ in this case. All settings in the numerical experiment are the same as those in Experiment 1. The resulting rankings for each performance measure with $\gamma = 5\%$ are shown in Table 7.

Comparing different performance measures in Table 7, we see that the Average Between Linkage heuristic with $L\infty$ similarity coefficient performs better than other algorithms if we are minimizing *OL*. However, its performance is among the worst if we are interested in *CR*. In addition, comparing Table 7 with Table 2, Average Within Linkage heuristic is the best heuristic for *CR* in example 1. However, the best heuristic for *CR* in example 2 is the Single Linkage heuristic. The performance of heuristic algorithms can be highly sensitive not only to the performance measure, but also to the problem instances.

Table 6.1. Machine-part incidence matrix for Example 2.

	Parts																
Machines	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17


8
1 1 1 1 1 1

Table 7. Ranking of Algorithms Using Performance Measure *CR*

	Ranking of Algorithms Using *CR*		Ranking of Algorithms Using *OL*		Ranking of Algorithms Using *Combination*	
		CR		*OL*		*Comb.*
1	SGL-L∞	52.30	ABL-L∞	13.42	SGL-L∞	0.66

Table 8. Computation cost for *CR* performance measure with different confidence level requirements. T.S.F. is the time savings factor when our techniques are applied as compared with the traditional approach. S.F.O.O. is the speedup factor of using OO+OCBA over using OO only. Thus S.F.O.O. can be an indicator of the effectiveness of the OCBA technique.

	γ = 2%. The comp. cost using traditional approach is 52689.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	309.0	170.5	95.9	549.4	3.2
95.0%	669.5	78.7	127.3	413.9	5.3
99.0%	1106.4	47.6	208.3	252.9	5.3
99.5%	1540.4	34.2	250.3	210.5	6.2
	γ = 5%. The comp. cost using traditional approach is 46264.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	586.7	78.9	132.1	350.2	4.4
95.0%	890.0	52.0	202.9	228.0	4.4
99.0%	1744.9	26.5	321.4	144.0	5.4
99.5%	2000.1	23.1	370.2	125.0	5.4
	γ = 15%. The comp. cost using traditional approach is 21761.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	466.7	46.6	136.6	159.4	3.4
95.0%	830.1	26.2	170.7	127.5	4.9
99.0%	1191.7	18.3	275.7	79.0	4.3
99.5%	1439.6	15.1	306.7	71.0	4.7

Table 9. Computation cost for *OL* performance measure with different confidence level requirements.

	γ = 2%. The comp. cost using traditional approach is 143231.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	433.9	330.1	120.5	1188.7	3.6
95.0%	824.6	173.7	153.6	932.5	5.4
99.0%	1483.9	96.5	303.3	472.3	4.9
99.5%	2223.0	64.4	340.2	421.0	6.5
	γ = 5%. The comp. cost using traditional approach is 181289.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	99.9	1814.8	60.6	2991.7	1.6
95.0%	128.8	1407.6	67.1	2701.9	1.9
99.0%	206.6	877.5	85.1	2130.4	2.4
99.5%	249.1	727.8	93.3	1943.1	2.7
	γ = 15%. The comp. cost using traditional approach is 211442.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	159.4	1326.5	70.2	3012.1	2.3
95.0%	216.1	978.5	81.5	2594.5	2.7
99.0%	354.9	595.8	112.7	1876.2	3.1
99.5%	385.2	548.9	131.2	1611.6	2.9

Table 10. Computation cost for 0.0068*CR* + 0.0411*OL* performance measure with different confidence level requirements.

	γ = 2%. The comp. cost using traditional approach is 1342026.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	151.1	8881.7	68.3	19648.9	2.2
95.0%	197.1	6808.8	76.6	17519.9	2.6
99.0%	351.5	3818.0	99.9	13433.6	3.5
99.5%	434.7	3087.2	113.0	11876.3	3.8
	γ = 5%. The comp. cost using traditional approach is 1234472.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	309.0	3995.0	95.9	12872.4	3.2
95.0%	669.5	1843.9	127.3	9697.3	5.3
99.0%	1106.4	1115.8	208.3	5926.4	5.3
99.5%	1540.4	801.4	250.2	4933.9	6.2
	γ = 15%. The comp. cost using traditional approach is 755589.				
	Ordinal Optimization		**Ordinal Opt + OCBA**		
P^*	Comp. Cost	T.S.F.	Comp. Cost	T.S.F.	S.F.O.O.
90.0%	431.2	1752.3	125.7	6011.0	3.4
95.0%	924.8	817.0	177.6	4254.4	5.2
99.0%	1611.9	468.8	287.4	2629.0	5.6
99.5%	1824.6	414.1	347.0	2177.5	5.3

6. General Applicability of the Method

As previously suggested, the proposed scheme could be used to compare different algorithms or different parameter settings of an algorithm. On the other hand, the approach could be used on

implementing a designed computational experiment, assuming each algorithm as a capsulated module that takes the problem input and provides a solution that can be evaluated by a certain performance measure.

However, from an implementation point of view, several issues must be addressed when setting up heuristic algorithms for comparison. First, the method assumes the existence of a problem instance and its statistical variations. In practice, this information must be made available from historic data or other *a priori* knowledge of the problem. For example, for production scheduling problems, static information on job routing and processing time distributions are typically available in the information system. Given *a priori* information on the incoming job orders, a problem instance could be constructed with proper statistical variations. . The sources of statistical variations may include processing times, setup requirements, job routings and the presence of alternative machines. With the capability of current information systems, constructing a problem instance and its statistical variations for a specified planning period should not be difficult. A similar setting can be applied to vehicle routing problems where incoming customer demands over the near-term planning periods are analyzed *a priori*, providing a statistical basis for the algorithm selection test instances.

Another issue involves the testing of a diverse set of algorithms. While comparing different parameter settings of an algorithm or a set of "standardized" algorithms (such is the case in the IMSL library) is straightforward, comparing independently developed algorithms remains difficult. As well illustrated by Barr et al. (1995) and Hooker (1995), the difficulties are due to the wide variety of programming options available (e.g, data structures, memory management scheme), presumed computing platforms, and the very intepretation of *algorithm comparison*. Clearly some effort on setting up the ground rules, or even standardizing the heuristic algorithms is necessary before a fair comparison can start. As specialized software libraries (e.g., class libraries in C++, or Java) becomes common place for mathematical and statistical algorithms, it is not unreasonable to assume that the algorithms under comparison are standardized under a common set of assumptions.

7. Conclusions

In this paper, we demonstrated that the performance of algorithms could be highly sensitive to problem instances, parameter settings and performance measures. In extreme cases, an algorithm

may perform exceptionally well on a particular set of instances while fail to produce acceptable solutions on others. Furthermore, as we have shown in our experiments, a heuristic algorithm may be superior on a particular performance measure while performing poorly on another performance measure. Two methods of ordinal comparison presented in this paper offer an efficient scheme for selecting heuristic algorithms given a desired confidence level and a particular set of problem instances. Computational testing on a set of statistical clustering algorithms demonstrates that our method can effectively compare and select algorithms that are expected to perform the best on given problem instances. The time savings factors of using *ordinal optimization* in the computational testing range from 23 to 8881. The application of *optimal computing budget allocation* on *ordinal optimization* can further push the savings factor up to as high as 19648.

Our proposed approach for algorithm comparison is quite general with a few mild assumptions. A major assumption is that the variation/noise of the testing result has a finite moment generating function, which is true for most real-world distributions. The restriction of a finite moment generating function is to ensure the exponential convergence property of ordinal optimization. However, even if this assumption is not valid and thus, the exponential convergence property is not ensured, the OCBA scheme is still applicable and could still reduce computation cost significantly.

Appendix

The computation of *APCS* in this paper is a special case discussed in Chen (1996). In the following we give the proof of Theorem 2. Let X_1, X_2, .., X_k be k random variables, and X_2, X_3, .., X_k are mutually independent.

Lemma 1. $P\{X_1 < X_i \cap X_1 < X_j \} \geq P\{X_1 < X_i \} P\{X_1 < X_j \}$, $i \neq j \neq 1$.

<pf> $P\{X_1 < X_i \cap X_1 < X_j \}$

$= \int\int P\{X_1 < a \cap X_1 < b \} f_{X_i, X_j}(a,b)\, da\, db$ (f is the density fun.)

$= \int\int P\{X_1 < a \cap X_1 < b \} f_{X_i}(a) f_{X_j}(b)\, da\, db$ (independence)

$\int\int P\{X_1 < a\} P\{X_1 < b \} f_{X_i}(a) f_{X_j}(b)\, da\, db$

(because $P\{X_1<a \cap X_1<b\} = \min[P\{X_1<a\}, P\{X_1<b\}] \quad P\{X_1<a\}P\{X_1<b\}$)

$= \int P\{X_1 < a\} f_{X_i}(a)\, da \cdot \int P\{X_1 < b\} f_{X_j}(b)\, db$

$= P\{X_1 < X_i\}\ P\{X_1 < X_j\}$. #

Lemma 2. $P\{X_1 < X_i,\ i \neq 1\} \quad \prod_{i=2}^{k} P\{X_1 < X_i\}$

<pf> $P\{X_1 < X_i,\ i \neq 1\}$

$= P\{X_1 < \max[X_2, .., X_k]\}$

$= P\{X_1 < X_2 \cap X_1 < \max[X_3, .., X_k]\}$

$P\{X_1 < X_2\}\ P\{X_1 < \max[X_3, .., X_k]\}$ (According to Lemma 1)

As the same way,

$P\{X_1 < X_i,\ i \neq 1\}$

$P\{X_1 < X_2\}\ P\{X_1 < X_3\}\ P\{X_1 < \max[X_4, .., X_k]\}$

.....

$P\{X_1 < X_2\}\ P\{X_1 < X_3\}\ P\{X_1 < X_4\}\ \cdot\cdot P\{X_1 < X_k\}$ #

Proof of Theorem 2.

Under the Bayesian model,

$P\{\text{CS}\}$ = $P\{$ The current top-raking algorithm b is actually the best algorithm $\}$.

$= P\{\ \tilde{J}_b < \tilde{J}_i,\ i \neq b\}$

Apply Lemma 2,

$P\{\text{CS}\} \geq \prod_{i=1, i \neq b}^{k} P\{\tilde{J}_b < \tilde{J}_i\}$ #

Acknowledgment

The authors would like to thank D. Chang, D. Saparilla, and M. Messina at the University of Pennsylvania for implementing the computer programs in Section 4. In addition, the authors would like to thank the Editor, Associate Editor, and three anonymous referees for their helpful suggestions and valuable comments.

References

1. Ahuja, R.K., T.L. Magnanti and N.B. Orlin, *Network Flows: Theory, Algorithms and Applications,* Prentice Hall, Englewood Cliffs, NJ, 1993.
2. Amini, M.M. and M. Racer, "A Variable-Depth-Search Heuristic for the Generalized Assignment Problem," Working Paper, Civil Engineering, Memphis State University, Memphis, Tenn., 1992.
3. Askin, Ronald G., and Charles R. Standridge, *Modeling and Analysis of Manufacturing Systems*, John Wiley & Sons, Inc., 1993.
4. Anderberg, Michael R. *Cluster Analysis For Applications*, Academic Press, 1973.
5. Banks, J., and Carson, J. S., Nelson, B. L., *Discrete-Event System Simulation*, Prentice-Hall, 1995.
6. Barnhart, C. M., Wieselthier, J. E., and Ephremides, A., "Ordinal Optimization by Means of Standard Clock Simulation and Crude Analytical Models," *Proceedings of the 33rd IEEE Conference on Decision and Control*, pp. 2645-2647, December, 1994.
7. Barr, R.S. B.L. Golden, J.P. Kelly, M.G.C. Resende and W.R. Stewart, "Designing and Reporting on Computational Experiments with Heuristic Methods," *Journal of Heuristics,* Vol. 1, No. 1, pp. 9-32, 1995.
8. Barton, R.R., "Testing Strategies for Simulation Optimization," *Proceedings of the 1987 Winter Simulation Conference,* A. Thesen and H. Grant (eds.), IEEE Press, New York, 391-401, 1987.
9. Bechhofer, R. E., T. J. Santner, and D. M. Goldsman, *Design and Analysis of Experiments for Statistical Selection, Screening, and Multiple Comparisons*, John Wiley & Sons, Inc., 1995.
10. Bentley, J.L., "Experiments on Geometric Traveling Salesman Heuristics," *Computer Science Technical Report 151,* AT&T Bell Laboratories, Holmedel, NJ, 1990.
11. Bernardo, J.M., and A.F.M. Smith. *Bayesian Theory*. Wiley, 1994.
12. Berry D. A., and D. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*, Chapman and Hall, 1985
13. Casella, G., and Berger, R. L., *Statistical Inference*, Wadsworth, 1990.
14. Cassandras, C. G., and Bao, G., "A Stochastic Comparison Algorithm for Continuous Optimization with Estimations," *Proceedings of the 33rd IEEE Conference on Decision and Control*, pp. 676-677, December, 1994.
15. Cassandras, C. G., and Julka, V., "Descent Algorithms for Discrete Resource Allocation Problems," *Proceedings of the 33rd IEEE Conference on Decision and Control*, pp. 676-677, December, 1994.
16. Chandrasekharan, M.P. and R. Rajagopalan. "An ideal seed non-hierarchical clustering algorithm for cellular manufacturing," *International Journal of Production Research*, Vol 24, No. 2. pp.451-464, 1986.
17. Chen, C. H. "A Lower Bound for the Correct Subset-Selection Probability and Its Application to Discrete Event System Simulations," *IEEE Transactions on Automatic Control*, Vol. 41, No. 8, pp. 1227-1231, August 1996.

18. Chen, C. H., L. Dai, H. C. Chen, and E. Yucesan, "Efficient Computation of Optimal Budget Allocation for Discrete Event Simulation Experiment," submitted to *Operations Research*, 1998.
19. Chen, C. H., V. Kumar, and Y. C. Luo, "Motion Planning of Walking Robots in Environments with Uncertainty," *IEEE Robotics and Automation Magazine*, Vol. 4, No. 4, December 1997a.
20. Chen, H. C., C. H. Chen, L. Dai, and E. Yucesan, "New Development of Optimal Computing Budget Allocation For Discrete Event Simulation," *Proceedings of the 1997 Winter Simulation Conference*, pp. 334-341, December 1997b.
21. Chen, C. H. and V. Kumar, "Motion Planning of Walking Robots in Environments with Uncertainty," *Proceedings of the 1996 IEEE International Conference on Robotics and Automation* , pp. 3277-3282, April 1996.
22. Crowder, H.P., R.S. Dembo and J.M. Mulvey, “On Reporting Computing Experiments with Mathematical Software,” *ACM Transactions on Mathematical Software,* 5, pp. 193-203, 1979.
23. Crowder. H.P. and P.B. Saunders, “Results of a Survey on MP Performance Indicators,” *COAL Newsletter,* January, pp. 2-6, 1980
24. Dai, L., "Convergence Properties of Ordinal Comparison in the Simulation of Discrete Event Dynamic Systems," *Journal of Optimization Theory and Applications*, November, 1996.
25. Dai, L., and Chen, C. H. "Rate of Convergence for Ordinal Comparison of Dependent Simulations in Discrete Event Dynamic Systems," To appear in *Journal of Optimization Theory and Applications*, Vol.94, No.1, July 1997.
26. Everitt, B. S., *Cluster Analysis*, 3rd Edition, John Wiley & Sons Inc., 1993.
27. Fabian, V., *Stochastic Approximation*

35. Ho, Y. C., "Heuristics, Rule of Thumb, and the 80/20 Proposition," *IEEE Transactions on Automatic Control*, Vol. 39, pp. 1025-1027, 1994.

36. Ho, Y. C., "Overview of Ordinal Optimization," *Proceedings of the 33rd IEEE Conference on Decision and Control*, pp. 1975-1977, December 1994.

37. Ho, Y. C., R. S. Sreenivas, and P. Vakili, "Ordinal Optimization of DEDS", *Journal of Discrete Event Dynamic Systems*, 2, #2, pp. 61-88, 1992.

38. Holland, J. H. *Adaptation in Natural and Artificial Systems*, MIT Press, 1992.

39. Holland, J. H. "Genetic Algorithms and The Optimal Allocation of Trials," *SIAM J. Comput.*, 1973.

40. Hooker, J.N., "Needed: An Empirical Science of Algorithms," *Operations Research*, Vol. 42, No.2, pp.201-212, 1994.

41. Hooker, J.N., "Testing Heuristics: We Have It All Wrong," *Journal of Heuristics*, Vol. 1,No.1, pp. 33-42, 1995.

42. Inoue, K., and S. E. Chick. "Comparison of Bayesian and Frequentist Assessments of Uncertainty for Selecting the Best System," to appear in the *Proceedings of the 1998 Winter Simulation Conference*, 1998.

43. Jackson, R.H.B., and J.M. Mulvey, "A Critical Review of Comparisons of Mathematical Programming Algorithms and Software (1953-1977)," *Journal of Research of the National Bureau of Standards*, 83, pp. 563-584, 1978.

44. Jackson, R.H.B., P.T. Boggs, S.G. Nash and S. Powell, "Report of the ad hoc committee to Revise the Guidelines for Reporting Computational Experiments in Mathematical Programming," *COAL Newsletter*, 18, pp. 3-14.

45. Kushner, H. J., and Clark, D. S., *Stochastic Approximation for Constrained and Unconstrained Systems*, Springer-Verlag, 1978.

46. Law, A. M. and W. D. Kelton, *Simulation Modeling & Analysis*, McGraw-Hill, Inc., 1991.

47. McGeoch, C.C., *Experimental Analysis of Algorithms.* Unpulished Ph.D. dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1986.

48. McGeoch, C.C., "Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups," *Comput. Sur.*, Vol. 24, 195-212, 1992.

49. McGeoch, C.C. and D. Tygar, "Optimal Sampling Strategies for Quicksort," in *Proceedings of 28th Allerton Conference on Computing, Control and Communication*, University of Illinois, Urbana-Champaign, Ill., 62-71, 1991.

50. Patsis, N. T., C. H. Chen, and M. E. Larson, "SIMD Parallel Discrete Event Dynamic System Simulation," *IEEE Transactions on Control Systems Technology*, Vol. 5, No. 3, pp. 30-41, January 1997.

51. Yan, D. and H. Mukai, "Optimization Alogrithm with Probabilistic Estimation," *Journal of Optimization Theory and Applications*, Vol.79, pp. 345-371, 1993.