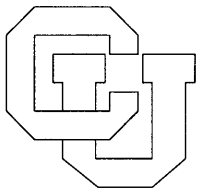

University of Colorado at Denver

Heuristics for Efficient Classification

Kathryn Fraughnaugh, Jennifer Ryan,
Holly Zullo, and Louis Anthony Cox, Jr.

December 1993 UCD/CCM Report No. 9

CENTER FOR COMPUTATIONAL MATHEMATICS REPORTS

Heuristics for Efficient Classification

Kathryn Fraughnaugh, Jennifer Ryan, and Holly Zullo
University of Colorado at Denver
P.O. Box 173364
Denver, Colorado 80217-3364
Phone: (303) 556-8461 FAX: (303) 556-8550
Bitnet: kjones@cudnvr.denver.colorado.edu

Louis Anthony Cox, Jr.
US West Advanced Technologies
4100 Discovery Drive, Suite 280
Boulder, Colorado 80303
Phone: (303) 541-4221
Bitnet: tony@atqm.advtech.uswest.com

December, 1993

Abstract

The classification problem is to determine the class of an object when it is costly to observe the values of its attributes. This type of problem arises in fault diagnosis, in the design of interactive expert systems, in reliability analysis of coherent systems, in discriminant analysis of test data, and in many other applications. We introduce a generic decision rule that specifies the next attribute to test at any location in a decision tree. Random searches and tabu searches are applied to determine the best specific form of the rule. The most successful heuristics that we developed are based on the tabu search paradigm. We present computational results for problems with a variety of characteristics and compare our heuristics to an exact dynamic programming algorithm.

This work was supported by the Colorado Advanced Software Institute (CASI) under Grant 93-007. CASI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development.

1 Introduction

This paper reports the results of a collaboration between a team of researchers at the University of Colorado at Denver and at US West Advanced Technologies. The objective of this project was to develop, implement and test heuristics to find an inspection strategy for classification. That is, given a set of attribute values for deciding class membership, prior statistical information about the relative frequencies of attribute values, and costs of inspection of attribute values, what is an optimal sequential inspection strategy for determining the class of some object? We developed a number of different algorithms using various heuristic techniques including hillclimbing, random search, and tabu search. We also developed an exact dynamic programming algorithm for comparison purposes on small problems.

This work makes several important contributions to the fields of classification and of heuristic search. Earlier work to develop heuristics for sequential inspection and classification strategies has emphasized *constructive* heuristics [2] and [3]. Indeed at first glance, the complexity needed to fully represent a solution (a decision tree) makes an iterative search heuristic seem impractical if not impossible. This paper introduces a simple dynamic rule for classification that is easily represented. Equally important, simple variation of components of the rule lead to a wide search of the set of all decision trees. This is borne out by our results of Section 4.4, where the outcome of a random search of all decision trees is compared with that of a random search of decision trees that can be represented by our rule. The construction of our rule is flexible, and could easily be varied to encompass important components of classification problems that differ from ours.

The extreme simplicity of the representation of our rule leads to an appealing application of tabu search. In most applications, the number of possible moves grows with the size of the problem. Searching the entire neighborhood of solutions becomes less and less attractive when solving larger problems. In general, one must search a smaller neighborhood for large problems, reducing the effectiveness of the search. With our rule, there are only nine possible moves, *no matter what the size of the problem.* Thus we can search the entire neighborhood, even for large problems. This feature of our rule would carry over to related applications. For instance, even if there were one or two more parameters, the number of moves would increase only to 27 or 81. Again, it is significant that, unlike with most applications, the number of moves does not increase with the size of the problem.

The results show that the tabu searches are very effective, delivering a near optimal strategy that a classifier can use repeatedly with near minimum expected long run inspection costs.

2 The Problem

Classification of objects has been an area of extensive interest in recent years (e.g. [1]). The question is how to design a classification system that utilizes prior statistical information in classifying objects and minimizes the expected cost of determining the class of an object.

A simple example comes from the medical field where classes can be thought of as diseases and the objects as people. Each person has a certain set of symptoms, but a cost of testing (actual dollars, time, pain) is associated with each symptom. In addition, we are provided with statistical information about how often each symptom occurs in the population. Then the problem is to determine for which symptoms to test first to classify a person's disease with the smallest expected cost.

There are many applications of classification techniques. One application of particular interest to US West is fault diagnosis in communication networks. In this case, a system failure may be known to lie along one or more paths in the network; each component of the path has an associated cost of testing and a probability of failure. An efficient classification scheme gives a sequence of components to test in order to diagnose the system failure while minimizing the expected inspection cost. In the design of a classification strategy, we assume that failure of components is statistically independent. This problem is easily generalized to failure diagnosis for general networks.

Another application important to US West and many companies is in the analysis of customer service data. Initially, a questionnaire is given to a test population with perhaps 100 questions (or any unreasonably large number of questions for the general public). In addition, the test population is requested to rate their satisfaction with their service on a scale from (say) A to F. Here the classes correspond to the grades A through F and the associated vectors are the question responses, i.e., each class consists of many different response vectors summarizing customer covariates (sex, age, etc.) and experiences (ease of reaching a US West representative, timeliness of response, etc.). An efficient classification scheme would give information for the design of a smaller set of questions whose answers

determine customer satisfaction. The goal is to identify the most important factors for predicting (classifying) customer satisfaction.

Additional applications are in discriminant analysis of test data, research and development planning (e.g. in allocation of limited funds among high risk projects), in speech/voice recognition (i.e. in classification of pattern vectors), in distributed computing, in the design of interactive expert systems, in reliability analysis of coherent systems, and in many other areas.

Formally, the problem can be stated as follows: We are given

1. a set of classes $C_1 = [c_{11}, c_{12}, \ldots, c_{1n}], C_2 = [c_{21}, c_{22}, \ldots, c_{2n}], \ldots, C_m = [c_{m1}, c_{m2}, \ldots, c_{mn}]$, each characterized by the value c_{ij} of attribute j required for membership in class C_i,

2. prior statistical information $p_{kj} = Pr(a_j = k)$, the probability that attribute j has value k, that gives relative frequencies of attribute values in the population, where the attribute values are assumed to be statistically independent, and

3. costs c_j for observations to determine the value of attribute j.

The classification problem is to design an efficient classification system to acquire relevant facts and to classify new cases while minimizing the cost of inspection. The output of the classification system is a classification tree ([1]) that represents a strategy for choosing the next attribute to test to minimize expected cost of classification.

For example, in the medical diagnosis problem, a class C_m corresponds to a diagnosis or disease, the attributes correspond to the results of various medical tests with values in some range that are indicative of the disease C_m. The cost of the test, perhaps as a function both of monetary cost and discomfort, is the testing cost and of course the associated probabilities are the probabilities that the medical tests give values in the given ranges. A classification system for the medical diagnosis problem seeks an optimum ordering of administering medical tests for diagnosis.

Although in general a classification system will be implemented with a range of attribute values for each class, for ease of formulation and experimentation we work with a simplified version of the problem. Sufficient generality for our investigations can be obtained if we consider only objects with binary valued attributes, i.e., each attribute is either present or not present. We choose the classes to have three possible values for each attribute: present,

not present or don't care. A variation on the classification problem, which we have not incorporated, is the addition of a loss function for misclassification of an object.

Before proceeding, a very important point must be emphasized. The heuristics developed in this project for this problem are *not* classifiers. Instead the heuristics develop a classifier, which has polynomial computational complexity. Thus the heuristic is used only once for a given application. The classification strategy produced by the heuristic will be used over and over by the user to classify as needed. That the classifier will be used repeatedly underscores the importance of finding a strategy with small expected (long run) cost.

3 Adaptation of Problem for Heuristics

Any solution to the above problem is a set of rules for the selection of attributes to be tested and can be represented as a decision tree, i.e., if attributes $a_1, a_2, ..., a_k$ have been tested, then we decide which attribute to test next, taking into consideration which classes have been eliminated so far. Thus a node of the decision tree may be represented by the current set of (non-eliminated) classes and of the untested attributes together with a rule that dictates the attribute to test next. Based on the outcome of the test, the set of classes is further reduced, and the set of untested attributes decreases by one. The cost of testing this attribute is incurred *unless* all of the remaining classes do not care what the value of this attribute is, in which case it will not actually be tested.

Let $Cost(R, C, A)$ be the expected cost of applying the attribute selection rule R to the set of classes C and attributes A. Let $R(C, A)$ denote the attribute selected by R in this situation. Let $c^*(a, C)$ be 0 if attribute a is at the don't care level in each of the classes of C. Otherwise let $c^*(a, C)$ be the cost of testing attribute a. Let $C_{in}(a)$ be the set of classes in C that have attribute a, and let $C_{out}(a)$ be the set of classes in C that do not have attribute a. Finally, let p_a denote the probability that attribute a occurs in the population.

Then $Cost(R, C, A)$ can be expressed recursively as follows:

$$Cost(R, C, A) = c^*(R(C, A), C) + p_{R(C,A)} Cost(R, C_{in}(R(C, A)), A \setminus R(C, A))$$

$$+(1 - p_{R(C,A)}) Cost(R, C_{out}(R(C, A)), A \setminus R(C, A)).$$

This is the means of evaluation that we use in our heuristics. We can approximate this cost by testing a random selection of population members and observing the cost of classifying them.

If the sample is large enough, the observed average cost will approximate the expected cost. We ran some experiments to determine what "large enough" is and when approximation of the expected cost might be faster than calculation of the exact expected cost. We found that the sample must be greater than the product of the number of classes and the number of attributes, and for all of our problems, it is faster to calculate the exact expected cost.

Although we know the structure of a solution and how to evaluate its cost, the number of decision trees that must be investigated is vast. To count the number of decision trees for a problem with n attributes that take on m values, we first observe that any of the n attributes can be selected as the first one to test. Suppose attribute a_j has been chosen as the first node of the tree. Then for each of the m possible values of a_j, there are $n-1$ attributes left to test. Continuing, at the kth level there are m^k nodes and for each there are $n-k$ attributes left to test. Thus the number of decision trees for a problem with n attributes that take on values in a set of size m is

$$n(n-1)^{m^1}(n-2)^{m^2}\ldots(n-k)^{m^k}\ldots(n-(n-1))^{m^{n-1}}$$

$$= n[(n-1)!]^m[(n-2)!]^{m^2-m}\ldots[(n-k)!]^{m^k-m^{k-1}}\ldots[(n-(n-1))!]^{m^{n-1}-m^{n-2}}.$$

The rapid growth of the factorial function is well known. Here, for just seven binary-valued attributes, the number of decision trees already exceeds 10^{24}. Clearly exhaustive search is completely impractical and heuristics must be carefully designed to overcome the combinatorial explosion inherent in the problem.

To overcome this difficulty, we developed a *generic decision rule* to determine what attribute to test next under the given conditions. Then we can store the rule and generate the tree as we go. We have identified three values to use in our rule:

1. *Cost* to test a given attribute,

2. *Probability* that a given attribute occurs in the population of objects, and

3. *Proportion* of untested classes in which a given attribute is required to be present.

The proportion is recalculated every time a class is eliminated from consideration. The value of the proportion gives information about the potential for reducing the subset of possible classes to which an object may belong if a given attribute is tested.

Using the above values, we developed the following generic decision rule: Choose the next attribute to test to be the one that minimizes

$$(cost)^{\alpha}(|0.5 - probability|)^{\beta}(|0.5 - proportion|)^{\gamma},$$

where the minimization is over all untested attributes. The parameters α, β, and γ are determined heuristically. This rule is dynamic due to the proportion factor.

Note that while our generic decision rule can represent many possible decision trees, it does not represent all trees. There are some decision trees that cannot be expressed in this form. However, our results show that the rule performs very well.

Since the function $x^{e_1}y^{e_2}z^{e_3} = (xy^{e_2/e_1}z^{e_3/e_1})^{e_1}$, minimization of this function is equivalent to minimization of the function $xy^{e_2/e_1}z^{e_3/e_1}$. This implies that a generic decision rule that selects attributes sequentially by minimization of such a function gives rise to a whole family of such rules. Thus in the implementation of our algorithms, we actually fix the exponent α to be 1 in the generic decision rule and search for the appropriate values of the exponents β and γ. This is necessary in order to prevent possible cycling in the heuristics.

4 Heuristics

To obtain a standard set of problems for comparison of our heuristics, we developed a problem generator to generate random problems of various sizes. The test bed of problems consists of a set of small problems and a set of larger problems, whose sizes are shown in Table 1. The test bed we have used consists of problems from three categories.

1. The first category is a set of problems with costs ranging from 1 to 300, and probabilities uniformly distributed in the interval (0,1). [TPROB1–7 and TPROB101–107]

2. The second category is a set of problems with costs ranging from 1 to 30, and probabilities uniformly distributed in the interval (0,1). [TPROB8–14 and TPROB108–114]

3. The third category is a set of problems with costs ranging from 1 to 30, and probabilities uniformly distributed in the interval (.3,.7). [TPROB15–21 and TPROB115–121]

Problems	# classes	# atts.
TPROB1,TPROB8,TPROB15	6	4
TPROB2,TPROB9,TPROB16	8	4
TPROB3,TPROB10,TPROB17	10	5
TPROB4,TPROB11,TPROB18	10	5
TPROB5,TPROB12,TPROB19	15	5
TPROB6,TPROB13,TPROB20	20	6
TPROB7,TPROB14,TPROB21	25	6
TPROB101,TPROB108,TPROB115	50	10
TPROB102,TPROB109,TPROB116	60	15
TPROB103,TPROB110,TPROB117	75	20
TPROB104,TPROB111,TPROB118	75	20
TPROB105,TPROB112,TPROB119	90	25
TPROB106,TPROB113,TPROB120	95	25
TPROB107,TPROB114,TPROB121	100	30

Table 1

4.1 Dynamic Programming Algorithm

For small problems, we can find the true optimal decision tree using a dynamic programming algorithm. Readers unfamiliar with the dynamic programming technique are referred to [6]. The algorithm finds the optimal selection strategy R^* by determination of the optimal decision to make in each possible circumstance. This is accomplished by the formula:

$$R^*(C, A) = \arg\min_{a \in A} \left(c^*(a, C) + p_a Cost(R^*, C_{in}(a), A \setminus a) + (1 - p_a) Cost(R^*, C_{out}(a), A \setminus a)\right).$$

In other words, we find a decision tree that has minimum expected cost among all correct decision trees for the problem data. As is usual with dynamic programming algorithms, to avoid repetition we solve all cases with small C and A first and work up to the solution for the set of all classes and attributes. We can then backtrack from the topmost solution to extract the optimal tree.

Results from the dynamic programming algorithm are presented in Table 2 and in Table 3 for comparison with the heuristics. These results are only available for the smallest problems due to the combinatorially explosive nature of the algorithm.

4.2 Permutations

Although a general solution to a classification problem is a decision tree, a decision rule that simply chooses the attributes to be tested in a predecided order may actually be a good solution. Such a solution corresponds to a permutation of the attributes. Although the number of permutations is quite large when the number of attributes if large ($k!$ if there are k attributes), the set of all permutations of the attributes is a subset of the vastly larger set of all decision trees. It is still possible that for most problems, searching permutations would give near optimal solutions. Before testing our generic decision rule, we decided to experiment with permutations.

For small problems (six or fewer attributes) we generated all permutations of attributes, and for each permutation we calculated the cost of testing the attributes in that order. For problems too large to generate all permutations, we looked at a random sample of permutations. The results of our tests are presented with the random search results in Tables 3 and 4, where they can be compared with the results from other heuristics. The results of these tests indicate that a decision rule that corresponds to a permutation of attributes is not, in general, a good solution to this problem. On occasion the optimal tree will actually correspond to a permutation-decision rule, but this case is rare. This observation indicates that an effective heuristic search strategy cannot reduce the search space to the set of permutations.

4.3 Tabu Search

A hillclimbing algorithm is a heuristic algorithm for optimization that moves from one element of the solution space to another by choosing the optimal improving move at each stage. It terminates when no improving move can be found. Hillclimbing algorithms are well known for obtaining good solutions to problems that suffer from combinatorial explosion, while having the propensity to become stuck at local optima. A tabu search algorithm investigates the same neighborhood as the hillclimber, but tabu search chooses the best move in the neighborhood, regardless of whether it is an improvement over the current solution. To avoid returning immediately to a point already visited, we keep a list of moves that are "tabu". The tabu search terminates after an input number of iterations (generally 50 iterations works well) and reports the best solution that was found. For a more detailed

description of tabu search, see [4] or [5]. The inputs to our implementation of tabu search for the classification problem are the number of iterations, the length of the tabu list, the step size, and the starting values for the parameters.

In general a tabu list is a list of move directions. If a move is made in a certain direction, then the reverse of that direction is made tabu. Since our problem only requires varying three parameters, it is feasible for us to keep a list of exact points that cannot be revisited. This is less restrictive than making the move directions tabu. However, it turns out that keeping the more restrictive list of move directions works better for this problem.

The generic decision rule that was presented in Section 2.3 actually evolved from testing many rule variations. We initially started with a very basic form of the generic decision rule:

$$(cost)^{\alpha}(probability)^{\beta}(proportion)^{\gamma}$$

where cost, probability and proportion have the same meaning as in Section 2.3. The results from this rule are shown in Table 2, under the heading "Tabexv". The next column of the table are the results obtained by using the same form of the rule, but calculating the proportion by disregarding the "don't care" values (represented by twos) in the classes. Another variation was to use the basic rule and add a fourth value called *numtwos*, that counts the number of "don't care" values for each attribute. The idea is that if an attribute has a "don't care" value in many of the classes, then we might not gain much information by testing this attribute. This form of the rule is:

$$(cost)^{\alpha}(probability)^{\beta}(proportion)^{\gamma}(numtwos)^{\lambda}.$$

These results are presented under the heading "Twotabu2". The next column in the table with heading "Halftabu" shows the results for the rule that was given in Section 2.3. Another variation is to use the *relative cost* of testing an attribute instead of the actual cost, where relative cost is calculated as the cost of testing the attribute divided by the total cost of testing the remaining attributes. This idea was incorporated in conjunction with measuring the probability and proportion distances from 0.5 to give the following form of the rule:

$$(relative\,cost)^{\alpha}(|0.5 - probability|)^{\beta}(|0.5 - proportion|)^{\gamma}.$$

These results are reported under the heading "Chtabu". Finally, we tried a composite rule that combines the previous rule and calculation of the proportion by disregarding the "don't care" values in the classes. These results are reported under the heading "Combtabu".

Problem	Optimal	Tabexv	Twotabu1	Twotabu2	Halftabu	Chtabu	Combtabu
TPROB1	545.08	556.428	556.428	556.428	558.555	558.555	558.555
TPROB2	736.987	765.128	736.987	736.987	736.987	736.987	736.987
TPROB3	516.08	521.195	521.886	521.195	521.67	521.67	521.67
TPROB4	557.52	561.319	600.625	605.223	561.319	561.319	560.96
TPROB5	433.22	438.543	438.543	438.543	433.22	433.22	433.22
TPROB6		533.27	536.709	542.449	537.873	537.873	540.811
TPROB7		536.043	536.043	536.043	528.53	528.53	534.333
TPROB8	49.778	50.435	50.435	53.378	50.46	50.46	53.767
Total	2838.665	2893.048	2904.904	2911.754	2862.211	2862.211	2865.159

Table 2

The optimal solutions in Table 2 are those found by the dynamic programming algorithm. Problems 6 and 7 are too big to execute this algorithm in a reasonable amount of time. The totals given are for Problems 1 through 5 and Problem 8. Halftabu and Chtabu tie for the lowest total, but Halftabu requires less computation. All other tabu search results reported in this paper use the rule from Halftabu.

4.4 Random Search

We could not run the dynamic programming algorithm on large problems, so we needed some other means to evaluate the tabu search. We did this with two types of random searches.

One version of the random search for this problem is to randomly choose the parameters α, β, and γ. This allows us to evaluate how well the tabu search searches the solution space.

The other version of the random search uses random trees. In this algorithm, each time we need to choose the attribute to test next, we choose that attribute randomly. The result is that we are generating decision trees randomly. As was mentioned earlier, our rule does not represent all possible trees. By looking at random trees, we are able to evaluate whether the set of trees represented by our rule is sufficient to find a good solution.

We ran each version of the random search ten times on each problem and calculated the best solution found as well as the average solution. The tabu search results should be compared against the average solution, since that is what we would expect for any given run of the random search. However, the best solution is useful because it gives an indication of the potential of each method.

Problem	Optimal	All Perms	Random Trees Best	Random Trees Avg.	Random Param. Best	Random Param. Avg.	Tabu
TPROB1	545.08	575.2	545.08	546.05	558.555	558.917	558.555
TPROB2	736.987	765.1	736.987	736.987	736.987	736.987	736.987
TPROB3	516.08	521.2	516.3	517.05	520.815	521.067	521.67
TPROB4	557.52	566.3	557.75	559.64	561.319	561.445	561.319
TPROB5	433.22	466.7	436.6	437.6	433.22	433.22	433.22
TPROB6		537.9	531.8	551.2	537.87	537.87	537.873
TPROB7		539.6	533.95	584.9	528.53	528.53	528.53
TPROB8	49.778	50.4	49.78	49.78	50.435	50.435	50.46
Total	2838.66	2944.9	2842.50	2847.11	2861.33	2862.07	2862.21

Table 3

We report results in two categories: small problems and large problems. The sizes of the small problems range from 6 classes with 4 attributes to 25 classes with 6 attributes. The sizes of the large problems range from 50 classes with 10 attributes to 100 classes with 30 attributes. The optimal solutions found by the dynamic programming algorithm are given for the problems with five or fewer attributes. We solve the small problems in under one and one-half minutes each, taking just a few seconds for the smallest ones. The execution times for the larger problems vary with the size of the problem from five minutes to two and one-half hours. These running times are for an implementation in PASCAL on a VAX 8800.

The results for the small problems are presented in Table 3. The totals given are for Problems 1 through 5 and Problem 8. The random searches on trees and parameters each tested 450 solutions. All permutations were examined for each problem. We see that though occasionally a permutation gives a good solution, this is not true in general. For a problem with 4 attributes there are only 576 possible trees, so it is not too surprising that we do well by choosing trees randomly for these small problems. As the size of the problem grows, we expect the solutions found by randomly searching trees to become less competitive with those of tabu search.

The results for the large problems are presented in Table 4. The random searches were executed for a set number of iterations chosen so that all solutions are found in comparable time. As expected, the solutions found by randomly generating trees have significantly higher cost than the tabu search solutions, as do the solutions found with random permutations.

Problem	Random Perms Best	Random Perms Avg.	Random Trees Best	Random Trees Avg.	Random Param. Best	Random Param. Avg.	Tabu
TPROB101	1389	1412	1261	1305	1153.9	1179.7	1143.34
TPROB102	925	1013	747	780	479.9	481.4	475.25
TPROB103	1132	1317	977	977	527.14	550.4	522.27
TPROB104	639	721	641	641	297.6	307.9	302.72
TPROB105	1535	1651	1214	1334	940.7	948.4	938.5
TPROB106					241.9	243.7	237.5
TPROB107					307.8	315.4	306.16
TPROB108					41.14	41.19	41.14
TPROB109					77.3	79.3	77.6
TPROB110					78.1	81.1	80.7
TPROB111					61.4	62.6	62.2
TPROB112					43.3	43.6	43.24
TPROB113					40.6	41.2	40.5
TPROB114					38.1	38.7	38.1
TPROB115					87.07	87.35	86.99
TPROB116					73.26	73.63	73.36
TPROB117					52.36	52.71	52.15
TPROB118					43.81	44.06	43.45
TPROB119					41.95	42.32	42.24
TPROB120					50.85	51.36	50.61
TPROB121					43.08	43.37	42.93

Table 4

Furthermore, the solutions found by randomly choosing parameters for our rule are also much better than the randomly generated trees and permutations. For these reasons, we collected data for random trees and permutations only for five problems.

Now compare the tabu search results to the random parameter results. The tabu search beats the average random solution on every problem. So the tabu search is indeed searching the solution space intelligently. This claim is further substantiated by noting that on many of the problems the tabu search finds a better solution than ten runs of the random search, and in one-tenth of the time.

5 Summary

The tabu search approaches are very successful. They are accurate and deliver the near optimal rule in very little time. When assessing the running time of the heuristic, it is important to note that this algorithm is only run once for a given application. The rule returned by the algorithm is then used repeatedly by the user for classification.

We are currently developing some integrated heuristics for this problem. We are taking our most successful heuristics and combining them, striving for even better results.

6 References

1. L. Breimann, J. Friedman, R. Olsen, and C. Stone, *Classification and Regression Trees*, Wadsworth Statistics/Probability Series, 1984.

2. L.A. Cox Jr., Y. Qiu, and W. Kuehner, Heuristic Least-Cost Computation of Discrete Classification Functions with Uncertain Argument Values, *Annals of Operations Research* **21** (1989), 1–30.

3. L.A. Cox Jr., Incorporating Statistical Information into Expert Classification Systems to Reduce Classification Costs, *Annals of Mathematics and Artificial Intelligence* **2** (1990) 93–108.

4. F. Glover, Tabu Search Part I, *ORSA Journal on Computing*, Vol. 1, #3, 190–206, 1989.

5. F. Glover and M. Laguna, Tabu search, *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves ed., Blackwell Scientific Publications, Oxford, 1993, 70–150.

6. G. Nemhauser, *Introduction to Dynamic Programming*, Wiley, New York, 1966.