An Evaluation of Feature Selection Methods
and Their Application to Computer Security

Justin Doak

CSE-92-18

An Evaluation of Feature Selection Methods
and Their Application to Computer Security

Justin Doak
Graduate Group in Computer Science
University of California at Davis

Abstract

The growing concern over the security of computer installations has led to the recent development of numerous intrusion detection systems. *All* of these systems rely on features of user and system behaviour to determine the likelihood of an attack. The choice of these features is somewhat arbitrary and is based solely on the opinion of an expert. This work surveys the existing field of feature selection in an attempt to discover efficient algorithms, both search procedures and evaluation functions, which select effective features for intrusion detection systems. In addition, a new search algorithm, random generation plus sequential selection, is presented and compared to existing algorithms. Experiments show that random generation plus sequential selection, backward sequential selection, and beam search are the best search procedures to use in feature selection for intrusion detection systems. The experiments also showed that error rate evaluation functions are far superior to other methods of evaluating feature subsets. A final and surprising result is that backward sequential selection, although examining only a small portion of the search space, always found small feature subsets of high predictive ability and is clearly the best overall search algorithm.

Contents

List of Figures

List of Tables

I. Introduction

Along with the current realization of the importance of computer security, a large number of intrusion detection systems (IDSs) have been recently developed [1-5]. An IDS analyzes the audit trails produced by the computer system(s) it is monitoring in an attempt to recognize patterns of intrusive behaviour. One type of pattern which an IDS might look for is that of a doorknob attack where an attacker attempts to gain illegal access to a computer system by entering numerous common login / password combinations. An IDS could detect this type of attack by noting that a large number of audit records indicating failed login attempts are being produced.

All of the current IDSs rely on features of user and system behaviour (e.g., number of files deleted between login and logout) as input to their analysis algorithms which then determine the likelihood of an attack. The choice of these features is somewhat arbitrary and is based solely on the experience of an expert. *We propose a method for selecting features to be used in IDSs that is based upon the* ***experimentally*** *derived effectiveness of the features at classifying users as attackers or non-attackers.* In turn, this technique will improve the usefulness of current IDSs. Additionally, the complexity of features used in IDSs may permit the application of this algorithm to many other domains, such as medicine[1], where features of equal or less complexity are present.

The development of an efficient feature selection algorithm to be used in an IDS involves the implementation and testing of existing feature selection methods. In addition, a new feature selection method is implemented and its performance is compared to that of existing algorithms. The results are presented in a survey format since the work is comprehensive when feature selection is viewed as a search problem.

The primary goal of this work is the following:

- *discover an efficient feature selection algorithm which allows designers of IDSs to determine the best features to monitor in their domain.*

We are supplying a tool for designers of IDSs. A designer will need to develop a database of attack data and normal data to be used as input to the feature selection system. The system will then determine which of the features present in the attack database are the most valuable at distinguishing attackers from non-attackers. These are the features which should be monitored by the IDS.

A sub-goal of this work is to

- *test existing feature selection search algorithms*

where a search algorithm determines which feature subsets should be evaluated in the search space. An entire section is devoted to search algorithms later in this document.

Also motivating this work is the desire to

- *develop a new search algorithm with improved performance, at least in particular domains, over existing search algorithms.*

We are particularly interested in engineering a search algorithm that is more effective in the domain of intrusion detection.

---

[1] A heart disease database was used as one of the testbeds for the feature selection algorithms. The features contained in this database were extremely complex in nature consisting of binary, ordinal, and categorical features.

An important component of any feature selection algorithm is the evaluation function. The evaluation function determines the value of feature subsets and serves to guide the search algorithm. This leads to an additional goal which is to

- *test existing evaluation functions*

for both time complexity and accuracy. Again, an entire section is later devoted to this topic.

Finally, we

- *determine the best combinations of search algorithm and evaluation function*

by experimenting with numerous pairings.

Section II provides the necessary background on feature selection and intrusion detection. Since we are viewing feature selection in the context of a state space search problem, search algorithms and evaluation functions are needed, and they are discussed in Sections III and IV, respectively. The databases used in testing the algorithms are described in Section V. They include three simulated or randomly generated databases and three machine learning datasets (i.e., datasets used to test the performance of machine learning algorithms). Section VI presents the experimental results and discusses reasonable parameter selection in those search algorithms which are parameterized (e.g., genetic search requires that the user specify numerous parameters including the population size). The conclusions which can be drawn from this work, in light of the goals, are presented in Section VII, and Section VIII provides a discussion on possible directions for future work.

II. Background

This section begins with a discussion of where feature selection belongs when viewed from the broad perspective of machine learning. This is followed by a discussion of the framework of feature selection, particularly when viewed as a search problem. Open problems in the field of feature selection are presented next with the emphasis on those problems which this work is primarily addressing. Finally, we introduce intrusion detection, including its motivation, a general model, and the application of feature selection.

II.A Machine Learning

Machine learning can be grouped into the following categories:

- concept learning
- clustering
- prediction
- feature selection

In concept learning, one is given a training and a test set of examples where each example includes feature values and its classification. For example, if the concept one is trying to learn is (X or Y), then X=1, Y=0 is a positive example and X=0, Y=0 is a negative example. The learning algorithm analyzes the training set and attempts to uncover a concept which accurately describes the examples it sees. We determine the accuracy of the learning algorithm by calculating the percentage of examples from the test set which were incorrectly classified using the learned concept (i.e., its error rate). A classic example of a concept learner is the ID3 decision tree algorithm [38].

Clustering differs from concept learning in that the examples used in the learning process are *not* given a classification. Reverting to the examples of the previous paragraph, (X=1, Y=0) and (X=0, Y=0) are instances, but no indication of their class is given. The job of the clustering algorithm is to group together examples from the training set which are similar, thereby discovering the classes. Then, the algorithm can be evaluated on a test set to see if new instances are placed in the correct class. An example of an effective clustering algorithm is presented in [39].

Another sub-field of machine learning, called prediction, involves determining what is most likely to follow given what has been seen in the past. For instance, suppose a prediction algorithm has been given the following set of symbols:

$ @ @ $ @ @ @ $ @ @ @ @ $ @

and is asked to guess what the next symbol is most likely to be. In this case, a good prediction algorithm would say that '@' is the most likely candidate for the next character. A detailed description of a prediction algorithm can be found in [40].

The majority of the work in machine learning has focused on concept learning. This has caused a large gap between the sophistication of techniques used in concept learning and the other three sub-fields, particularly feature selection. However, the most sophisticated of concept learning methods can only do as well as the features it is given. When one also notes that clustering and prediction depend on the features they use to perform effectively, the need for research on feature selection is apparent.

II.B Feature Selection

One can view feature selection as a search through the space of all possible combinations of features [6] in an attempt to find the subset of features which will provide the best classification performance (e.g., provides the smallest error when distinguishing between

attacks and non-attacks). A state in this space is a specific instance of one of these various feature combinations. We can represent a state by a binary vector whose length is the total number of features. A '1' entry represents a feature's inclusion in a state and a '0' represents its exclusion. Fig. 1 shows the search space for three features where the vector <1,0,1> represents the state in which features one and three are included and feature two is excluded.

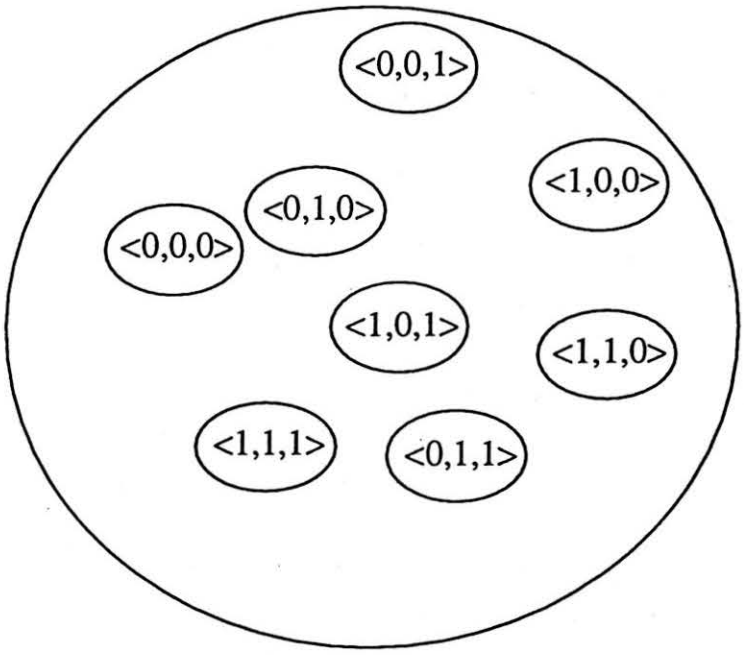

Fig. 1. Search space for three features.

For the search, we need an evaluation function which tells us the effectiveness of a particular subset of features and an algorithm to guide the search. A heuristic search[2] is usually necessary since the size of the state space being searched is 2^n where n is the number of features (i.e., a feature is either included in the set of features comprising a state or it is not). The next two sections present various search algorithms and evaluation functions.

Open Problems in Feature Selection

There are two major open problems in feature selection, as well as a number of less significant ones. We outline them below.

> *Problem 1*: Quickly find an optimal or near-optimal subset of features given an initial set of features.

Given a certain number of features, n, the task is to choose the subset of k features (k = 1, ..., n) which give the optimal performance. If the features are being chosen for a classifier (i.e., a concept learner), the subset of features which provides the lowest error rate on the

---

[2] A heuristic search is also referred to as a sub-optimal search since it does not examime every state in the search space and therefore may not find the optimal solution.

classification task is considered as the optimal. This has two important results. First, by choosing a subset of features which is smaller than the initial set, we reduce the complexity of the task at hand. (There is always some complexity associated with the extraction of a feature and its use in the given task.) Second, it is known that removing irrelevant or noisy features from data can *decrease* the error rate of a classification algorithm.

In intrusion detection, this problem translates into finding the best features to extract from the audit trail[3] given all the information which can be audited by the system. Reducing the number of features that need to be monitored is extremely important since auditing can be a significant burden (say 20% of the CPU time) on the system. Intrusion detection is also a classification task since it attempts to distinguish between attack and non-attack behaviour. This implies that it may be wise to use an error rate evaluation function when selecting features for an IDS. By removing irrelevant or noisy features, we should be able to improve the error rate of the IDS.

Problem 2: Automatically derive features which improve the classification task.

The input to an algorithm which attempts to solve this problem consists of both positive and negative examples of the target concept (i.e., the concept we are trying to learn) with the examples containing *all* available descriptive information. In other words, there has not been any filtering of the data to create an initial set of features so that the algorithm must *derive* the most effective features. Note how this differs from Problem 1 where we are selecting the best subset of features *given* an initial set of features. In Problem 2, we do not have this pre-filtering of the data yet we are still required to find a set of relevant features. We can thus see how Problem 2 is a generalization of Problem 1. A data analysis technique might be one method of solving this problem.

For an IDS, this translates into feeding the algorithm with unfiltered audit data consisting of both attack and non-attack behaviour. The algorithm then determines the features it can derive from the audit data which are the most predictive of an attack. Compare this to Problem 1 where we begin with an initial set of features, selected by experts as relevant to detecting attacks, and then we choose the optimal subset from this initial set. Problem 2, on the other hand, derives the optimal subset of features from all available information in the audit trail.

There are other less significant problems in feature selection that need to be addressed as well.

Problem 3: How do we construct higher-level features which can improve the classification accuracy?

There may be certain ways of combining features which improve the performance of a classifier. An example of feature construction in intrusion detection might be to combine the features *login time* and *number of files deleted.* Fig. 2 illustrates this combination with a decision tree where a single node representing the new feature is created from two nodes.

---

[3] An audit trail is a file (or files) containing records which describe activity on the system. An example record might contain the name of a user, a program he executed, an input file, and other relevant information.

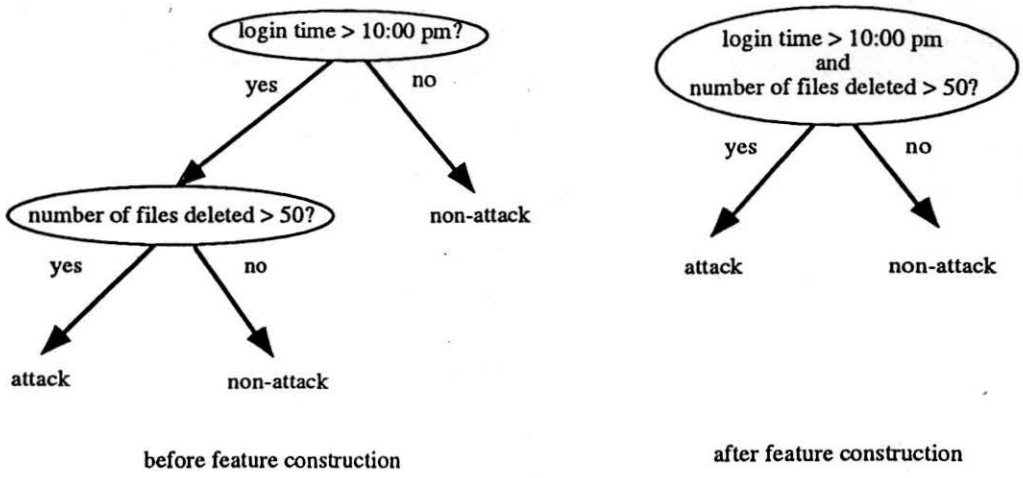

Fig. 2. Feature construction.

There has been considerable work done on this problem in the area of machine learning [19].

Problem 4: Once we have selected an optimal set of features, how do we prioritize the various features?

A feature selection algorithm gives the optimal subset of features, but it does not indicate how to attach weights or importance to the features. For instance, *failed login* and *number of browsing events* may be selected as the optimal set of features. Further, let us assume that *failed login* is a more predictive feature (i.e., is more effective) than *number of browsing events*. Our analysis algorithm should reflect this by giving a higher weight to *failed login.*

Feature threshold algorithms, as in [17], are methods which might be applied to this problem. These techniques are currently used as feature selection algorithms. As each training example is seen, weights which correspond to the features are adjusted to reflect the new information given by the example. At the end of the training, only those features whose weight is above a certain threshold are included in the final feature subset. Using such an algorithm to determine the feature weights is straightforward: use the weights determined by the algorithm as the weights of the features in the optimal feature subset.

II.C Intrusion Detection

Intrusion detection is the art of analyzing audit trails on a computer system in order to discover unauthorized behaviour, as determined by a security policy. Fig. 3 shows the process of intrusion detection. Features are extracted from the audit trails and sent to the various analysis components of the intrusion detection algorithm. These components, which are described in the following paragraph, use the features to make predictions about whether or not current behaviour is intrusive and then feeds these predictions to an expert system. The job of the expert system is to resolve the predictions made by the analysis components and produce an overall intrusion score. Note that the expert system may itself need to extract features from the audit trail in order to resolve the various predictions.

The current consensus among the intrusion detection community is that there should be two analysis components in an intrusion detection algorithm: 1) signature analysis and 2) anomaly detection. Signature analysis refers to the process of matching current behaviour

to known attack scenarios stored in a database. If current behaviour closely matches one or more of the scenarios, it is predicted with high certainty that an attack is in progress. Anomaly detection refers to how closely current behaviour of a particular user statistically matches previous behaviour for the same user. Large differences between current and historical behaviour may indicate that a legitimate user's account has been compromised or that the legitimate user is an insider threat. When resolving the predictions made by these two analysis components, more importance is given to the prediction generated by the signature analysis component. In other words, behaviour which closely matches a known attack is more indicative of intrusive activity than behaviour which is unusual.

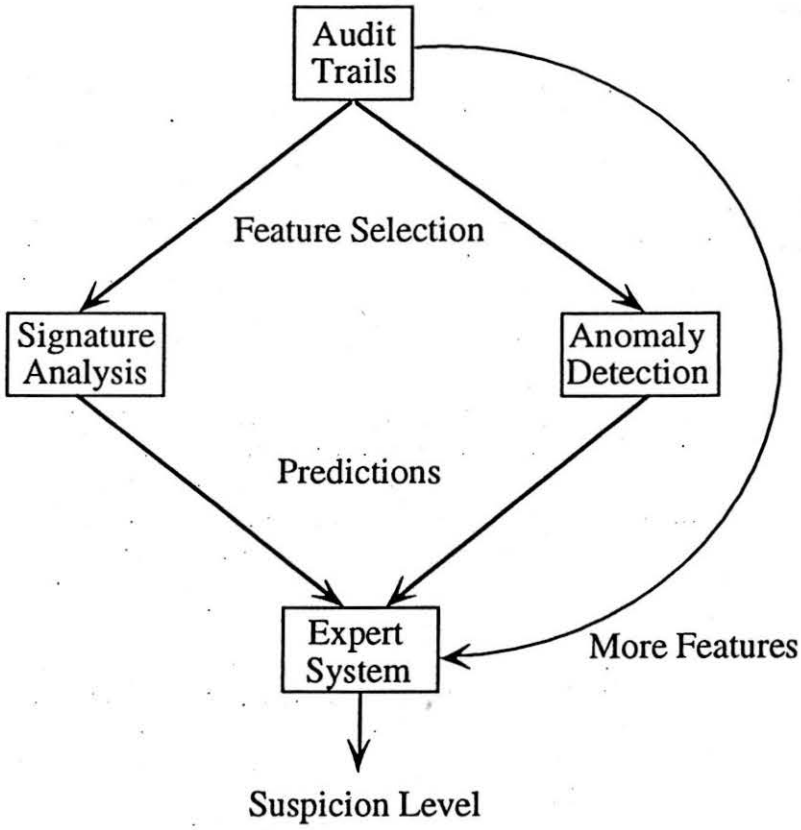

Fig. 3. A general model of intrusion detection.

II.C.1 Motivation

There are two frequently asked questions about the need for intrusion detection:

1) *Do people break into computer systems*?

2) *Why not make computer systems highly secure*?

The first question can be responded to by discussing results obtained by UC Davis' Network Security Monitor (NSM) [4]. The NSM detected approximately 400 attacks in a three month period when installed on the UC Davis Computer Science Department's Ethernet. In addition, when installed at an unnamed DOE site for three consecutive nights, it detected several attacks *each* night. The attacks discovered by the NSM provide a lower bound on the number of attacks actually occurring since the NSM only detects fairly simple

intrusion attempts such as doorknob attacks. The results obtained by the NSM are clear evidence that intrusions on computer systems are extremely frequent.

The second question, "Why not make computer systems highly secure?" can also be effectively answered. First, it is *infeasible to install a highly secure system.* Enormous amounts of time and money have been invested in current systems. The cost of their removal and the purchase of the new systems would be enormous. Making a highly secure system also *removes much of the flexibility* that helped to make the systems attractive in the first place (i.e., there is no way one can achieve high levels of security without placing severe restrictions on valid users). An example restriction is fire walling in which the computer system is completely separated from the outside world. *How do you prove that there are no holes in a secure system?* In other words, one cannot be certain that the new system will not have any security flaws. Moreover, *even a highly secure system is still vulnerable to attacks by insiders* (i.e., being able to do something is different from being allowed to do so by the security policy). For example, the super user has permission to delete all user files on his system, but is not given the right to do so by policy. As an alternative, we might decide to correct all the flaws in our current system. Unfortunately, it is *impossible to find all security flaws in a system* implying that the system would still be vulnerable to unknown forms of attack. Intrusion detection algorithms (IDAs) can be used to incrementally increase the security of a system by *finding previously unknown security flaws* through the detection of intrusions that exploit them.

II.C.2 Feature Selection in Intrusion Detection

Considering again the intrusion detection model in Fig. 3, the role of feature selection in an IDS is evident. Feature selection is used to determine which features are most beneficial in the prediction process of the analysis components. Features that do not relate to intrusive behaviour[4] will hinder the most sophisticated of analysis algorithms. The expert system may also find it necessary to extract features from the audit trail in order to resolve predictions made by the analysis algorithms. Again, feature selection can be used to determine the best features to use in this resolution stage.

Those features which prove to be useful to IDSs will in all likelihood include binary, discrete, and continuous features as well as higher-level combinations of attributes (e.g., combining the *change directory* command with the *list contents of directory* command to create a feature called *browsing behaviour*). A feature must be able to handle the fact that there will be many negative instances (i.e., most user sessions are not attacks) and few positive instances (attacks). For instance, if a feature is very good at detecting positive instances but raises many false alarms (classifying non-intrusive behaviour as intrusive), it may be less valuable than a feature that is only average at detecting attacks but raises no false alarms. In addition, it is likely that a large number of features will be necessary since some features will be better than others at detecting certain kinds of intrusions. For example, a large amount of I/O activity may be indicative of information leakage whereas a domination of CPU time may be a denial of service attack. Another indication of the need for a large number of features are the varying levels of abstraction at which features might be useful. High-level features are combinations of features, such as *browsing behaviour*, which might be useful at characterizing the general nature of a user session. Low-level features, on the other hand, are the physical characteristics of a user's keystroke dynamics

---

[4] Features can be positively or negatively correlated to intrusive behavior. For instance, a positively correlated feature might be whether or not a user attempted to modify the password file. If this feature is positive, it may indicate that the user is an attacker. A negatively correlated feature might be whether or not a user read their mail since many attackers do not read the mail for accounts into which they have broken.

and could be useful at detecting masqueraders (someone other than the legitimate user who accesses an account). This could be done by comparing the masquerader's typing behaviour to a model of the legitimate user's keystroke patterns. The model may contain 'normal' ranges for values such as overall typing speed and may also include patterns of frequently keyed words (e.g., login and password). The pattern of a word could include the time between keystrokes, the force with which keys are struck, etc.

III. Search Algorithms

A search algorithm is needed to direct the feature selection process as it explores the space of all possible combinations of features. A search procedure usually examines a small portion of the search space since this space can be enormous. When determining which state to evaluate next, a search algorithm makes use of the values of previously visited states in order to guide the feature selection engine into those regions of the search space where individual states have low error rates and few included features.

Some of the search algorithms have similar characteristics allowing us to organize them according to the following categories[5]:

- Exhaustive Algorithms
- Sequential Algorithms
- Randomized Algorithms.

The characteristics of each category are discussed and several examples from each category are presented in the remainder of this section.

III.A Exhaustive Algorithms

This class includes the following search algorithms:

- Exhaustive Search (ES)
- Branch and Bound (B&B)
- Approximate Monotonicity with Branch and Bound (AMB&B)
- Beam Search (BS).

ES is a true exhaustive algorithm since it examines every state in the search space. B&B and AMB&B, on the other hand, do not examine every state; however, they perform exhaustive search in the limit. BS is a variation on the exhaustive best-first search algorithm. In order to keep the search within reasonable bounds, a limit is placed on the memory of BS.

III.A.1 Exhaustive Search

ES naively examines every state in the search space. It is an optimal search in that it is guaranteed to find the best feature subset, as determined by the evaluation function in use, if the algorithm ever terminates. As mentioned previously, ES can only be used when the full feature set contains a small number of features, e.g., less than 20, since the size of the search space is exponential in the number of features.

The order of this algorithm is $O\left(2^n\right)$ as the size of the search space is 2^n and ES examines every state in that space. Fig. 4 shows the region of the search space examined by ES. The top of the region is the empty feature subset (i.e., the subset containing no features) and the bottom represents the full feature subset (i.e., the subset containing all the features). The search space is hierarchically organized from small feature subsets to larger ones as we move from the empty feature subset to the full feature subset. In other words, feature subsets close to the full feature subset contain more features than those subsets

---

[5] In reference [22], the various search algorithms are also divided into three categories: historical algorithms, algorithms from artificial intelligence, and monte carlo algorithms. The monte carlo algorithms directly correspond to the randomized algorithms; however, the characteristics of the first two categories do not correspond to the exhaustive or the sequential algorithms.

close to the empty feature subset. Notice that the graph is entirely shaded to signify the completeness of ES.

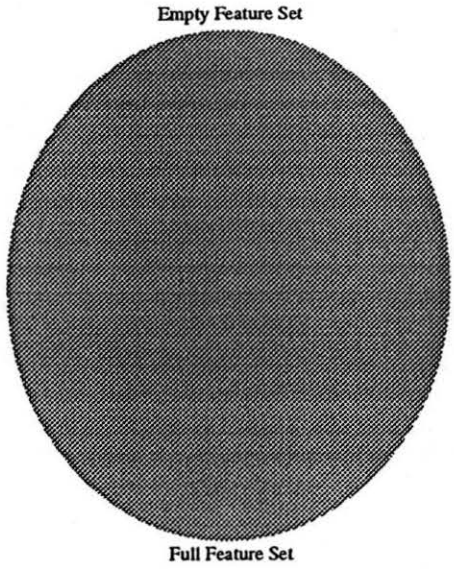

Fig. 4. Exhaustive search.

III.A.2 Branch and Bound [9]

This heuristic from the class of exhaustive algorithms is *guaranteed* to find the optimal feature set as long as the evaluation function is monotone. Monotone means that a set of a set of features will perform *no better* than any set of which it is a proper subset. In other words, if we remove any feature or combination of features from a set of features, the resulting subset performs no better and may perform worse than the original set.

There are two criteria that the B&B algorithm attempts to satisfy. The first criteria is that the optimal feature subset be as small as possible. (Recall from the discussion in Section II.B, Open Problems in Feature Selection, that removing irrelevant or noisy features not only reduces the complexity but also improves the accuracy of the learning algorithm which makes use of the features.) The second criteria is a threshold placed on the value calculated by the evaluation function. For instance, if the error rate of a classifier is used as the evaluation function, feature subsets which have an error rate greater than the threshold are discarded and are not considered for the optimal feature subset.

B&B begins its search from the full feature set. The search proceeds by removing features from this set until the error rate goes beyond the threshold amount. Because of the monotonicity property, we know that removing any more features from this set will not improve its value and the search can be cut off below this state. The algorithm now backtracks exploring other ways of removing features (analogous to depth-first search) which do not violate the threshold and which may contain fewer features.

The threshold placed on the evaluation function is B&B's primary advantage over ES. The threshold allows us to ignore, during the remainder of the search, all proper subsets of any state whose error is greater than the threshold. This greatly reduces the size of the search space and still guarantees that we will find the optimal feature subset. Disadvantages of B&B include its reliance on the monotonicity of the evaluation function. Most classifiers

do not exhibit this property and therefore cannot be reliably used as evaluation functions with a B&B search. The other disadvantage of B&B is that, although it can significantly reduce the size of the search space, its complexity is still exponential in the limit. This is due to the fact that B&B performs exhaustive search in the portion of the search space it visits.

In Fig. 5, we see that all feature subsets with an error rate less than that of the threshold are examined leading to an exhaustive search around the full feature subset. Also note that none of the feature subsets towards the empty feature subset are examined. The implications of this are that good feature subsets with few features are not likely to be found.

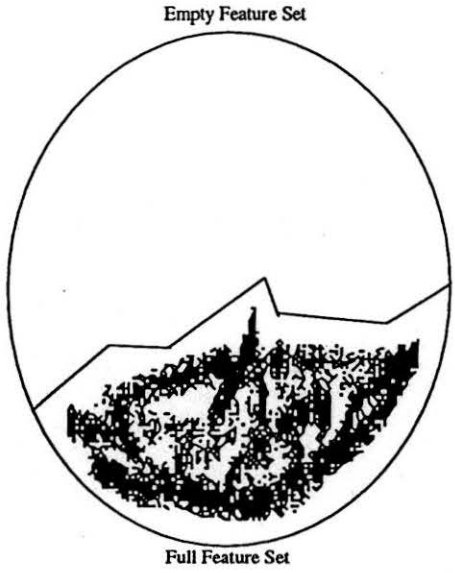

Fig. 5. Branch and bound.

III.A.3 Approximate Monotonicity with Branch and Bound

AMB&B [20] is an improvement on the classical B&B algorithm. In the classical version, the search space below (i.e., proper subsets of) feature subsets which violate the threshold of the evaluation function may be ignored. In the modified version, the algorithm is able to effectively use non-monotonic evaluation functions, particularly classifiers, since the threshold used to terminate the search is relaxed.

Let us assume that the evaluation function, represented by $E(\cdot)$, is the error rate of a classifier and that the threshold is τ. Further, let us assume that there is a tolerance Δ placed on the threshold. We classify a feature subset F as [22]

1) feasible if $E(F) \leq \tau$.

2) conditionally feasible if $\tau < E(F) \leq \tau \cdot (1 + \Delta)$.

3) infeasible if $E(F) > \tau \cdot (1 + \Delta)$.

Feature subsets in Class 1 above are within the threshold placed on the evaluation function. From amongst these feature subsets, the best feature subset is chosen to be the one with the fewest number of features. Class 2 subsets are not within the threshold, but are within the

tolerance placed on the threshold. None of these feature subsets will be chosen as the best subset since their error rates are too high; however, the search is allowed to continue from these states. (In classical B&B, there are no Class 2 feature subsets.) Those feature subsets in Class 3 are beyond the tolerance level and the search is not allowed to expand from these states.

The primary motivation for AMB&B is the fact that the performance of classifiers is frequently improved by removing irrelevant or noisy features. If one attempts to use a classifier as the evaluation function for B&B, some states with an error rate lower than the threshold may never be found. This could happen when states exist which have error rates lower than the threshold and are proper subsets of states whose error rates are greater than the threshold. In AMB&B, the search can continue beyond the original threshold so that feasible states containing fewer features may be found. For example, in Fig. 6, the first feature subset contains four features and has an error rate of 15%, feasible with both classical B&B and AMB&B. However, the state of three features has an error of 17% which is beyond the threshold of 15%. The classical algorithm will not evaluate any proper subsets of <0,1,1,0,1> due to the assumption of strict monotonicity. AMB&B, on the other hand, considers this state to be conditionally feasible since its error is within the tolerance level of 22.5%. The search is allowed to continue and a feasible set of only two features may be found.

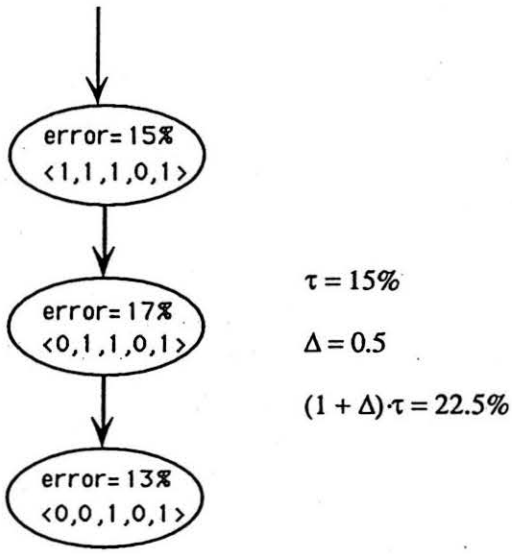

Fig. 6. AMB&B example.

AMB&B is at least as complex as B&B and is therefore an exhaustive search in the limit. Fig. 7 shows three regions corresponding to the three classes of feature subsets in an AMB&B search. The clear region contains those subsets whose value is beyond both the threshold and the tolerance level. Subsets in this region are not visited. Those subsets in the regions denoted by the thick lines are beyond the threshold but within the tolerance level. These subsets are visited but cannot be chosen as the optimal feature subset. Finally, the region closest to the full feature subset contains those subsets whose value is less than the threshold. The feature subsets in this region can be chosen as the best feature subset.

Fig. 7. Approximate monotonicity with branch and bound.

III.A.4 Beam Search

BS is a type of best-first search which uses a bounded queue to limit the scope of the search. The queue is ordered from best to worst with the best states placed at the front of the queue. The search algorithm operates by taking the state at the front of the queue, theoretically the most promising state, and extending the search from that state. Each new state which is visited is placed in its proper sorted position in the queue.

Our current implementation of BS is fundamentally the same as FSS (see III.B.1). That is, as each state is removed from the head of the queue, all possible ways of adding one feature to that feature subset are attempted and the resulting states are inserted in their proper position in the queue. For example, consider a BS with a queue size of five in a search space of three features (see Fig. 1) with the queue initialized to the empty feature subset.

tail->(0,0,0)<-head

The next states to be examined are (1,0,0), (0,1,0), and (0,0,1) after removing the empty feature subset from the queue, and they have error rates of 25%, 15%, and 27%, respectively. The resulting state of the queue becomes:

tail->(0,0,1)->(1,0,0)->(0,1,0)<-head

In FSS, only the state (0,1,0) would have been kept and the other two states discarded. Assuming that (1,0,1) is the state with minimum error, FSS will never find it since it has no way of backing up by removing the second feature. BS, on the other hand, still retains the state (1,0,0) which will lead it to the optimal state.

If there is no limit on the size of the queue, BS is an exhaustive search. However, by placing a limit on the queue size, those states with low values (e.g., high error rates) will eventually fall off the end of the queue, effectively pruning the search at those states. It is interesting to note that, when the limit on the size of the queue is one, beam search is equivalent to FSS and has a complexity of $O(n^2)$.

One way of explaining BS is that it explores promising paths in the search space until they are exhausted. For example, consider the case where BS has found a region of the search space that produces feature subsets of high value. As the search is extended from the best subset, this region of the space will be examined until all states from this region have been removed from the queue. At this point, the states with lower values will have moved from the tail of the queue to the head and will be expanded. One of these states may lead to another region containing states of high value. If so, this region will also be thoroughly examined and so on.

If one of the first regions examined contained states with relatively high values but was in reality only locally optimal, it is hoped that the search which expands from the states of smaller values will eventually lead to the global optimum. In effect, BS avoids local optima by preserving solutions from varying regions of the search space. In Fig. 8, we see how BS sends many *fingers* down into the search space as it explores various paths.

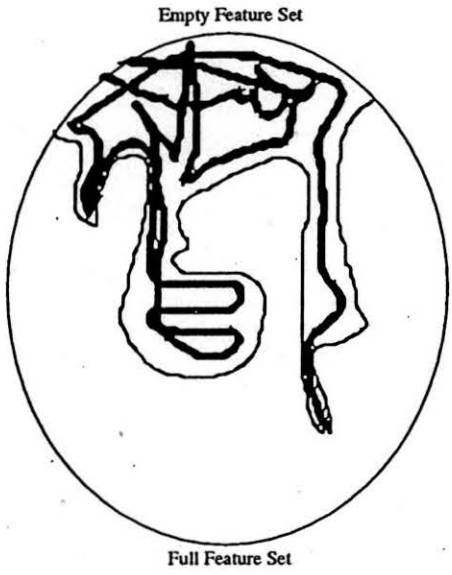

Fig. 8. Beam search.

Parameter Selection

The only parameter to BS is the maximum length of the queue. Our experiments with varying queue lengths, which were performed on all the datasets presented in Section V, indicate that a maximum queue size of at least the number of features is enough to obtain accurate results. Unfortunately, when using a slow evaluation function (e.g., a classifier that must be trained and tested for each subset), datasets with more than approximately 30 attributes could not be run with a queue size greater than ten. This was because queue sizes greater than ten caused the search algorithm to perform a near exhaustive search.

III.B Sequential Algorithms

The class of sequential algorithms includes the following search procedures:

- Forward Sequential Selection (FSS)
- Backward Sequential Selection (BSS)

- (p, q) Sequential Selection (PQSS)
- Bi-Directional Search (BDS).

All these algorithms either add or remove one or more features during each iteration. FSS adds one feature at each step whereas BSS removes exactly one feature. PQSS alternates between adding features in one step and removing features in the next step. BDS simultaneously performs two searches: one which adds a single feature at each step and another that removes a single feature.

III.B.1 Forward Sequential Selection

The algorithm for FSS begins by adding the individual feature which obtains the best performance to the empty set of best features. Next, all the remaining features are tested with this single feature to see which combination performs the best. The pair of features providing the highest predictive ability is assigned to the best set of features. Continue adding features to the feature set in this fashion until no feature increases the value of the evaluation function by at least some pre-determined threshold amount. The search stops at this point. The optimal set of features returned by this heuristic are those features contained in the best set.

The inability of this algorithm to remove features once they have been added allows features that have become obsolete, after the addition of other features, to remain in the best feature set. The algorithm is thus limited in that it does not examine every state in the search space (caused by its inability to backtrack) and may not find the best solution.

If we were to apply *FSS* to the search space for three features depicted in Fig. 1, the states <1,0,0>, <0,1,0> and <0,0,1> would be evaluated first. If <0,0,1> was determined to be the state with the highest value, given the evaluation function, the next states evaluated would be <1,0,1> and <0,1,1>. Assuming that neither of these increased the value of the evaluation function by at least the threshold amount, FSS would stop and state <0,0,1> would be selected as the optimal state. This means that the combination of features which gives the best performance, as determined by the search algorithm and evaluation function in use, is feature one by itself.

The complexity of FSS can be determined by examining the number of feature subsets evaluated each time a new feature is selected. Initially, no features have been selected and the algorithm must look at how each feature performs individually. This requires n evaluations. During the next stage, only those n-1 feature subsets resulting from adding the remaining features one by one to the current set of selected features are evaluated. Continuing in this fashion, the algorithm performs 1 + 2 + ... + n-1 + n calculations plus one calculation for the empty subset.

$$\left(\sum_{i=1}^{n} i\right) + 1 = \frac{(n+1)\cdot n}{2} + 1$$

Looking at the previous closed form expression, FSS is seen to be an $O\left(n^2\right)$ algorithm.

Fig. 9 shows that part of the search space examined by FSS, denoted by the area of the graph containing the thick lines. We can see that most of the subsets evaluated are those which contain only a few features since the search is broadest near the empty feature subset. Towards the full feature subset, the region examined by FSS narrows as the options for adding features decrease because many of the features have already been selected. It is important to understand that, although the region of the search space examined by FSS narrows as the algorithm approaches the full feature set, the number of features in the subsets is increasing. In other words, the graph refers to the states visited

by the search, not the number of features per state. Note that the graph is not to scale since the actual size of the search space examined is much smaller.

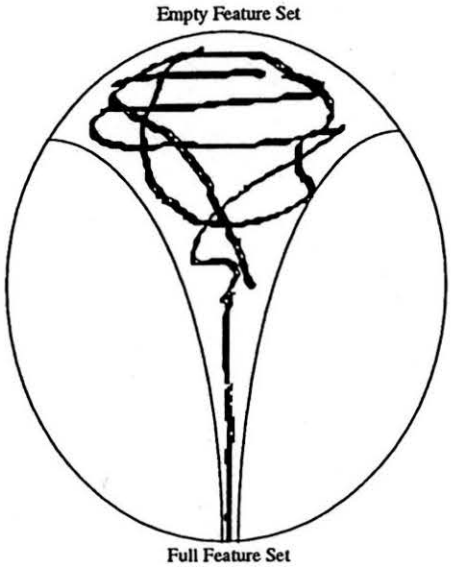

Fig. 9. Forward sequential selection.

III.B.2 Backward Sequential Selection

BSS works in the opposite manner when compared to FSS since it *removes* features instead of adding them. At each step in the algorithm, the feature whose removal results in the smallest decrease in the value of the evaluation function is removed. (In some cases, removing a feature may actually increase the value of the evaluation function. These features should clearly be removed as well.) The algorithm halts when it is impossible to remove any single remaining feature without a significant loss in performance. Its principal disadvantage is its inability to reevaluate the usefulness of a feature after it has been discarded.

The order of BSS is identical to that of FSS, $O(n^2)$. In addition, the appearance of the search space examined by BSS is similar to that examined by FSS, except that the majority of the search occurs near the full feature subset instead of near the empty feature subset (see Fig. 10).

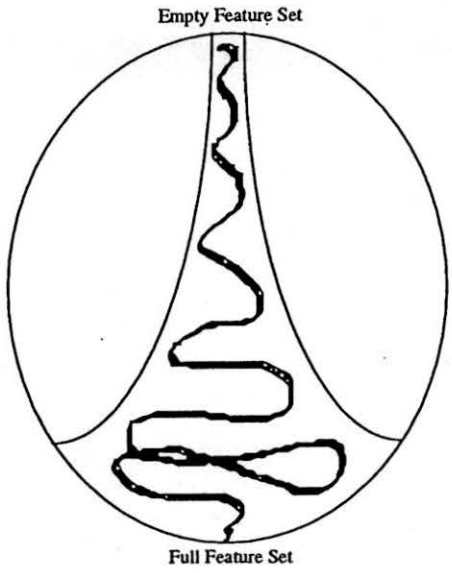

Fig. 10. Backward sequential selection.

III.B.3 (p, q) Sequential Selection

PQSS is a generalization of the forward and backward sequential selection techniques. p and q are both positive integers with p referring to the number of features to add at each step and q the number of features to remove. For instance, (2, 1) sequential selection begins by adding the best pair of features to the feature set. (The two best single features are not added, but the two features which when evaluated together give the best result are added.) This is followed by the removal of the single worst feature from the best feature set. This process is continued until the desired feature set size has been reached (i.e., the user may want to know the best feature subset of size k out of n features) or the best feature set contains all the features. Note that (0, 1) sequential selection is equivalent to BSS and (1, 0) sequential selection is equivalent to FSS.

This generalized sequential selection technique attempts to compensate for the weaknesses of FSS and BSS by allowing some degree of backtracking. In particular, it attempts to remove features that have been wrongly chosen by FSS and reconsiders features that have been unwisely discarded by BSS.

Although slightly more complex than the straightforward sequential selection techniques, the number of states evaluated by PQSS is only a constant factor larger than the number of states examined by FSS or BSS; thus, the order is the same. Fig. 11 highlights the area of the search space explored by (2, 1) sequential selection. The algorithm moves up and down in the search space as it alternates between adding and removing features. In this particular example, the search gradually approaches the full feature set since the number of features added at each step is greater than the number of features removed. If there were more features removed than added at each step, the algorithm would search from the full feature set to the empty feature set.

Fig. 11. (p, q) sequential selection.

III.B.4 Bi-Directional Search [22]

In this search heuristic, both the forward and backward sequential techniques are utilized. FSS is performed from the empty feature set and BSS is performed from the full feature set with the two searches converging in the middle of the search space. At each step of FSS, care must be taken to avoid adding a feature which has already been discarded by BSS. Conversely, BSS must never discard a feature which has already been added by FSS. If either of these conditions is broken, the two algorithms will never converge to the same feature subset in the search space.

This heuristic utilizes information from both FSS and BSS by forcing the algorithms to agree on a feature subset. Before FSS selects its next feature, it checks to see if it has been discarded by BSS. If it has, this is an indication that selecting this feature is unwise and the second best feature is selected. Before BSS removes its next feature, it checks to see if that feature has already been added by FSS and, if it has, it attempts to discard its second worst feature. Of course, if the second choice is also disallowed by the other search algorithm, the third choice is attempted and so on.

The heart of this algorithm is both BSS and FSS and consequently shares the same $O\left(n^2\right)$ complexity. We can see its resemblance to the basic sequential selection techniques in Fig. 12. The only difference is that the forward and backward searches performed as part of the BDS halt once they meet in the middle of the space.

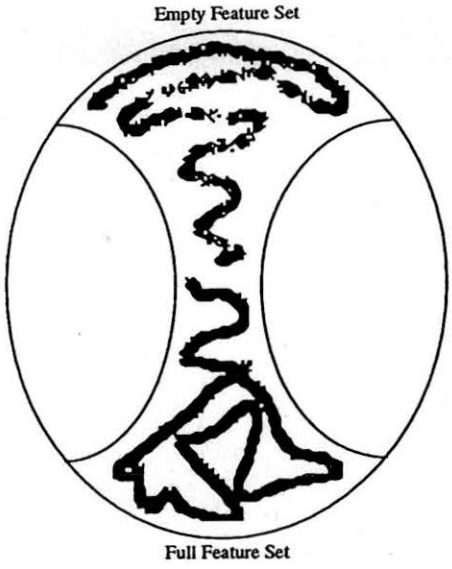

Fig. 12. Bi-directional search.

III.D Randomized Algorithms

Algorithms from this class include

- Genetic Search (GS)
- Simulated Annealing (SA)
- Random Generation plus Sequential Selection (RGSS).

The common thread linking this class of search algorithms is their use of randomness to escape local minima in the search space. For instance, if the error rate of a classifier is used as the evaluation function, we may arrive at a region of the search space where, no matter which state is visited next, the error rate does not improve. It may be that another region of the search space contains feature subsets with lower error rates, and, unless we can escape from the current region, they will never be found. Randomness allows the search to leave such local minima by periodically accepting moves to states with higher error rates. The idea is that other regions of the search space may be more fruitful and should be examined.

III.D.1 Genetic Search

Genetic algorithms are an attempt to mimic evolutionary processes in the hope of finding efficient solutions to problems. They commence by constructing an initial population consisting of candidate solutions to the problem being solved. Then, the algorithm proceeds through a number of loops, known as generations, during which new solutions are created and evaluated. Each generation begins with the construction of a mating set, usually consisting of numerous copies of the best candidate solutions from the population. Offspring are then created in pairs from the mating set, in an operation known as crossover, by combining characteristics of two members of the mating set for each pair. Next, the offspring are mutated by some random perturbation of each member (e.g., randomly adding or removing a feature from a set of features). Finally, a new population is formed either by selecting the best solutions from the population and the offspring or by selecting the best solutions from only the offspring.

The basic algorithm is as follows.

```
initialize population
for i←1 to number_of_generations do
        create a mating set from the population
        perform crossover on the mating set to create the offspring
        mutate the offspring
        create a new population from the old population and the offspring
endfor
```

Fig. 13. Genetic algorithm.

When adapting genetic algorithms to the task of feature selection, the population is initialized to contain binary vectors representing feature subsets. The implementation of GS used for the experiments randomly generates feature vectors to initialize the population. Using a random number generator, a bit of a vector is initialized to one if the corresponding random number is less than 0.5 and is initialized to zero otherwise. Other possibilities exist for the creation of an initial population which may be more effective for different experimental goals (e.g., one can bias the initial population to contain small feature subsets if one is primarily interested in finding small subsets and is not as concerned with the error rate). However, this seemed the most general and universally applicable method of performing the initialization.

After creating the initial population, the algorithm iterates through the generations. A mating set is created at the beginning of each generation which consists of numerous copies of the better feature subsets from the population. The number of copies made of a particular subset is proportional to the worth of the subset (i.e., the best subset will be more prevalent in the mating set than only a good subset).

The breeding or crossover stage of the algorithm occurs next. This stage answers the question of how to combine the various members of the mating set to produce the offspring. The implementation used here can be described by the following algorithm:

```
for i←1 to (population_size / 2) do
        select two members from the mating set
        select a crossover point
        with probability Pc, create two new members of the offspring by combining
                the members of the mating set at the crossover point.
endfor
```

Fig. 14. Crossover algorithm.

At each step through the algorithm, two members (the parents) are chosen randomly from the mating set. Next, a crossover point somewhere between one and the maximum number of features is randomly selected. If a random number (between 0.0 and 1.0) is less than

P_C, the crossover rate, the two parents are split at the crossover point and their respective sections combined to create two new members of the offspring. For example, if the two members of our mating set are chosen to be

<0,1,0,0,1> and <1,1,1,0,0>

and the crossover point is chosen as two, the two offspring are

<0,1,1,0,0> and <1,1,0,0,1>.

Crossover on the mating set is followed by the mutation operation on the offspring. The implementation used for the experiments randomly flips bits in the offspring's binary vectors with probability P_m, the mutation probability.

The final step during each generation is the creation of a new population from the old population and the offspring. The method used in our implementation selects the best members from both the old population and the offspring.

The order of this algorithm can be determined by observing that, for each generation, an offspring set is created which is then merged with the old population to form the new population. Therefore, the only new subsets which need to be evaluated during each generation are in the offspring set. In addition, an inspection of the algorithm shows that the offspring set size will be no greater than the population size. Given that g is the number of generations and p is the population size, the order is $O(g \cdot p)$.

GS can be thought of as exploring various solutions from the population in parallel. A portion of the population may be in a particular region of the search space while other groups lie in separate regions altogether. This parallel search is depicted in Fig. 15 by the numerous unconnected regions which are shaded.

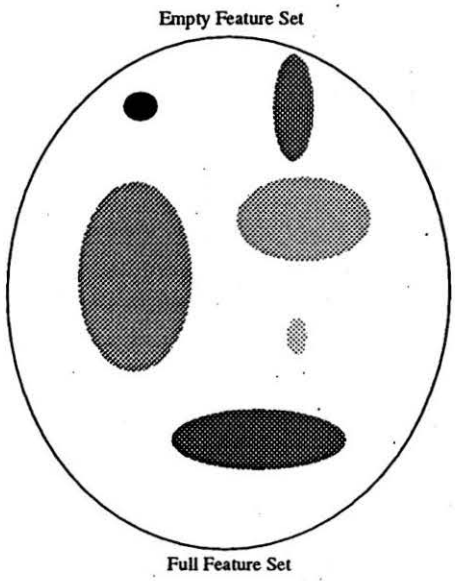

Fig. 15. Genetic search.

Parameter Selection

Our implementation of genetic feature selection requires that four parameters be input by the user[6]. The parameters are

- number of generations
- population size
- crossover rate
- mutation rate.

Optimal selection of parameters is an open problem in the field of genetic algorithms; however, some work on the subject has been performed [28-32]. Most of this work suggests that the population size is the most important choice and that optimal parameter selection is very dependent on the problem being solved. In addition, it appears that there is a connection between the selection of the number of generations and the population size. If the population size is small, good results can still be obtained if the number of generations is high. Conversely, a small number of generations can be compensated for by a large population size [41].

Reference [32] describes results from experiments which involved a meta-level search (also genetic) through the space of possible control parameters. The testbed had five optimization problems, each of which was a numerical test function. Two different optimal parameter settings were found based on two metrics used to evaluate the parameter vectors. One of the measures was obtained by taking the averaged results of performance and was referred to as the on-line measure. The other measure, the off-line measure, was the global best solution found by GS using the parameter set. Using the on-line measure, these were the best parameters found [32]:

- population size = 30
- crossover rate = .95
- mutation probability = 0.01

and for the off-line measure:

- population size = 80
- crossover rate = 0.45
- mutation probability = 0.01.

The number of generations was not a parameter to the algorithms since each one was tested by evaluating exactly 5000 solutions in the search space. The number of generations, therefore, was high for small populations and small for large populations.

Several interesting points about parameter selection for genetic algorithms were made in [32] as well. It was pointed out that smaller populations allow quicker convergence since the genetic operators of crossover and mutation are performed more rapidly. Additionally, small populations require that at least one of either the crossover rate or the mutation rate parameters be high, else high performance structures (feature subsets) quickly dominate the population. Also, it appears that the crossover rate is inversely proportional to the population size. In other words, as the population size increases, the crossover rate should be decreased for better results. (This can be seen in the two optimal settings shown above as one has a population size of 30 and a crossover rate of 0.95 whereas the other has 80

---

[6] Different implementations require different parameters. In general, the population size, crossover rate, and mutation rate are inherent in all implementations. Other parameters include the generation gap, scaling window, and selection strategy. See [32] for details.

and 0.45, respectively.) This is probably related to the fact that a high crossover rate (or high mutation rate) helps keep high performance members from dominating small populations. Another point made was that care must be taken in choosing the mutation rate since values above 0.1 performed similar to pure random search. A final comment was that, although genetic algorithms perform well given a wide range of parameter selections, poorly selected control parameters can cause premature convergence to sub-optimal solutions.

Another standard parameter vector for genetic algorithms is presented in [34]. The parameter settings are

- population size = 50
- crossover rate = 0.6
- mutation probability = 0.01.

These standard parameter settings were used to obtain a rough indication of the best parameters to a genetic algorithm performing feature selection. However, more weight was given to the results of our experiments which were specifically designed to determine good parameter settings. The experiments were run on the disjunctive normal form database, described later in Section V.A.2, with different parameters used for each experiment. Basically, all parameters except one were held constant in order to individually determine the optimal setting for each parameter. Based on these experiments, the following values for the parameters were chosen and used in the remainder of the trials:

number of generations ≥ 25

population size $\geq$ number of features

crossover rate = 0.75

mutation rate = 0.1

In general, the number of generations and population size should be as large as possible within the user's complexity constraints. It appears, however, that if these two parameters are at least as large as the limits given above, the results are acceptable. Choosing the crossover rate, since it appears to have little effect on the efficiency of the algorithm, is a much less vital decision. A general rule of thumb is that this value should be reasonably close to 1.0, otherwise the size of the offspring set will may be too small. (One may also want to heed the previous advice of using larger crossover rates for smaller populations.) The mutation rate *does* have a significant effect on the outcome of the algorithm. An optimal choice of the mutation probability will allow the search to focus on the more promising regions of the search space without becoming trapped in local minima. A large value (e.g., ≥ 0.25) of the mutation probability will not allow the search to focus in the better regions of the search space and will in fact closely resemble random search. A small value (e.g., close to 0.0) will not allow the search to escape local minima. Experimentation indicated that a value of 0.1 is a reasonable choice for this parameter.

III.D.2 Simulated Annealing

In order to understand SA, it is first helpful to understand the origin of the algorithm in physical annealing. Annealing concerns the cooling of a substance to its low temperature equilibrium. Care must be taken during the annealing process to cool slowly when approaching the equilibrium temperature or, instead of arriving at a crystalline structure with few or no defects, the resulting solid may be glassy or may even be crystalline but with many defects.

References [44-46] all concern the process of simulating annealing; however, the pioneering work on the subject is attributed to Metropolis, et.al. [33]. They produced a

model which accurately simulated the behaviour of thermal systems through the use of random variates and a probabilistic model. In the model, changes in the system are always accepted if they lead to states with lower temperatures. More significantly, a system change leading to a *higher* temperature state is accepted with a certain probability. As the system approaches equilibrium, the probability of accepting such moves decreases in proportion to the system temperature.

The analogy to feature selection is virtually straightforward. Continue to accept moves that lead to, for example, lower error rates. To avoid becoming trapped in local minima, occasionally accept moves leading to states with higher error rates. However, as the error rate approaches 0.0, decrease the probability of accepting moves to higher error rates as the search is very likely close to the global minimum.

A more detailed description of the algorithm follows [22]:

```
pick an annealing schedule
create initial solution
current_solution ← initial_solution
for i←1 to steps do
    while times_to_loop[i] ≠ 0 do
        new_solution ← transform(current_solution)
        ΔE = E(new_solution) - E(current_solution)
        accept the move with probability P = exp(-ΔE/Ti)
        if "move not accepted" then
            times_to_loop[i] ← times_to_loop[i] - 1
        else
            current_solution ← new_solution
        endif
    endwhile
endfor
```

Fig. 16. Simulated annealing algorithm.

The initialization phase of the algorithm involves picking an annealing schedule and creating an initial solution. An annealing schedule is of the form $[T_0$, times_to_loop[0]], ... , [T_{steps}, times_to_loop[steps]] where T_i is the temperature at step i and times_to_loop[i] is the number of times unaccepted moves can be generated at the ith step before going to the next step. The construction of the annealing schedule requires four parameters from the user: the number of steps in the annealing schedule, the initial temperature (T_0), a constant r used to calculate the temperatures in the annealing schedule, and the times to loop at each step. The remaining temperatures (T_1 through T_{steps}) in the annealing schedule are derived by taking $T_{i+1} = r \cdot T_i$. As in physical annealing, the

annealing schedule must be carefully constructed in order to reach optimal or near optimal solutions.

The initial solution for feature selection is the feature subset from which the search originates. The null feature subset was used in our experiments, mainly to bias the search towards small feature subsets. Other initial solutions are possible and would alter the outcome of the search.

Following the creation of an initial solution, the main loop of the program iterates through the annealing schedule. Each step of the annealing schedule is itself a loop which is controlled by the values of T_i and times_to_loop at that step in the annealing process. For instance, at step i, the temperature used to calculate the probability of accepting a move is T_i and the number of unaccepted moves which can be generated is times_to_loop[i].

Each iteration of the inner loop transforms the current solution into a new solution. The bits in the binary vector of the current solution were randomly flipped in order to perform the transformation in our experiments. Next, the difference between the error rate (ΔE) of the new solution and the current solution is calculated and used, along with the temperature at this step, to determine whether or not to accept the new solution. (Note that, if the new solution has a lower error rate than the current solution, the move is always accepted.) When the move is not accepted, the number of times to loop at this step of the annealing schedule is decremented; otherwise, the current solution is set to the new solution.

An analysis of the algorithm shows that it has a nested loop structure with the number of steps in the annealing schedule being the limit on the outer loop and the times to loop at each step the limit on the inner loop. Notice on the inner loop that times_to_loop is not decremented at each iteration, but only when a move is not accepted. This means that the algorithm may evaluate several states before generating a move that is not accepted. Nonetheless, the number of states evaluated before a move is not accepted should average to a constant and have little effect on the order analysis. If s is the number of steps in the annealing schedule and t is the number of times to loop at each step, the order of SA is $O(s \cdot t)$.

Fig. 17 demonstrates how SA escapes local minima by probabilistically accepting moves to states with higher error rates. The search first narrows to a very small region of the search space which, let us assume, corresponds to a local minima. Then, a move to a higher error rate may be accepted which breaks the search out of the local minima, possibly moving it closer to the global minimum. Again the search may find a local optima and escape to find a region of even lower error rates and so on.

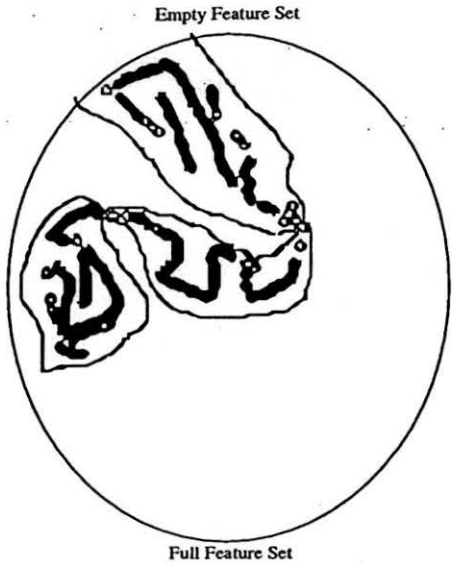

Fig. 17. Simulated annealing.

Parameter Selection

Numerous parameters and the implementation chosen for the transform operator strongly affect the performance of SA. The transform operator determines which state to visit next given the current solution. If the transform operator does not depend on the current solution, SA reverts to random search. The implementation used for our experiments flipped bits in the binary vector of the current solution with probability *mutation_probability*. In general, choice of the transform function may be the most important decision in the implementation of a SA search algorithm.

The parameters used in our experiments include

- mutation probability of the transform operator
- number of steps in the annealing schedule
- the constant r used to calculate the temperatures in the annealing schedule
- number of times to loop at each step
- initial temperature.

The number of steps in the annealing schedule is primarily an issue of complexity; the more the steps, the higher is the complexity and usually the more accurate are the results. However, the constant r ($0.0 \leq r \geq 1.0$), which is used in the construction of the annealing schedule, has a more significant effect on the search algorithm. If r is large (i.e., close to 1.0), the temperatures in the annealing schedule decrease gradually, allowing moves to states of higher error rates to occur more frequently. As a result, the algorithm is very unlikely to become trapped in local minima, but it is also very unlikely to converge to an optimal solution since it is frequently accepts moves to higher error rates. If r is near 0.0, the temperatures in the annealing schedule rapidly approach 0.0 implying that moves to states of higher error rates will almost never be accepted. The search will converge quickly, but most often only to a locally optimal solution.

The number of times to loop at each step, much like the number of steps in the annealing schedule, is primarily a matter of complexity. A larger value adds to the complexity and is also more likely to find a better solution. The final parameter, the initial temperature, is similar to the constant r in its effect. A high initial temperature increases the randomness of the search whereas a low initial temperature may force the search to rapidly converge to a locally optimal solution.

We performed extensive experimentation on the disjunctive normal form dataset, described in Section V.A.2, using various parameter settings to arrive at these values:

- number of steps in annealing schedule ≥ 20
- $r = 0.9$
- initial temperature = 0.1
- times to loop ≥ 50
- mutation probability = 0.1

III.D.3 Random Generation Plus Sequential Selection [42]

This search algorithm is another attempt to use randomness as part of the feature selection process. In particular, it injects randomness into the classical forward and backward sequential selection techniques. Here is a brief description of the algorithm.

```
for i←1 to number_of_generations do
        randomly generate a feature subset
        perform FSS from this subset
        perform BSS from this subset
endfor
```

Fig. 18. RGSS algorithm.

In our implementation, a feature subset is created by generating a random number between 0.0 and 1.0 for each bit in the subset. If the random number is less than 0.5, the bit is one and is zero otherwise.

FSS and BSS by themselves are plagued by the inability to backtrack and reevaluate earlier decisions. RGSS is an attempt to alleviate this drawback by originating the sequential search procedures in varying regions of the search space. One can think of the search space as being broken into partitions. Inside of each of these partitions, either FSS or BSS will obtain the locally optimal solution (i.e., the best solution in that partition). The hope is that one of the randomly generated subsets will fall inside the partition containing the globally optimal solution and either FSS or BSS will proceed to find it. Depending on the complexity of the problem, there may be many or relatively few of these partitions. The number of generations should be large for problems of great complexity to increase the probability that a randomly generated subset will fall in a partition containing the optimal or a near optimal solution.

The complexity of this algorithm is determined by observing that, during each generation, both FSS and BSS are performed. This means that the number of states examined per

generation is $O(n^2)$, the complexity of FSS and BSS individually. Thus, if g is the number of generations, the complexity of the algorithm is $O(g \cdot n^2)$.

The large black points in Fig. 19 are the randomly generated feature subsets. From each of these points, the upside-down funnels represent BSS and the upright funnels represent FSS.

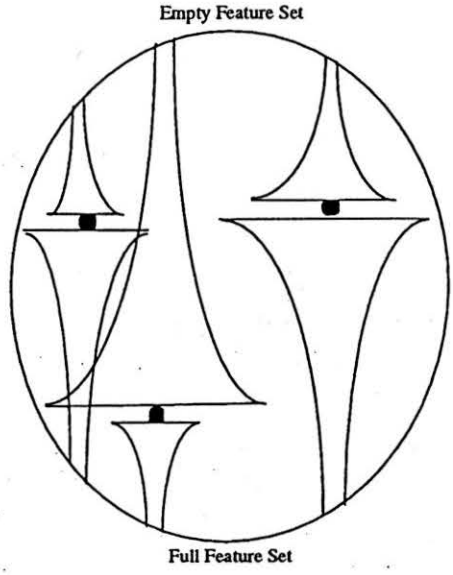

Fig. 19. Random generation plus sequential selection.

Parameter Selection

The only parameter to RGSS is the number of generations. A value of ten was used for all of the experiments and seemed sufficiently large to obtain accurate results. However, since the probability of finding an optimal feature subset increases with each generation, this value should be as large as possible within the given complexity constraints.

III.E Considerations When Choosing a Search Algorithm

Choosing a search algorithm for a particular feature selection task is a complicated process which is highly dependent on the characteristics of the dataset (i.e., the number of features, and whether the features are continuous, categorical, and/or binary). For instance, if we have a dataset for which feature subsets can only be accurately evaluated at high costs (e.g., by using a neural network), we will be forced to use one of the sequential techniques or a scaled down version of one of the random searches to stay within our complexity constraints. For example, we could use genetic search with a small population size and few generations. On the other hand, if your evaluation function is simple and still reasonably accurate on the dataset, a more exhaustive search such as BS or B&B may be in order.

It is also important to know how optimal a feature subset you wish to find and how long you are willing to wait for a search algorithm to find such a subset. If you do not require that a near optimal feature subset be found and you want fast convergence to a solution, one of the sequential techniques may be called for. However, if you require that a very

optimal subset be found and do not care how long the algorithm executes, you may wish to use one of the exhaustive algorithms.

In general terms, an effective search algorithm has both rapid convergence and the ability to avoid local minima. If the algorithm does not converge rapidly, the complexity is usually unmanageable. On the other hand, the inability to avoid local optima will cause the algorithm to find solutions that are unacceptable due to their high degree of sub-optimality. Without proper care, rapidly converging algorithms will only find a local minimum and algorithms which find the global minimum may perform nearly exhaustive searches.

III.E.1 Exhaustive Algorithms

III.E.1.a Exhaustive Search

As the size of the search space is exponential in the number of features, ES can only be used when the number of features is less than around 15 (which yields approximately 32,000 states). One possibility for the use of ES is to first run one of the other search methods to reduce the number of features to a workable number, and then perform ES.

III.E.1.b Branch and Bound Techniques

Both the B&B and AMB&B are subject to two primary drawbacks: 1) in the limit, they perform exponential search; 2) the evaluation function used must be absolutely monotonic for the B&B and nearly so for AMB&B.

The second drawback, that of requiring a monotonic evaluation function, is perhaps the biggest limitation of the algorithm since it biases the algorithm towards large feature subsets. This can readily be seen by recalling that, when using a monotonic evaluation function, the value of any set of features will be no greater than the value of any set of which it is a proper subset. Therefore, small feature subsets, which are optimal or near-optimal, may never be found.

B&B does not appear to be very useful as a general feature selection algorithm since it is limited by both its complexity and its need for a monotonic evaluation function. This means that it cannot be used on large datasets (e.g., greater than 20 attributes) or with classifiers that are non-monotonic. Since most classifiers improve their performance when irrelevant and noisy features are removed, the great majority cannot be used with B&B.

III.E.1.c Beam Search

Recall from Section III.A.4 that BS is basically an improvement over FSS. Consequently, BS should perform well in all situations where FSS is appropriate and give slightly better results. This hypothesis is supported by the experiments presented in Section VI.

III.E.2 Sequential Algorithms

Unlike ES which examines every state in the search space, the sequential techniques (FSS, BSS, PQSS, and BDS) visit only a small fraction of the space. Consequently, these methods can be very sub-optimal for some kinds of datasets (e.g., datasets with a large number of features) since optimal or near-optimal solutions may lie in a region of the search space which was not visited.

We expect that the sequential techniques may not work well in situations where the ability to backtrack is essential. To demonstrate where the ability to backtrack is needed, let us assume that we are running an experiment and are at a point where FSS has selected a set of five features. However, let us also assume that there is another set of five features which is different than the set found by FSS and provides a lower error rate. (Recall that this is possible since FSS does not examine every feature subset consisting of five features, but only those subsets containing the previous best subset of four features plus each

individual remaining feature.) FSS does not have the ability to backtrack and remove features in order to select the features corresponding to the more effective subset.

On the other hand, the sequential methods are likely to perform effectively on simple datasets describing, for example, linear functions since backtracking should not be necessary. In addition, sequential techniques may be the only alternative when using a slow evaluation function since more optimal searches are not feasible.

III.E.3 Randomized Algorithms

GS, SA, and RGSS all introduce a degree of randomness into their respective searches. This randomness can allow these algorithms to avoid local minima and converge closer to globally optimal solutions. Parameter selection is *extremely* important (especially to GS and SA) and exhibits a profound effect on their performance. The art of selecting these parameters is poorly understood at this time and is a potential limitation to their use. Fortunately, the algorithms are robust enough to perform reasonably well enen for mediocre control parameters.

These appear to be very promising choices for a search algorithm. It is conceivable that, with proper parameter selection, the algorithms can both converge rapidly (e.g., a small population size for genetic algorithms) and find globally optimal solutions.

III.E.4 Summary

Table 1 summarizes the main properties of the algorithms belonging to each of the categories discussed above.

	accuracy	complexity	advantages	disadvantages
exhaustive algorithms	always find optimal solution	exponential	high accuracy	high complexity
sequential selection algorithms	good if no backtracking needed	quadratic	simple and fast	cannot backtrack
randomized algorithms	good with proper control parameters	generally low	designed to escape local minima	difficult to choose good parameters

Table 1. A summary of search algorithms.

IV. Evaluation Functions

An evaluation function is needed to calculate the effectiveness of a particular set of features after the search algorithm has chosen it for examination. The search algorithm, whichever one is currently in use, utilizes the value returned by the evaluation function to help guide the search. For instance, FSS tries all possible ways of adding a feature to the current best feature subset at each step of the algorithm. The feature subset resulting in the best performance, *according to the evaluation function*, is made the current best subset. The search then continues from this current best feature subset.

The above example demonstrates the importance of the evaluation function to the search process. If the evaluation function indicates that a particular subset of features is more effective than another subset when in fact it is not, the best search algorithm will be led astray and will in all likelihood return an inferior solution. In other words, the *relative* accuracy of the evaluation function is important, not its absolute accuracy. To demonstrate this, let us assume that we are using a neural network based classifier as the evaluation function and it determines that two feature subsets S1 and S2 provide error rates of 3% and 5%, respectively. During another experiment, subsets S1 and S2 are again evaluated, this time by a Bayesian classifier, and obtain error rates of 25% and 20%, respectively. The neural net classifier indicates that S1 is the more effective subset whereas the Bayesian classifier indicates S2. Let us assume that, in reality, S2 is a more effective subset than S1. Since the Bayesian classifier has correctly determined that S2 is the better of the two subsets, it outperforms the neural net classifier in this situation although its absolute error rates are higher.

Another factor to be considered is the speed of an evaluation function. Given two evaluation functions that are equally adept at distinguishing between the worth of feature subsets, the one that calculates the value of a state more rapidly is the one which should be used. Also note that a particular evaluation function can be used with any of the search algorithms (except in the case of B&B, which requires a monotonic or near monotonic evaluation function), although different evaluation functions may work better with different search algorithms.

There are three different kinds of evaluation functions that can be used to determine the value of a state in the search. There are those that use

- the intrinsic properties of the data
- the classification accuracy
- an estimate of the classification accuracy or incrementally calculate the classification accuracy.

There are some fundamental differences between these three methods. For instance, using intrinsic measures is less costly than determining the error rate. This is because most intrinsic measures involve straightforward calculations of simple functions whereas classifiers require training and testing on a significant amount of data. On the other hand, the extra effort put into evaluating a state by using the classification accuracy is likely to result in a better estimate of its utility.

The three subsections which follow give examples and a general description of each of the various kinds of evaluation functions.

IV.A Intrinsic Properties of the Data

The term intrinsic properties refers to any information which can be extracted from the data and no attempt is made to classify previously unseen examples using this information. There are various categories of evaluation methods which are based upon the intrinsic properties of the data [9]:

- probabilistic distance measures
- probabilistic dependence measures
- interclass distance measures
- entropy measures.

The remaining subsections describe each of these in greater detail.

IV.A.1 Probabilistic Distance Measures

The following formula represents the general operation of probabilistic distance measures:

$$J = \int f\big(P(\xi/w_i), P(w_i)\big)\, d\xi, \quad i = 1,2$$

where $P(\xi/w_i)$ refers to the probability of feature vector ξ given class w_i and $P(w_i)$ is the *a priori* probability of class w_i (i.e., the number of instances from class w_i divided by the total number of instances). To qualify as a probabilistic distance measure, the function $f(\cdot)$ must be defined so that the value of J satisfies the following conditions: 1) $J \geq 0$, 2) $J = 0$ when $P(\xi/w_i), i = 1,2$ are equal (overlapping), and 3) J is maximum when $P(\xi/w_i), i = 1,2$ are well separated (non-overlapping). If the features used in the feature vectors are good, the gap between the conditional probabilities of the two classes for each feature vector will be significant and the value for J should be large. Poor features will likely result in nearly equal probabilities and the value for J should be small.

One type of probabilistic distance measure is the *Bhattacharyya distance* [9], given by

$$J = -\ln \int \sqrt{P(\xi/w_1) \cdot P(\xi/w_2)}\, d\xi$$

After a brief inspection, it can be seen that it satisfies the constraints, noted above, required of a probabilistic distance measure. The specific characteristics of the *Bhattacharyya distance* are that it does not use the *a priori* probabilities and that it takes the square root of the product of the conditional probabilities to calculate the value of a state. This method was implemented and used during our experimentation process.

IV.A.2 Probabilistic Dependence Measures

These measures calculate the dependence between the feature vector ξ and the class w. If they are independent, $\left|P(\xi/w) - P(\xi)\right|$ will be 0 since $P(\xi/w) = P(\xi)$ and no information is gained from the feature vector. However, when they are highly dependent, $\left|P(\xi/w) - P(\xi)\right|$ will be large since class knowledge either increases or decreases the probability that the feature vector will occur. The dependence implies that there is information in the feature vector ξ which distinguishes the conditional probability from the *a priori* probability. The presence of this information in turn implies that the features being used allow a more accurate prediction of a feature vector's membership in a particular class. The *Patrick-Fisher* method, shown below, is one example of a probabilistic dependence measure.

$$J = P(w_i) \cdot \left(\int \left(P(\xi/w_i) - P(\xi)\right)^{1/2} d\xi \right), \quad i = 1,2$$

IV.A.3 Interclass Distance Measures

Interclass distance measures [9], like the probabilistic measures, are methods used for calculating the utility of feature subsets without determining the error rate. These methods measure the ability of feature subsets to separate the classes and rely on the assumption that attribute vectors of different classes will lie in distinct regions of the search space. The average pairwise distance between training examples from different classes is calculated for the various feature subsets with higher values of the average pairwise distance being translated into better feature subsets.

An example of an interclass distance measure, the *Euclidean distance*, follows:

$$d(\xi_k, \xi_l) = \left(\sum_{j=1}^{n} (\xi_{kj} - \xi_{lj})^2 \right)^{1/2}$$

where n is the number of features in a pattern vector and ξ_{kj} is the jth feature of pattern vector ξ_k. This measure calculates the distance, Euclidean style, between attribute vectors ξ_k and ξ_l. The part of the formula inside the large parentheses is the sum of the squares of the differences between feature values from the two pattern vectors. The square root of this value is then taken and returned as the distance.

The formula given above only calculates the distance between two individual feature patterns. To calculate the *average pairwise distance* between two classes, we must use the formula:

$$J = P(w_1) \cdot P(w_2) \cdot \frac{1}{N_1 \cdot N_2} \cdot \sum_{k=1}^{N_1} \sum_{l=1}^{N_2} d(\xi_{1k}, \xi_{2l},)$$

where N_i is the number of examples in class w_i and ξ_{ik} is the kth pattern vector of class w_i. d represents the distance function of which the aforementioned *Euclidean distance* is an example.

IV.A.4 Entropy Measures

This type of measure is very similar to the methods of probabilistic dependence except that the *a posteriori* probabilities, $P(w/\xi)$, are used to determine the dependence between a feature vector and the various classes. (Recall that the probabilistic dependence measures use the probability of a feature vector given a class to assess the dependence.) Given a particular attribute vector, if all classes are equally probable, then the uncertainty is a maximum and little has been learned from the feature vector. A large variance between the *a posteriori* probabilities for a pattern vector indicates that the dependence is high and the features used to represent the attribute vectors are effective. One example of this kind of measure is the *Quadratic*:

$$J = \int \left(\sum_{i=1}^{2} P(w_i/\xi)^2 \cdot P(\xi) \right) d\xi$$

IV.B Classification Error Rate

The second type of evaluation function is a classifier. The idea is to train a classification system (e.g., back-propagation neural network, Bayesian classifier, decision tree, etc.) with some of the available data and test it on the remaining data. (Standard machine learning methodology is to train on two-thirds of the data and test on the remaining one-third.) The data are vectors of feature values where each vector is labeled with the class to

which it belongs. During the training, a classification algorithm attempts to learn the appropriate concept (e.g., rules distinguishing between attacks and non-attacks). The accuracy of the learned concept is then tested to determine the error rate. If the classifier performs well using a particular subset of features, it is a strong indication that this is a predictive combination of attributes.

It might be argued that using classifiers as evaluation functions is really no different than using measures which rely on intrinsic properties of the data since the classification algorithms themselves are trained by analyzing properties of the data (e.g., Bayesian learning uses statistical properties of the dataset to construct a classifier). However, the goal of feature selection is to extract the optimal feature set to be used *in a classification system* (e.g., an intrusion detection system where users are classified by their level of suspicion) and it is thus reasonable to consider the classification error rate as an evaluation function for feature selection[7].

The two types of classifiers used in our experiments are a Bayesian classifier[8] and the decision tree classifier C4 [43][9]. These two classifiers were chosen because they are symbolic learning algorithms and are easier to train than other types of classifiers such as neural networks. The Bayesian classifier operates by counting the number of occurrences of the various values of each feature for all the classes. For instance, assume that we are using binary valued attributes and have five initial features. Given the following three attribute vectors for a particular class,

- <0,1,1,1,1>
- <1,0,0,1,1>
- <0,0,1,0,0>,

the count vector for that class will be

- <[2,1],[2,1],[1,2],[1,2],[1,2]>.

This can be translated as meaning that '0' has occurred twice as the value of the first feature and '1' has occurred once. The second feature has taken on the value '0' twice and has taken on the value '1' once, and likewise for the remaining features. What is important to note is that these counts only need to be calculated once, not each time a new state is examined; therefore, they can be calculated before the search begins. When determining the value of the current search state, only those counts corresponding to the features contained in the state are used to classify examples from the testing set.

Once the counts have been derived from the training data, the algorithm in Fig. 20 is used to classify instances from the test set.

highest_probability ← 0.0

for i←**1 to** number_of_classes **do**

---

[7] If we are selecting features to be used with a particular classifier and want to know which features work the best *with that* classifier, the most effective evaluation function is the classifier itself.

[8] A Bayesian classifier written by Doug Mayfield, graduate student at UC Davis, was modified and used for the evaluation function.

[9] Thanks go to Wray Buntine and NASA Ames Research Center for the use of the IND Tree Package [35]. The package allows the user to specity options which control the characteristics of the decision tree. The particular options chosen implement Ross Quinlan's C4 algorithm.

```
        current_probability ← P(w_i)
        for j←1 to number_of_features do
            if feature_subset[j] = 1 then
                current_probability ← current_probability *
                    (counts[i][j][current_vector[j]] / number_in_class[i])
            endif
        endfor
        if current_probability > highest_probability then
            class ← i
        endif
    endfor
```

Fig. 20. Testing algorithm of the Bayesian classifier.

The algorithm works by determining the probability that a feature vector will belong to each of the various classes. The class which obtains the highest probability is returned as the predicted class of the test example. For each class, we initialize the probability that the testing example occurred in that class with the probability of the class itself (i.e., the number of occurrences of this class in the training set divided by the total number of examples in the training set). This means that, given no information from the features (i.e., in the case of the null feature subset), we return the most frequently occurring class as the predicted class. The step following initialization examines *feature_subset*, which denotes the current state in the search, to determine which features to use and which to ignore when calculating the probability of the class. When a position in *feature_subset* has the value of one, it means that the feature is included in the subset and we can utilize that feature in the calculation of the probability. We use a feature by multiplying *current_probability* by the probability of occurrence of the particular value of the feature given in the current testing example, *current_vector*. The probability of a feature value occurring is the number of times that value occurred for the feature for the particular class in question divided by the number of times examples from this class were seen in the training set.

Our implementation of the Bayesian classifier was not engineered to handle missing feature values (i.e., the values of features are unknown) or non-numerical data (e.g., the names of files accessed). As a result, the Bayesian classifier could only be applied to the simulated databases and not the machine learning databases, since the latter has both missing features and nominal data[10].

C4 is a decision tree classifier [43] and is generally considered by the machine learning community to perform well. It uses pessimistic pruning, thus generating fairly small trees, and calculates the information gain as its splitting criterion.

IV.C Estimated or Incremental Error Rate

Training and testing a classifier for each examined state in the search space can put severe limitations on the size of the search. If the error rate can somehow be estimated or incrementally calculated with little loss in accuracy, the computational savings will allow a

---

[10] The databases are described in Section V.

more thorough analysis of the search space. Terrance Goan, a graduate student at UC Davis, performed part of his thesis work on developing a decision tree classifier that allows features to be incrementally added or removed without retraining on the entire dataset [10]. This could result in substantial savings during the search since a decision tree would not need to be built entirely from scratch each time a new feature subset was evaluated. Unfortunately, the results of his work were not available in time to be used for our experimentation.

IV.D Considerations when Choosing an Evaluation Function

Two primary factors should be studied before selecting an evaluation function:

- search algorithm used
- characteristics of dataset

The search algorithm places a complexity constraint on the evaluation function. For example, with expensive search algorithms (e.g., simulated annealing with a large number of steps in the cooling schedule), slow evaluation functions cannot be used. The characteristics of the dataset determine what kinds of evaluation functions may give accurate results. A neural net classifier may be more suited to speech recognition than a decision tree classifier, but the decision tree may be more appropriate for learning numerical functions.

We now discuss factors specific to each of the various evaluation functions.

IV.D.1 Intrinsic Properties of the Data

The main attractiveness of this type of evaluation function is its speed. Using any of the various intrinsic measures allows a more optimal search (i.e., closer to exhaustive) since it takes less time to calculate the utility of individual states. On the other hand, these functions are usually less effective than slower evaluation functions such as classifiers.

Another drawback to these evaluation functions is their monotonicity. Recall that this means the value of a feature subset is non-increasing when features are removed from it. In other words, the feature subset with the highest value will *always* be the full feature subset. It is then up to the system designer to make an arbitrary cutoff on the number of features to be used in his particular IDS. Since one of the purposes of feature selection is to avoid these ad-hoc decisions whenever possible, this is a serious limitation to the usefulness of these measures in feature selection.

IV.D.2 Classification Error Rate

This type of evaluation function may be the most directly suitable to the feature selection task since the system for which the features are being chosen is itself a classifier. Classifiers are also non-monotonic implying that their performance will often degrade when irrelevant or noisy features are used in the classification task. This takes a good deal of the guessing out of feature selection since the algorithms will not always return the full feature subset as the one with the highest value, but may return that feature subset which contains the most relevant and the fewest irrelevant features. Unfortunately, training and testing a classifier for each state in the search may prove to be too costly for a more optimal search, dictating that we use one of the less expensive searches such as FSS.

IV.D.3 Estimated or Incremental Error Rate

Estimating or calculating the error rate incrementally is an attempt to combine the speed of the intrinsic measures with the accuracy of the classifiers. As such, when they can be used, they may be the best choice for the evaluation function. Unfortunately, their applicability has one major limitation: only those searches which add or subtract a few features at a time from the current best search state (e.g., BSS) can be used. Otherwise,

the feature subsets vary so greatly from step to step (i.e., few features overlap) that the classifier cannot be updated in an incremental fashion, nor can the error rate of the new state be estimated from that of the old.

IV.D.4 Summary

Table 2 summarizes the main properties of the evaluation functions belonging to each of the categories discussed above.

	accuracy	complexity	advantages	disadvantages
intrinsic properties of the data	low	low	very fast	inaccurate and monotonic
classification error rate	very high	high	very accurate	slow
estimated or incremental error rate	high	moderate	high accuracy *and* fast	restricts type of search algorithm

Table 2. A summary of evaluation functions.

V. Experimental Databases

Two types of databases were used in the experiments. The first variety, the simulated databases, describe simple to reasonably complex boolean functions and are artificial in the sense that the examples were created with the use of a random number generator. An example is first generated, then it is tested in order to determine its classification. These simulated databases were designed so that we would know which of the features used to represent the examples are relevant and which are irrelevant. In other words, only some of the features in a database pertain to the boolean function while the others are noise. This allows us to determine exactly how well the feature selection algorithms perform since we can determine the number of relevant and irrelevant features contained in feature subsets. The other databases are used primarily in machine learning and should indicate how well the algorithms perform on real world data. The attributes in these datasets are both categorical (discrete) and continuous and are more like the features we foresee to be needed in IDSs. Since we do not know the best feature subset for these datasets, the best subsets found by the feature selection algorithms are compared to the subset containing all the features. We expected to see an improvement in the error rate while reducing complexity since the feature subsets found by the algorithms should have fewer irrelevant or noisy features.

All the databases are split into a training set and a testing set with the training set generally twice as large as the testing set.

V.A Simulated Databases

All of the simulated databases consist of randomly generated positive and negative examples of boolean functions. Each example consists of the class followed by the values of each of the features. Although the features are only binary valued, these databases give us a very good indication of how well a feature selection algorithm performs since we know exactly which features are relevant and which are not.

V.A.1 Disjunction Database

These are the characteristics of this database:

- size of training set: 200 instances
- size of testing set: 100 instances
- number of classes: 2
- features: 10 binary features, f1, ... , f10
    - relevant: 5
    - irrelevant: 5
- learned function: f1 V f2 V f3 V f4 V f5 (V is the OR operator.)
- sample instances:

    positive < 0 1 0 0 0 1 0 0 0 1 >

    negative < 0 0 0 0 0 0 1 1 0 1 >

Looking at the learned function above, we see that when any of the features f1 through f5 have the value of '1', an example is classified as positive and is negative otherwise. The values of features f6 through f10 do not affect the classification of an example. In the first example shown under sample instances, observe that features f2, f6, and f10 have the value of '1'. Features f6 and f10 are irrelevant and do not affect the classification. However, since f2 is '1' and it is one of the first five features, the example is positive.

None of the first five features are '1' in the second example and it is negatively classified. Positive and negative examples were generated with equal probability.

V.A.2 Disjunctive Normal Form Database

The following are the characteristics of this database:

- size of training set: 1000 instances
- size of testing set: 500 instances
- number of classes: 2
- features: 30 binary features, f1, ... , f30
  - relevant: 9
  - irrelevant: 21
- learned function:

  $(f7 \wedge f12 \wedge f15) \vee (f17 \wedge f21 \wedge f23) \vee (f24 \wedge f25 \wedge f26)$
- sample instances:

  positive < 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1 >

  negative < 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 >

The learned function is a three term disjunctive normal form, 3-DNF. The positive instance above satisfies the function as features f7, f12, and f15 are '1'. Again, positive and negative examples were generated with equal probability.

V.A.3 Attack Database

These are the characteristics of this database:

- size of training set: 1000 instances
- size of testing set: 500 instances
- number of classes: 2
- features: 50 binary features, f1, ... , f50
  - relevant: 13
  - irrelevant: 37
- learned function:

  If any 5 of f6, f17, f28, f29, f40, f42, and f43 are '1'

  then class is positive

  else if any 4 of f1, f17, f20, f33 and f49 are '1'

  then class is positive

  else if all 3 of f5, f6 and f37 are 1

  then class is positive

  else

  class is negative

  endif

• sample instances:

negative <0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0
0 0 1 0 1 1 0 1 0 1 1 0>

positive <0 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 0
1 1 0 1 0 0 0 0 1 0 0 0>

This dataset was constructed by giving it the characteristics which we think an actual attack database would have. The features are binary and resemble the bernouli vector used by Haystack Laboratories in their IDS, *Haystack* [1]. A '1' value for a feature signifies that the feature value is out of range or abnormal. Haystack determines that a value is out of range by establishing limits inside of which 90% of historical activity has occurred. A value is abnormal if it falls outside of these limits. For instance, if a user logs in 90% of the time between 8:00 am and 9:00 am in the morning, a login by that user before 8:00 am or after 9:00 am would result in a '1' for the feature representing login time.

Each clause in the learned function represents a different method of attack. For instance, features f5, f6 and f37 might represent number_of_attempted_logins, login_location, and time_of_login, respectively. If all three of these features are abnormal, the system will classify the user as an attacker who is trying to gain illegal entry by attempting numerous login/password combinations.

Notice that feature f17 overlaps between the first and second clauses and that feature f6 overlaps between clauses one and three. In other words, some features may be useful at catching numerous kinds of attack. login_location, f6, may help to find not only doorknob attacks, but also attacks which are detected because they originate from suspicious sources.

Some dependencies between features have been inserted into the data. In the first clause, if f28=1, then f42 is strongly biased to be '1' and if f17=1, then f29 is biased to be '1'. The second clause has a dependency between f20 and f33 where if f20=1, f33 is very likely '1'. The dependencies were inserted to make the data more closely resemble real attack data since we suspect that there will be features which, when simultaneously abnormal, are indicative of an attack (e.g., when time_of_login is abnormal, login_location is likely to be abnormal and, when occurring together, the likelihood of an attack is high). The method for generating the dependencies was to first generate the value for a feature on which another feature depends. If the value was '1' (e.g., if f17=1), then the other feature (e.g., f29) was set to '1' with a 90% probability.

Except for the last clause, it is not necessary for all features to be '1' for the instance to be classified as positive. Rather, if most of the features are '1', the example is positive. This characteristic enables the dataset to resemble real world attack data where only a majority of features which indicate a particular intrusion type need to be abnormal to flag an instance as an attack.

On the average, an equal number of positive and negative examples are generated.

V.B Machine Learning Databases[11]

Three machine learning databases were used to test the feature selection algorithms on real world data. Binary, categorical (discrete) and continuous valued attributes are all present in these databases and there are unknown feature values. (It is also likely that many feature values are incorrect and that a significant number of features are irrelevant.) In general, we do not know which features in these databases are relevant and which are not. Therefore,

---

[11] The databases were obtained via anonymous ftp from UC Irvine at ics.uci.edu in the directory pub/machine-learning-databases.

we will evaluate the effectiveness of the algorithms by comparing the best feature subset found by an algorithm to the full feature set. If the error rate of the feature subset is less than the full set of features and it contains considerably fewer features, we can conclude that the feature selection algorithm has performed well.

For example, consider the case where we are trying to find the optimal feature subset from an initial set of five features and we know that the full feature set has an error rate of 13%. Assume that a feature selection algorithm determines that features four and five together form the optimal feature subset having an error of only 6%. Clearly, the algorithm has performed well since the error rate has dropped by 7% and we have reduced the size of the feature subset from five to two features. This decrease in size will reduce the complexity of the algorithm for which the features are being chosen since fewer features are used in its calculations.

The primary factor influencing the choice of databases was that they had to contain features which had some intuitive meaning. Although this does not allow for a rigorous determination of what features are relevant, it can give us a feeling for the effectiveness of the algorithm based upon our intuitive notions of which features should be relevant. Also affecting the selection of databases was the desire to determine the generality of the feature selection algorithms by analyzing their performance on databases with differing characteristics: one database contains only binary features; another contains only discrete features; the remaining database contains binary, discrete, and continuous features; and the sizes of the databases vary from 16 to 75 attributes. It is a strong indication that an algorithm will perform well on varying types of datasets if it performs well on all three databases.

One database, the heart disease database, was chosen because it has a large number of features (75) and a mixture of binary, categorical, and continuous attributes. Since we expect effective attack databases to contain large numbers of features of varying types, the performance of the feature selection algorithms on this dataset should be indicative of their performance on an actual attack dataset.

V.B.1 Congressional Voting Records Database [36]

These are the characteristics of this database:

- size of training set: 290 instances
- size of testing set: 145 instances
- number of classes: 2
    - republican, 38.6% of instances
    - democrat, 61.4% of instances
- features: 16 binary features, corresponding to yes or no votes on issues.
- missing feature values: yes
- sample instances:

    republican n y n y y y n n n n n y y y n ?

    democrat ? y y ? y y n n n n y n y y n n

    'y' is a positive stance on a vote, 'n' is a negative stance, and '?' is an unknown disposition.

This dataset contains votes on key issues, as determined by the Congressional Quarterly Almanac (CQA), for each of the U.S. House of Representatives Congressman during the 98th Congress, Second Session, 1984. There are nine different types of votes identified

by the CQA. Those translating into a 'yes' vote are voted for, paired for, and announced for. 'no' votes were derived from voted against, paired against, and announced against. Finally, unknown positions (?) were obtained from voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known.

Section 1 of Machine Learning Database Features in the Appendix describes each of the 16 key votes.

V.B.2 Soybean Disease Database[37]

These are the characteristics of this database:

- size of training set: 307 instances
- size of testing set: 376 instances

  The ratio of the size of the training set to the size of the test set is not 2:1 since the data we obtained was already broken into training and test sets containing 307 and 376 instances, respectively.

- number of classes: 19

  It is typical for researchers to use only 15 of the classes since the other four classes are only represented by a few samples.

- features: 35 discrete features are used to describe the condition of a soybean plant.
- missing feature values: yes
- sample instances:

diaporthe-stem-canker	6 0 2 1 0 1 1 1 0 0 1 1 0 2 2 0 0 0 1 1 3 1 1 1 0 0 0 0 4 0 0 0 0 0 0
charcoal-rot	6 0 0 2 0 1 3 1 1 0 1 1 0 2 2 0 0 0 1 0 0 3 0 0 0 2 1 0 4 0 0 0 0 0 0
rhizoctonia-root-rot	1 1 2 0 0 2 1 2 0 2 1 0 0 2 2 0 0 0 1 0 1 1 0 1 1 0 0 3 4 0 0 0 0 0 0

This is one of the more well-known machine learning datasets and was chosen precisely for its familiarity. It is hoped that by using such a familiar dataset, further work on feature selection will also use this dataset thereby providing an invaluable comparison of various works. See Section 2 of Machine Learning Database Features in the Appendix for a complete description of the classes and features.

V.B.3 Heart Disease Database

These are the characteristics of this database:

- size of training set: 600 instances
- size of testing set: 299 instances
- number of classes: 5

  The class attribute is integer valued from zero to four. Zero corresponds to no heart disease and four is severe heart disease.

- features: 75 binary, discrete, and continuous features are used to describe the heart condition of a patient.
- missing feature values: yes

• sample instances:

no heart disease: 0 (class) 1 0 63 1 ? ? ? ? 1 145 1 233 ? 50 20 1 ? 1 2 2 3 81 0 0 0 0 0 1 10.5 6 13 150 60 190 90 145 85 0 0 2.3 3 ? 172 0 ? ? ? ? ? ? 6 ? ? ? 2 16 81 1 1 1 ? 1 ? 1 ? 1 1 1 1 1 1 1 ? ? name

severe heart disease: 4 (class) 26 0 60 1 ? ? ? ? 4 130 1 206 ? 50 32 0 ? 0 2 5 19 81 0 1 0 0 0 1 5 3 7 132 83 140 80 130 88 1 0 2.4 2 ? 24 2 ? ? ? ? ? ? 7 ? ? ? 5 20 81 2 1 1 ? 2 ? 1 ? 2 1 1 1 1 1 1 ? ? name

This database is the combination of four individual heart disease databases from various institutions[12]. The results obtained with this database, along with the simulated attack database, will be given the most weight when determining good combinations of search algorithms and evaluation criteria to be used in selecting features for intrusion detection systems. This is because the characteristics of the dataset (i.e., many features of varying types) are considered to be similar to the characteristics of data that one would need to analyze in order to detect intrusive behaviour.

[12] Section 3 of the Appendix Machine Learning Database Features gives the names of the institutions and the principal investigators responsible for gathering the data.

VI. Experimental Results

This section presents the results of our experiments using various combinations of search algorithms and evaluation functions on the simulated and machine learning databases. Line graphs are used to present the results with the x-axis referring to the number of features in a subset (referred to as the level) and the y-axis denoting the value of the evaluation function. For a point in a graph, the y value refers to the best value, as determined by the evaluation function, of all feature subsets examined which contained the number of features denoted by the point's x value. For example, the point (3, 0.15) signifies that the best error rate found for all the feature subsets examined containing exactly three features was 15%. The range of the x-axis is 0 (no features) to the number of features in the full feature subset. The range of the y-axis is between 0.0 and 1.0 for the error rate evaluation functions, greater than 0.0 for the Euclidean distance, and is not bounded from either above or below for the Bhattacharyya distance.

The overall effectiveness of the search algorithms will be judged by both their complexity and accuracy. The complexity of an algorithm is determined by the number of states examined in the search space and is shown in the legend of each figure. For example, the legend in Fig. 21 shows that FSS examined 56 states during its search on the disjunction database. The accuracy is calculated by comparing the error and number of features in the full feature set to those of the best feature subset (i.e., the subset with the best combination of low error rate and small number of features) found by the search algorithm. In addition, a further analysis of the effectiveness of the algorithms on the simulated databases can be determined since we know *exactly* which features are relevant and which are not. How many of the relevant features were found? How many irrelevant features are included in the best feature subset?

When selecting that search algorithm which performed the best on a particular experiment, we need to combine the complexity and accuracy of the algorithms into one overall measure. The criteria used was first to select the search algorithm(s) which found the best feature subsets (those with the lowest error rates). Then, from these algorithms, the one which examined the fewest states was determined to be the top performer. For example, assume that GS and FSS both found feature subsets with the lowest overall error rate when run on the DNF database using the decision tree classifier as the evaluation function. However, if we also assume that FSS examined one-half the number of states as GS, FSS is superior on this database with this evaluation function.

We determine the effectiveness of the various evaluation functions also by creating a combined measure of their complexity and accuracy. For evaluation functions, complexity refers to the amount of work needed to evaluate one state in the search space. The accuracy of an evaluation function is its ability to distinguish between good and bad feature subsets. The evaluation function with the lowest complexity of those algorithms obtaining the highest accuracy is considered as the best.

When examining the search graphs where an error rate evaluation function was used, we noticed the tendency of the error rate to increase as the level approaches that of the full feature subset. The irrelevant and noisy features included at these high levels tend to worsen the performance of the classifiers.

The following is a list of the search algorithms implemented:

- exhaustive search (depth-first)
- beam search
- forward sequential selection
- backward sequential selection

- random generation plus sequential selection
- genetic search
- simulated annealing

An inspection of this list will show that two of the exhaustive algorithms, two of the sequential algorithms, and three of the randomized algorithms were implemented. Implementing at least two methods from each class of algorithms allowed us to compare the performance of algorithms from different classes and study the general characteristics of each class. In addition, since the exhaustive and sequential algorithms are already well understood, it was deemed better to place more emphasis on the randomized searches; hence, an additional algorithm from this generation was implemented.

The following four subsections are organized so that each subsection presents only those results using a particular evaluation function.. The first two pertain to the intrinsic measures and show only the performance of the FSS algorithm on the simulated datasets. Although other search algorithms were tested with the intrinsic measures, the intrinsic measures were so ineffective at evaluating subsets that only the results using FSS on the simulated databases are shown. The last two subsections present the results of using the C4 decision tree and the Bayesian classifier as the evaluation function. The focus in these sections is to compare the various search algorithms using the same evaluation function and dataset.

VI.A Euclidean Distance

The results of using the Euclidean distance with FSS on the simulated databases is shown in the following three subsections.

VI.A.1 Disjunction Database

Fig. 21 shows the results of running FSS on the disjunction database using the Euclidean distance as the evaluation function.

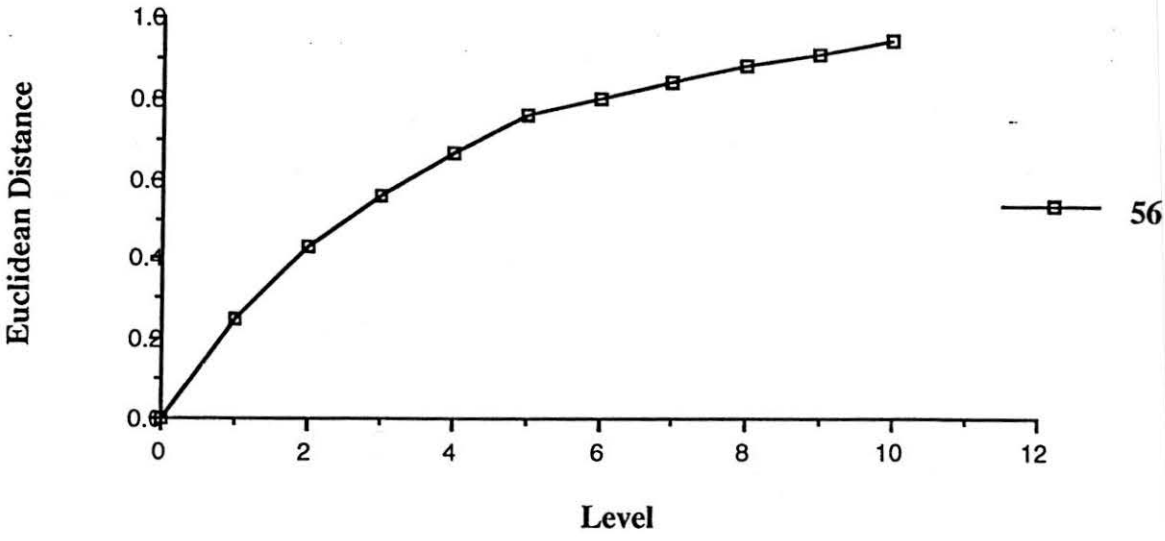

Fig. 21. Euclidean distance / FSS / disjunction database.

With only 56 states expanded, the complexity of FSS on the disjunction database is very low. Choosing the best feature subset found by the algorithm in order to evaluate its accuracy is difficult due to the monotonicity of the Euclidean distance. This is because there is no clear point at which adding features no longer improves the value of a feature subset. Going strictly by the value of the Euclidean distance, the best feature subset is the full feature set. However, since we are using a monotonic evaluation function, the best value at each level increases even when irrelevant features are added. Therefore, when deciding on the best feature subset, it is wise to select the one which corresponds to the point on the graph where the slope is decreasing and the change in the slope is the most drastic.

In Fig. 21, there appears to be a significant drop in the slope between levels five and six which might prompt a user to select the feature subset at level five as the best set of features. The best feature subset at level five, however, does not contain any of the relevant features (i.e., f1, ... , f5) and contains all of the irrelevant features. (Recall that since this is one of the simulated databases, we know exactly which features are relevant and which are not.) This means that the user, if guided by the Euclidean distance, may choose the worst possible feature subset as the best. The reason for this anomaly can be found by examining the characteristics of the disjunction database. In order to produce an equivalent number of positive and negative instances, f1 through f5 were generated as '0' most of the time (87%) and '1' only a small fraction of the time. As the Euclidean distance determines which features give the largest separation in values between the classes, it will not choose any of the first five features as the most relevant since they are '0' the majority of the time for *both* classes. An examination of the following two sample instances from the disjunction database highlights this anomaly:

- positive < 0 1 0 0 0 1 0 0 0 1 >
- negative < 0 0 0 0 0 0 1 1 0 1 >

Comparing these two instances, we see that only one bit differs in the first five features whereas three bits differ in the last five. When this effect is averaged over the entire dataset, the algorithm assumes that the last five features are better at distinguishing between the two classes since they give a greater separation between the classes.

VI.A.2 Disjunctive Normal Form Database

Fig. 22 presents the results of running FSS on the DNF database using the Euclidean distance as the evaluation function.

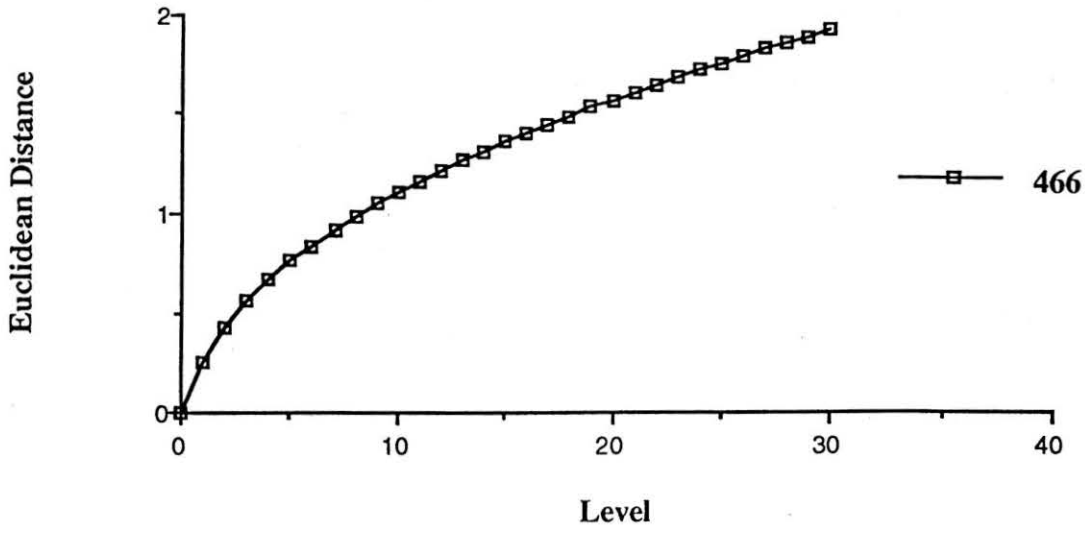

Fig. 22. Euclidean Distance / FSS / DNF database.

As always with the sequential selection techniques, the complexity is very low. Out of a possible 2^{30} feature subsets, only 466 are examined. To determine the accuracy, we must choose which feature subset evaluated by the algorithm is the best. Since we know that there are nine relevant features in this database, one way of determining the effectiveness of this evaluation function is to calculate the number of relevant and irrelevant features at level nine. If the function is performing effectively, it will have a high proportion of relevant features at this level.

Here is the best feature subset found at level nine:

1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0

Relevant features f17, f21, and f26 were found in addition to six irrelevant features. The algorithm appears to do little better than random draw since three out of the nine features selected were relevant whereas there are nine relevant features out of 30 overall, a ratio of one to three for both. As with the disjunction database, the poor performance of the Euclidean distance can be explained by examining the characteristics of the data. All features, regardless of class, are '1' or '0' with equal probability in the DNF database which means that, according to the Euclidean distance, each feature is equally effective at separating the two classes. This causes the algorithm to randomly pick the best feature subset at each level since there is no guidance from the individual features.

VI.A.3 Attack Database

Finally, Fig. 23 shows the results of using FSS on the attack database.

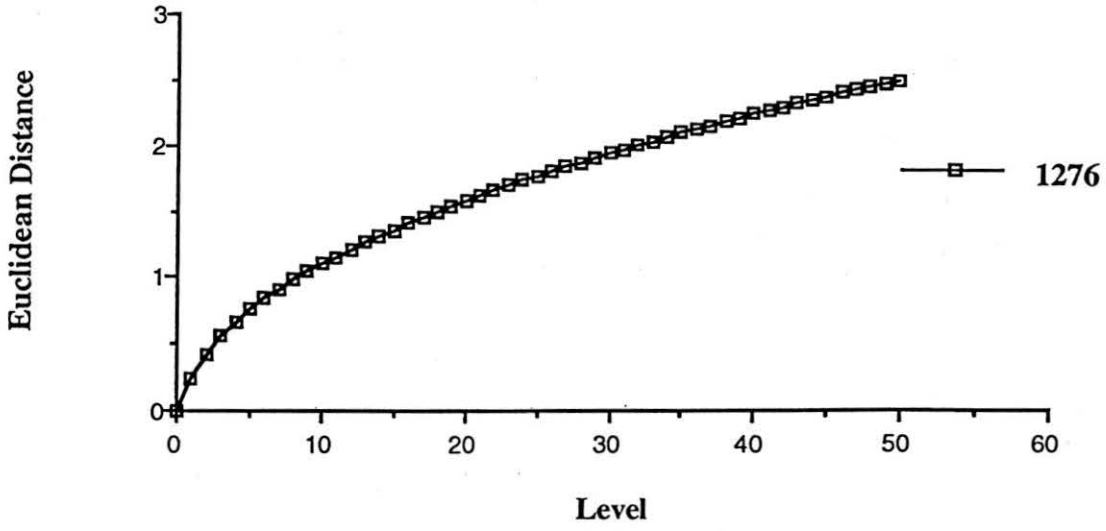

Fig. 23. Euclidean distance / FSS / attack database.

Let us judge the accuracy of this experiment by comparing the features included in the subset at level 13 to the 13 relevant features in the database. This is the best subset found at level 13:

0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0

Four out of the 13 relevant features are included in this subset. As for the DNF database, this is no better than random draw since the fraction of relevant features for the full feature set, 13/50, is approximately the same.

VI.A.4 Remarks

Other experiments were run using the Euclidean distance with different search algorithms and similar poor results were obtained. The main problem appears to be the assumption that the relevance of a feature subset can be determined by calculating the difference of feature values between the classes for those features included in the subset. As we saw in the three examples above, this is often an erroneous assumption.

VI.B Bhattacharyya Distance

The performance of the Bhattacharyya distance on the simulated databases using FSS is presented below.

VI.B.1 Disjunction Database

The results of FSS on the disjunction database are shown in Fig. 24.

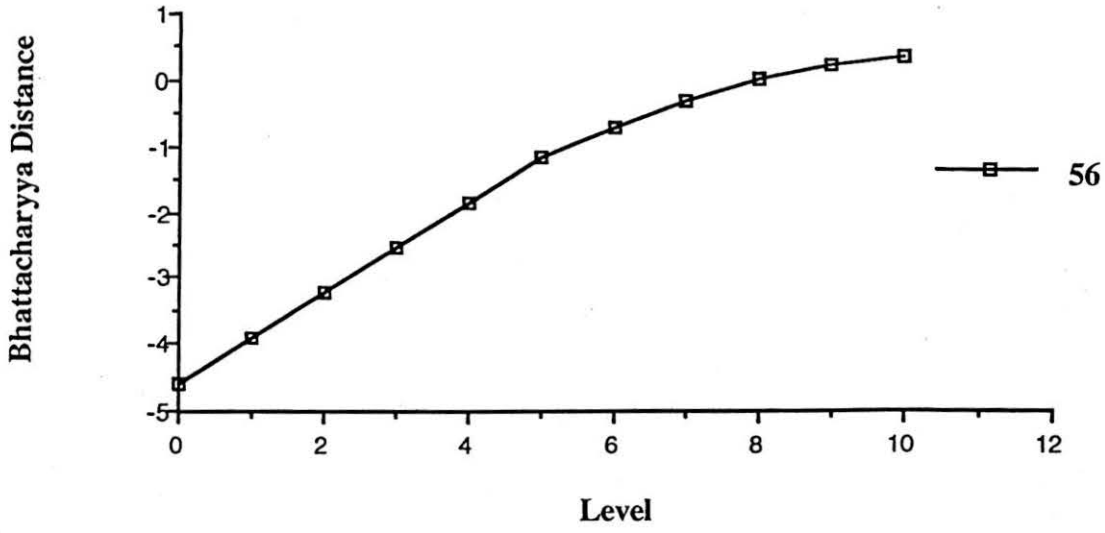

Fig. 24. Bhattacharyya distance / FSS / disjunction database.

Fig. 24 is almost identical to Fig. 21 which used the Euclidean distance with FSS on the disjunction database. Once again the features subset at level five appears to be the best choice and as before contains *only irrelevant* features. The reasons for this are similar to those discussed before. Namely, features f1 through f5 are '0' most of the time for both classes leading the function to conclude that they are poor features for determining the probabilistic distance between the classes.

VI.B.2 Disjunctive Normal Form Database

The Bhattacharyya distance, however, does perform significantly better than the Euclidean distance on the DNF dataset.

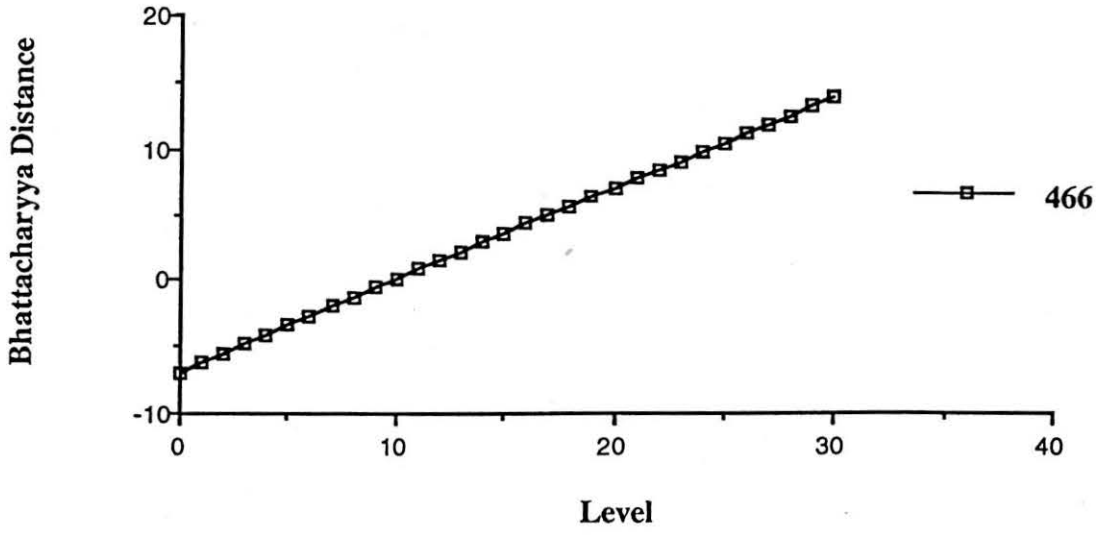

Fig. 25. Bhattacharyya distance / FSS / DNF database.

An inspection of Fig. 25 provides no indication of which feature subset is the most effective since the slope never changes. The feature subset at level nine turns out to be

0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 0

in which all nine relevant features are included. A designer of a system who knows approximately how many features in his dataset are relevant or who can only use a certain number of features due to complexity constraints, may find use for this evaluation function. He can simply choose the feature subset at the level he believes to be appropriate. Unfortunately, in applications such as intrusion detection, it is very unlikely that a system designer will know how many features are relevant. To select a level arbitrarily will undoubtedly cause some relevant features to be discarded or some irrelevant features to be included.

VI.B.3 Attack Database

Let us examine one final graph of the Bhattacharyya distance, this time with FSS on the simulated attack database (Fig. 26).

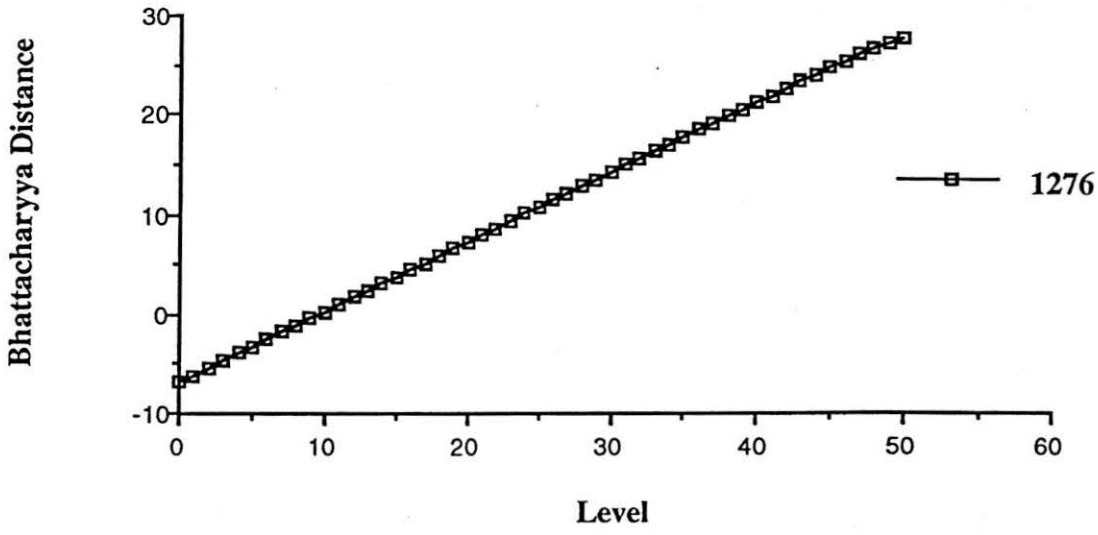

Fig. 26. Bhattacharyya distance / FSS / attack database.

Since there is again no way of determining from the figure which feature subset is the most effective, we count the number of relevant features at level 13. (There are 13 relevant features in the attack database.) The subset at level 13 is

1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0

which contains ten of the 13 relevant features. Examining only 1,276 states, this algorithm found all but three of the relevant features at level 13.

VI.B.4 Remarks

Other experiments were performed with the Bhattacharyya distance, but the same problems remained. Namely, the figures of the Bhattacharyya distance vs. level give no indication of the correct size to pick for the feature subset. As mentioned previously, this will probably cause irrelevant features to be chosen or relevant features to be discarded since a feature subset may be chosen that is either too large or too small. In addition, the Bhattacharyya evaluation function, like the Euclidean distance, measures the distance that a feature subset separates the various classes. (The Bhattacharyya distance is a probabilistic distance measure whereas the Euclidean distance is an interclass distance measure.) The results from the disjunction database where all five irrelevant features were chosen demonstrate that this is not always an effective measure of the value of a feature subset.

VI.C Bayesian Classifier

Missing feature values and non-numerical data cannot be handled by our implementation of the Bayesian classifier. Consequently, only results from the simulated databases are presented since the machine learning databases contain missing features and nominal data.

The Bayesian classifier, as noted in Section IV, only needs to be trained once, not for every feature subset, and is thus a modestly efficient evaluation function even though it is a classifier. This fact allowed us to take the average performance of the randomized algorithms (RGSS, GS, and SA) over ten runs. Since all the algorithms from this class obtain different results for each run, a better idea of their performance is obtained by taking

such an average. No averages were taken for the disjunction database since the optimal feature subset was found by all the algorithms in a single run.

VI.C.1 Disjunction Database

Fig. 27 shows the performance of the various search algorithms on the disjunction database using a Bayesian classifier as the evaluation function.

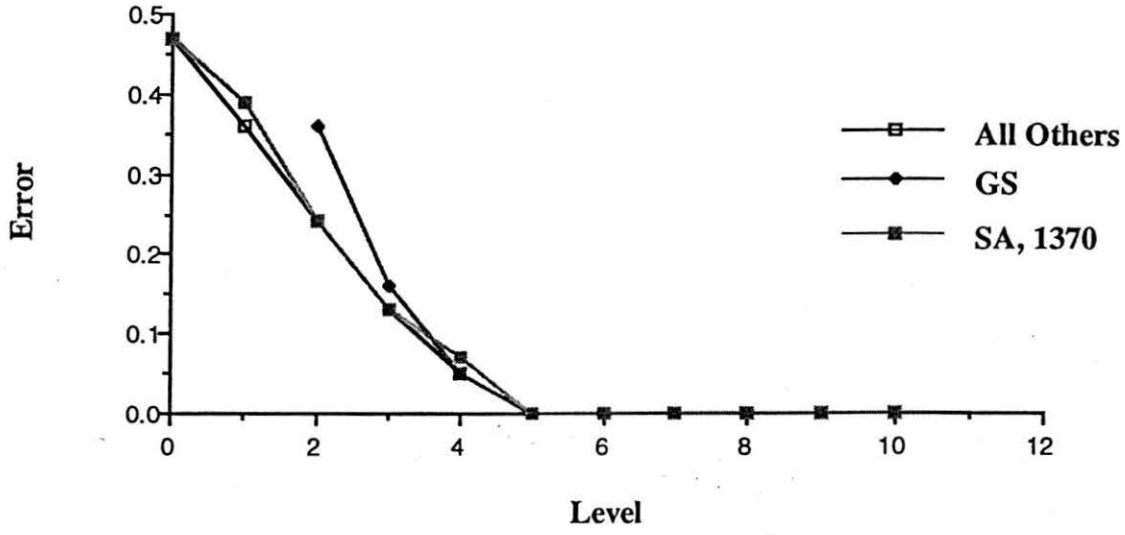

Fig. 27. Bayesian classifier / disjunction database.

All seven of the search algorithms found the feature subset consisting of all the relevant features, f1, ... , f5, and none of the irrelevant features. Only GS and SA strayed from the base line for smaller feature subsets due to their random nature. The line denoted by *All Others* is used to represent the results obtained by the other five search algorithms since they are identical. The number of states expanded was 1024 for ES and 56 for both FSS and BSS. The number of states expanded was not determined for the other search algorithms.

The primary advantage of feature selection here is not the decrease in error rate (i.e., the error is 0% for all levels from five to ten). Rather, it is the fact that we can obtain the same performance as the full feature set with five fewer features.

VI.C.2 Disjunctive Normal Form Database

The search algorithms were run on the DNF database using the Bayesian classifier and the results are shown in Fig. 28.

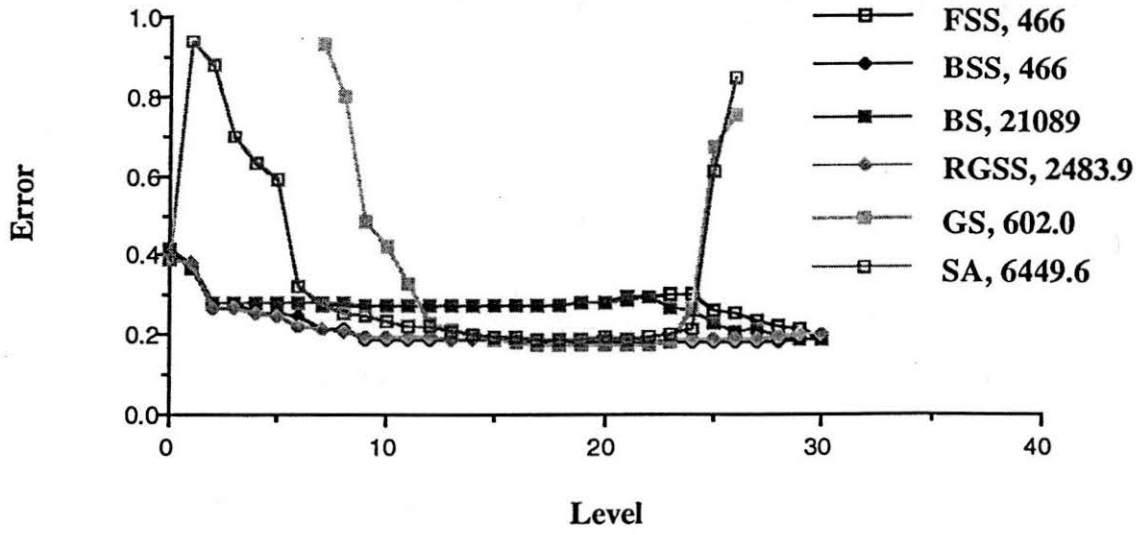

Fig. 28. Bayesian classifier / DNF database.

FSS and BS perform nearly identically on this database. (This is, in fact, not a surprising result since BS is based upon FSS.) Both algorithms improve rapidly with the addition of the first few features and then level off around a 30% error rate. GS and SA exhibit the chaotic behaviour near levels zero or 30. The reason for this is that these algorithms tend to focus their search in the middle levels of the search space and rarely venture towards the extremes. In other words, GS and SA do not examine many states with a number of included features near zero or 30, thereby decreasing the chances that a good subset will be found at these levels. In fact, some points for these algorithms near levels zero and 30 are missing altogether since *no* feature subsets corresponding to the levels at these points were visited. SA and GS do, however, obtain excellent error rates but at the cost of larger feature sets.

The best performance is obtained using either BSS or RGSS. Both steadily decrease their error rate until leveling off around level nine. This makes intuitive sense since the DNF database has nine relevant features. The algorithms correctly choose the best feature subsets at each level until no more relevant features can be added at level ten. The similar performance can be explained by the use of BSS in RGSS once a feature subset has been randomly generated. In effect, RGSS is just a variation on both BSS and FSS. A comparison of only these two searches is given in Fig. 29.

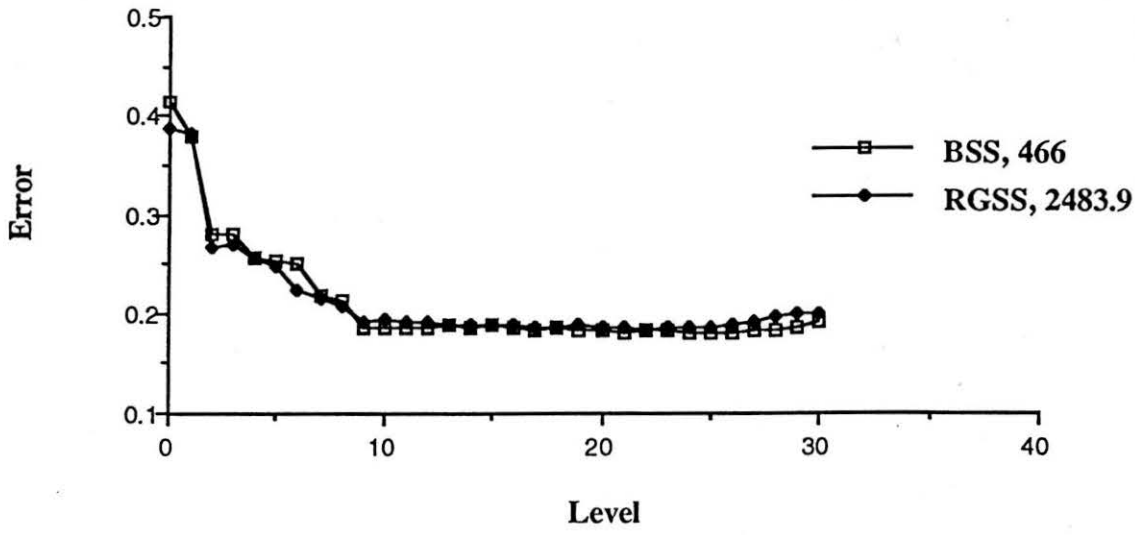

Fig. 29. Bayesian classifier / DNF database / BSS and RGSS.

No significant differences between the two algorithms are visible, although BSS appears to have a slightly lower error rate at most levels. The best subset found by BSS at level nine, the point at which the curve levels out, is

0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 0

which has an error rate of 18.6% and contains all nine of the relevant features and no irrelevant features.

The full feature subset in this database has 30 features and an error rate of 19.1%. Feature selection has not only reduced our error rate by 0.5%, but more importantly has reduced the number of features from 30 to nine.

VI.C.3 Attack Database

The results of running the search algorithms with the Bayesian classifier on the attack database are shown in Fig. 30.

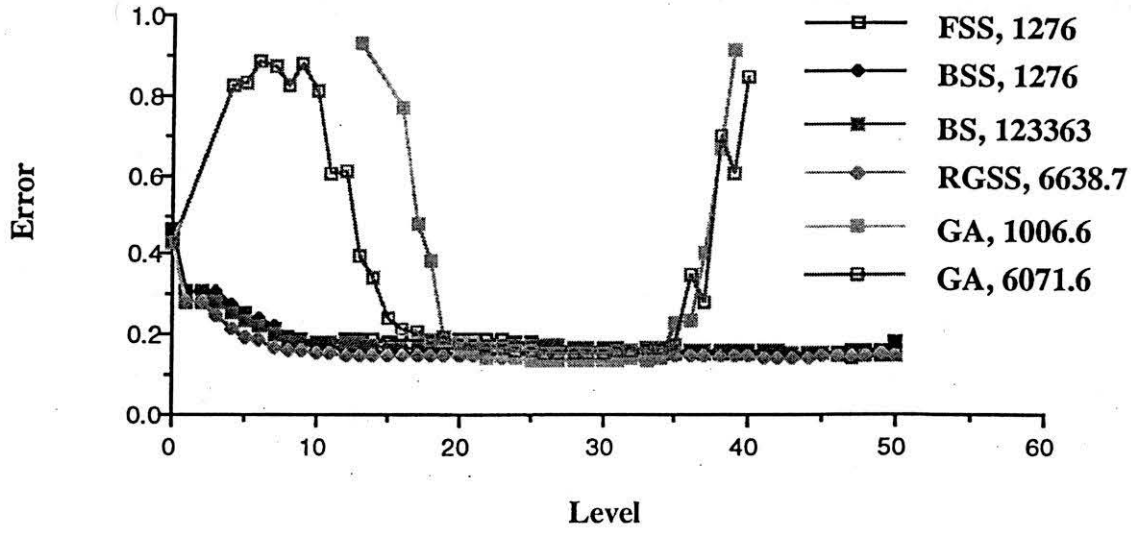

Fig. 30. Bayesian classifier / attack database.

Inspection of the figure shows that many of the search algorithms performed in a similar fashion on the DNF database. For instance, GS and SA again tend to focus their search around the middle levels and obtain low error rates. Also, BSS and RGSS appear to perform well as they did before. The main difference is the performance of FSS and BS which seem to do as well as any of the other algorithms. In an attempt to find the algorithm which gives the lowest error rate with the fewest features, Fig. 31 omits data points with error greater than 20% and excludes GS and SA.

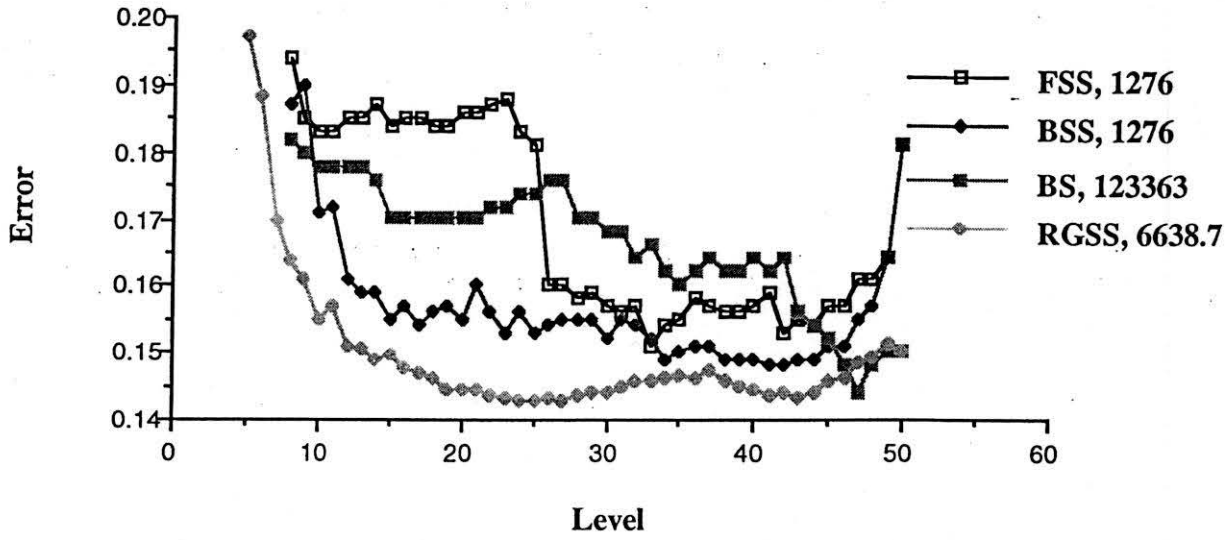

Fig. 31. Bayesian classifier / attack database / FSS, BSS, BS and RGSS.

RGSS is the obvious winner when a more detailed look at the graph is taken. It performs better than all the other algorithms at all levels with the exception of levels 47 - 50. Here is the global best subset found by RGSS:

1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0

It is at level ten and has an error rate of 13.6%. The subset contains ten of the 13 relevant features and no irrelevant features.

The full feature set has 50 features and an error rate of 18.1%. We have reduced the number of features from 50 to ten and decreased the error rate by 4.5%.

VI.C.4 Remarks

The experiments demonstrated that the Bayesian classifier can be used as an accurate evaluation function on databases with binary features. The best subsets found contained most of the relevant features and none of the irrelevant ones. In addition, BSS and RGSS were the top performing search algorithms and seem to work well with the Bayesian classifier.

VI.D Decision Tree Classifier

One of the primary differences between the decision tree classifier and the Bayesian classifier is speed. Recall that the Bayesian classifier only needs to be trained once, not for every feature subset, and it performs simple calculations to classify new instances. The decision tree classifier, on the other hand, needs to be retrained for *every* feature subset. In addition, it is a much more sophisticated algorithm than the Bayesian classifier because of the splitting rule (information gain) and pruning algorithm (pessimistic pruning) employed during creation of the tree[13]. The result is that the decision tree classifier is much slower than the Bayesian classifier, taking a few seconds to train and test for each feature subset compared to the milliseconds needed by the Bayesian classifier to test a subset. The runs of the randomized algorithms are therefore not averaged as they were with the Bayesian classifier.

The C4 algorithm [43] is considerably more robust than our implementation of the Bayesian classifier. In particular, C4 can easily handle missing feature values and non-numerical data (e.g., the names of files accessed) whereas our Bayesian classifier cannot. Consequently, we can apply the decision tree algorithm to real-world data, such as the machine learning databases, in addition to simulated data.

The results from experiments run on the disjunction database are not presented since all the search algorithms readily found the optimal feature subset.

VI.D.1 Disjunctive Normal Form Database

The decision tree classifier was used as the evaluation function in combination with the various search algorithms on the DNF database and the corresponding results are shown in Fig. 32.

---

[13] See [43] for details of the C4 decision tree classifier.

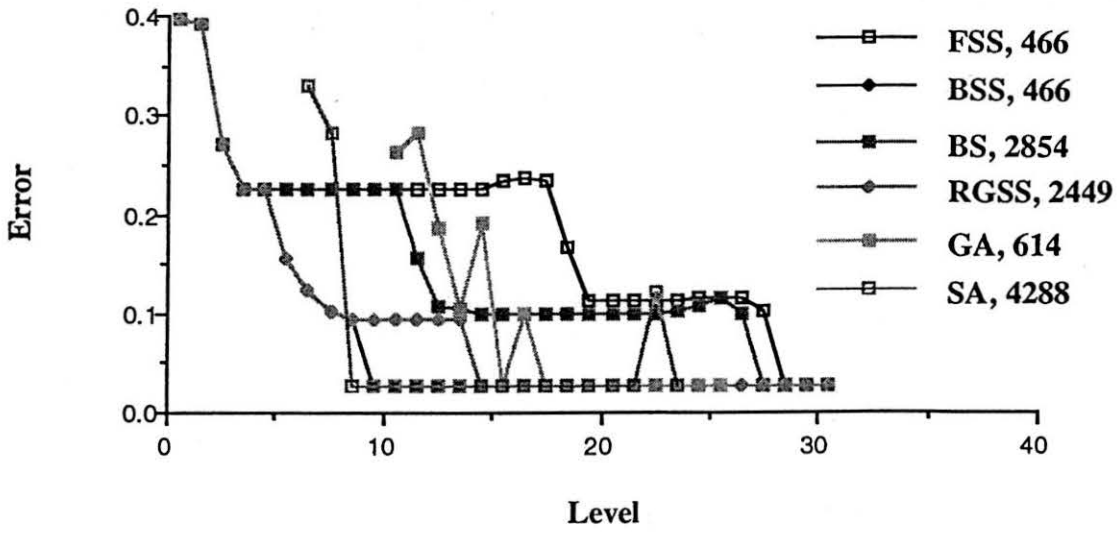

Fig. 32. Decision tree classifier / DNF database.

One thing to notice from Fig. 32 is the lower error rates obtained by the feature subsets when evaluated with the decision tree classifier, as compared to the error rates obtained by the Bayesian classifier in Fig. 28. The error rates appear to reach their lowest values at 2% using the decision tree classifier but they never drop below 18% using the Bayesian classifier. This does not imply that the C4 algorithm is better suited for feature selection. The only requirement we have is that an evaluation function recognize when one feature subset performs better than another. In other words, we are not interested in the absolute error rate, but the relative error rate between feature subsets.

Another difference between this evaluation function and the Bayesian classifier is the performance of SA. Although it obtains low error rates using the Bayesian classifier, the feature subsets with the low error rates contain many irrelevant features. Using the C4 algorithm, SA finds the subset that not only provides the lowest error rate, but also contains fewer features (eight) than all other subsets with identical error rates. The BSS algorithm, as expected, also performs well and Fig. 33 is a comparison of only BSS and SA.

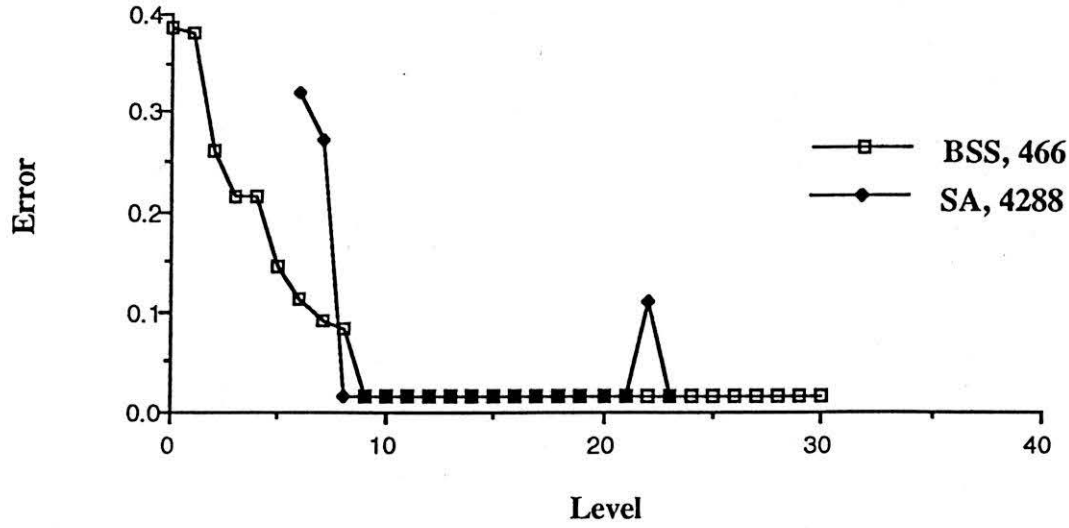

Fig. 33. Decision tree classifier / DNF database / BSS and SA.

SA performs slightly better since it found the feature subset providing the lowest error rate with the fewest features. Here is that subset:

1 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0

It obtains an error rate of only 1.6%, but surprisingly contains only three of the nine relevant features in the database. The reason for this anomaly is not entirely clear. It is possible that, for the specific data used in training and testing, the three relevant features found by the algorithm give the lowest error rate. Unfortunately, the algorithm has not done what we want it to do; namely, find the best features in general, not just for the particular instances in the training and testing set.

Given this new information, let us examine the best subset found by the BSS search algorithm:

0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 0

It has all nine of the relevant features and no irrelevant features with an error rate of only 1.6%. Although the error rate is the same as that obtained by SA, the fact that BSS found all the relevant features is an indication that it is performing the feature selection task much more effectively. Also, SA examines nine times as many states as BSS. Given the added complexity of SA and the fact that BSS found all nine of the relevant features, BSS appears to be a much better choice on this database when using the C4 decision tree as the evaluation function.

The full feature subset obtained an error rate of 1.6%, identical to the error rate of the best feature subset found. However, feature selection has reduced the number of features from 30 to nine.

VI.D.2 Attack Database

Fig. 34 shows the accuracy of the search algorithms with the decision tree classifier on the attack database.

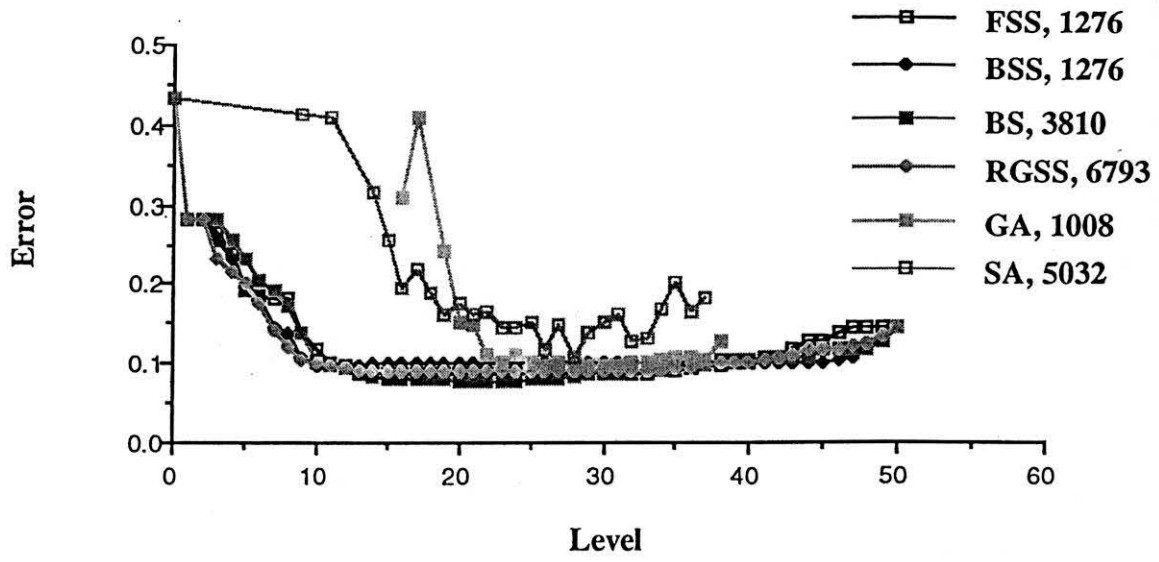

Fig. 34. Decision tree classifier / attack database.

GS and SA do not find small feature subsets of low error rates. More interesting is the nearly exact resemblance of the curves of the other four algorithms. Let us take a closer look at these algorithms (FSS, BSS, BS, and RGSS) in Fig. 35.

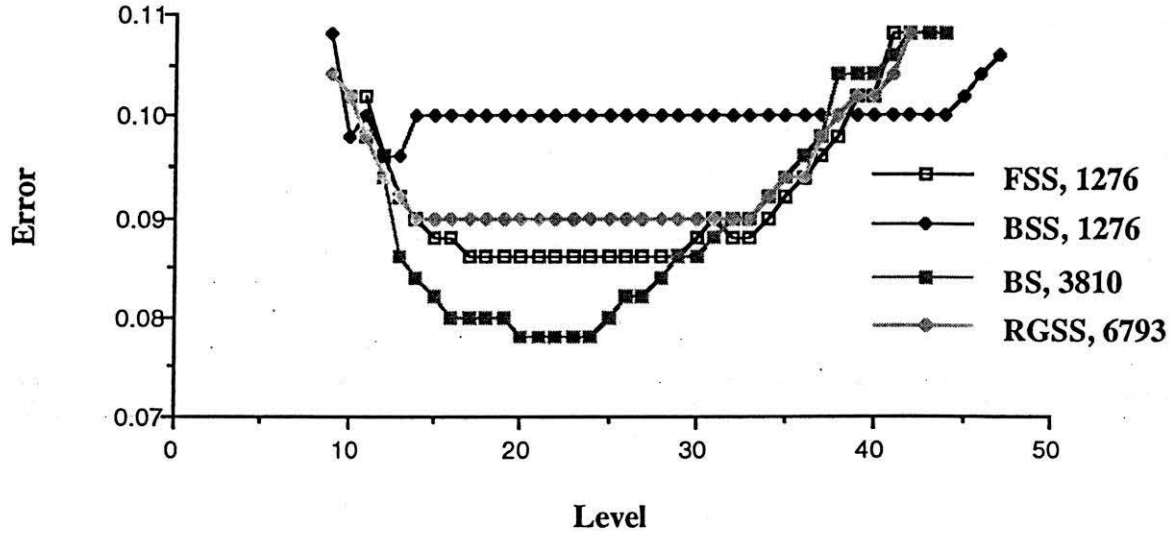

Fig. 35. Decision tree classifier / attack database / FSS, BSS, BS and RGSS.

BS outperforms all the other algorithms and for the first time, FSS does better than BSS. From the figure, we see that the feature subset with the lowest error rate occurs at level 20 of BS. However, if we allow only a slightly larger error rate, we can remove seven features to obtain the following highly accurate feature subset of only 13 features

1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 1

which has an error rate of 8.6% and contains ten of the 13 relevant features.

14.4% was the error rate obtained by the full feature subset. The best feature subset found by feature selection improves the error rate by 5.8% and reduces the number of features by 37 since there are 50 features in the attack database..

VI.D.3 Congressional Voting Records Database

The decision tree classifier is used as the evaluation function for the various search algorithms on the voting records database in Fig. 36.

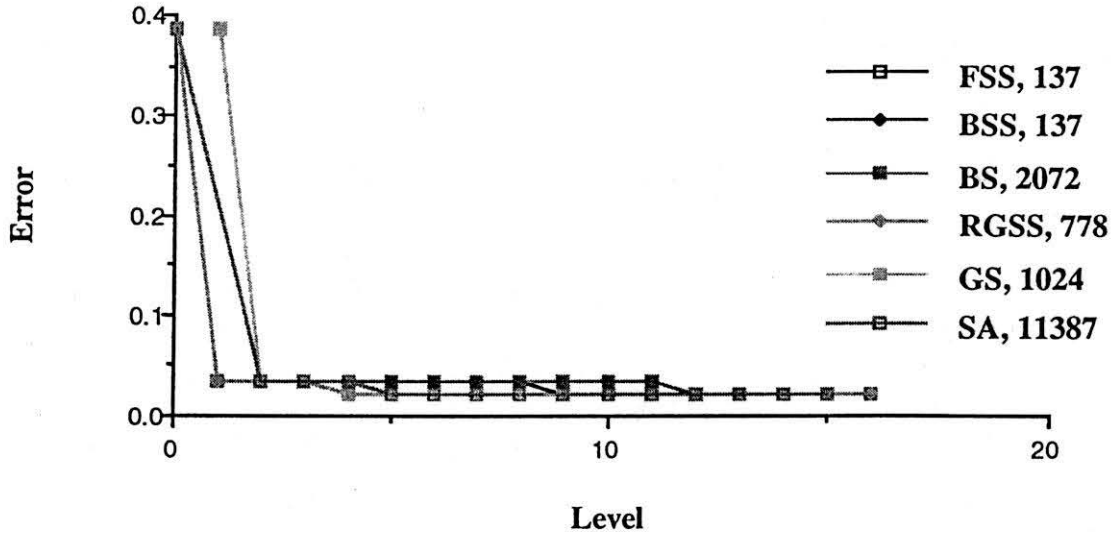

Fig. 36. Decision tree classifier / voting records database.

With only one feature, most of the search algorithms obtained an error rate of 3% implying that one of the voting issues is very good at distinguishing between democrats and republicans. The error rate improved again for most of the algorithms around four features. A more detailed look at the algorithms is shown in the following figures (Figs. 37 through 39).

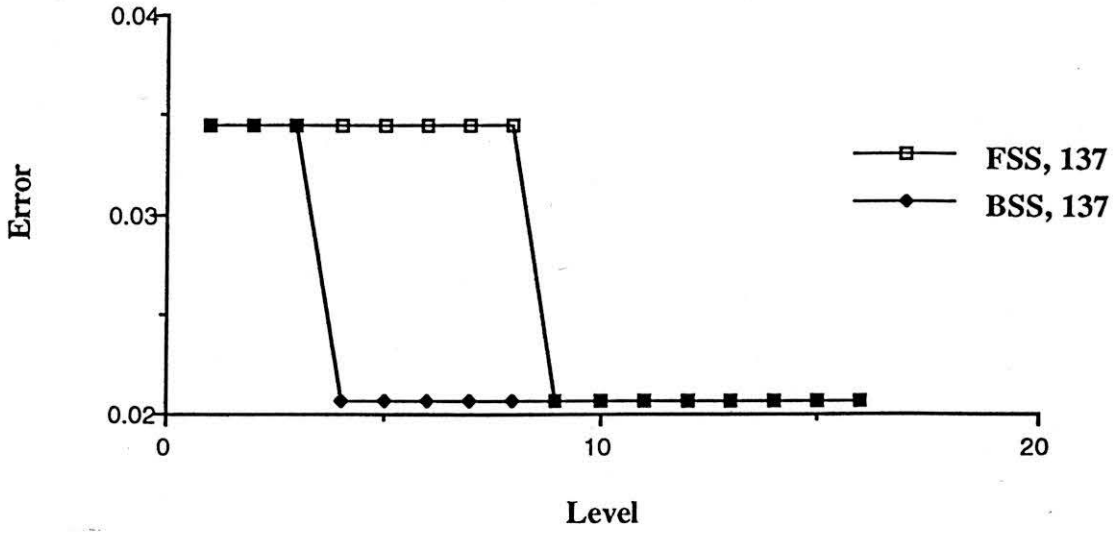

Fig. 37. Decision tree classifier / voting records database / FSS and BSS.

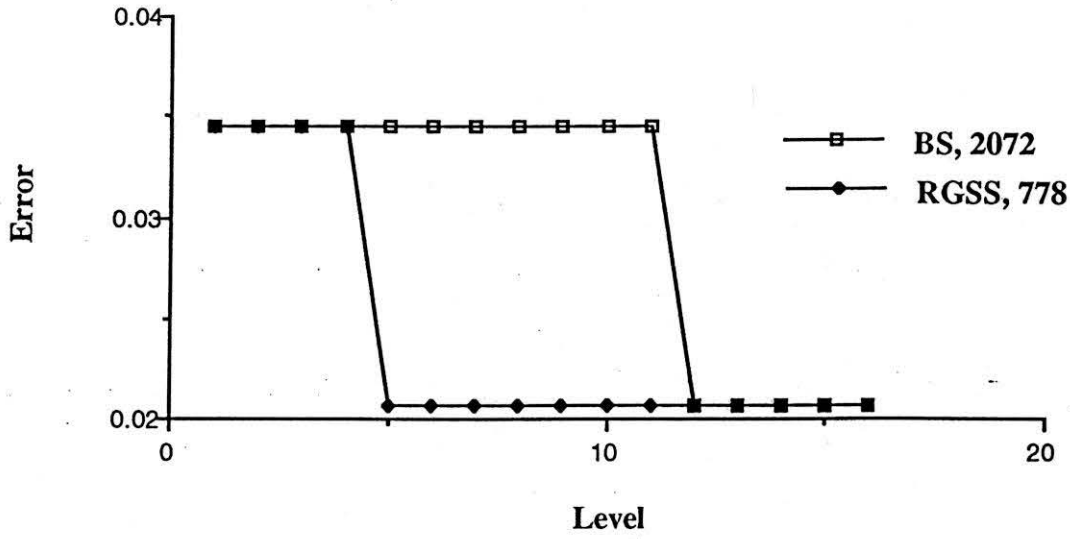

Fig. 38. Decision tree classifier / voting records database / BS and RGSS.

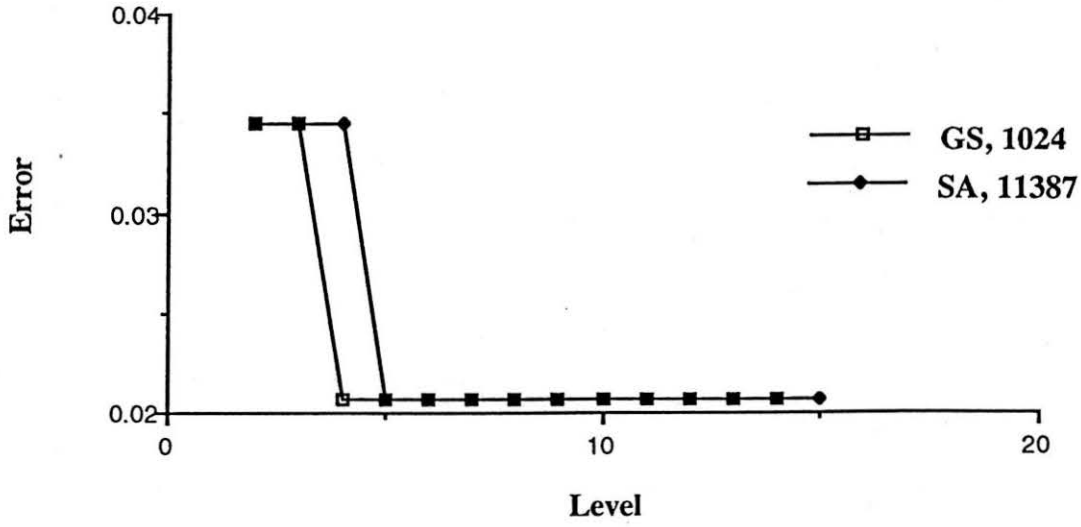

Fig. 39. Decision tree classifier / voting records database / GS and SA.

The lowest error rate of 2.1% is obtained at level four by both BSS and GS. However, the lower complexity of BSS, 137 to 1024 states expanded, makes it a better choice than GS.

Here are the features subsets found by BSS at levels one and four:

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0

with error rates of 3.4% and 2.1%, respectively. The lone feature in the first feature subset refers to the vote on whether or not to freeze the fees of physicians. The republicans tended to vote yes and the democrats tended to vote no. The four features in the second subset are 'adoption of the budget resolution', 'physician fee freeze', 'anti-satellite test ban', and 'synthetic fuels corporation cutback'.

The error rate was 2.1% for the full feature subset. Using feature selection, we obtained the same error rate at level four giving us a decrease of 12 features (there are 16 features in the voting records database) with no loss in performance.

VI.D.4 Soybean Disease Database

The results of running the various search algorithms on the soybean disease database with the decision tree classifier as the evaluation function are shown in Fig. 40.

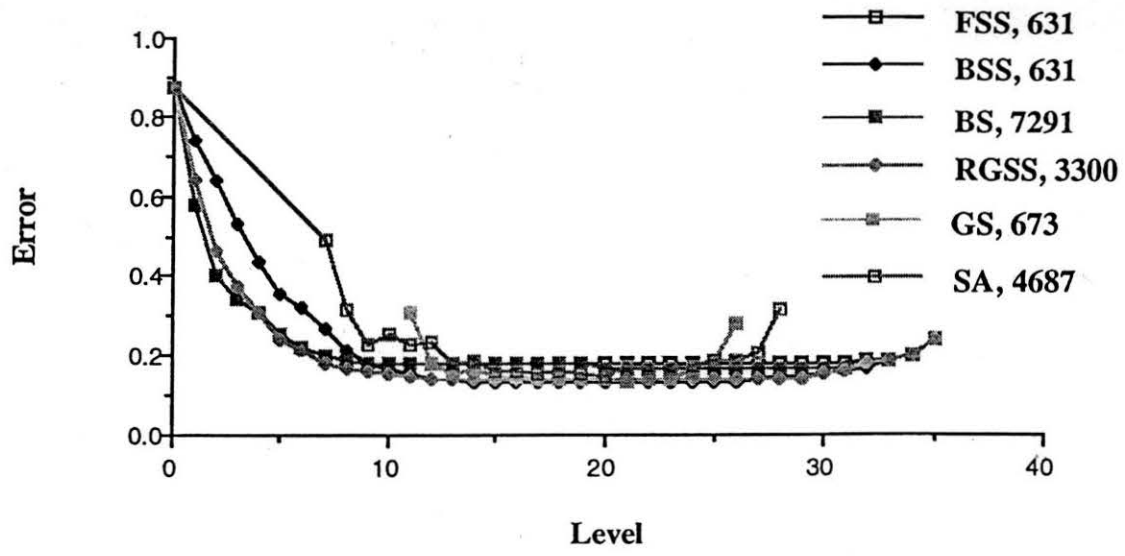

Fig. 40. Decision tree classifier / soybean disease database.

Other than GS and SA, the algorithms all follow the same basic curve. Let us take a closer look at FSS, BSS, BS, and RGSS in Fig. 41 to determine which one is the most effective.

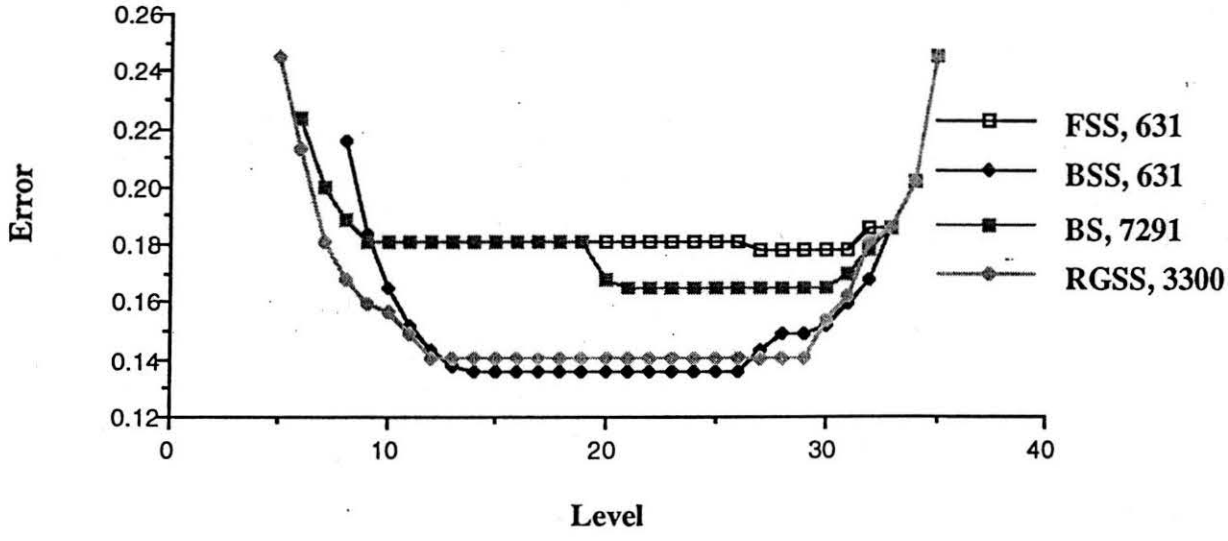

Fig. 41. Decision tree classifier / soybean disease database / FSS, BSS, BS & RGSS.

BSS and RGSS are the most effective algorithms on this database. The best subset appears to be that found by BSS at level 14:

0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 1

which has an error rate of 13.6%. The features included in this subset are described in Section 2 of the Appendix Machine Learning Database Features. It would be interesting to see whether or not an expert in the field of soybean diseases would agree that these are the most relevant features.

We can again see the benefits of feature selection by comparing the best subset found to the full feature subset which has a 24.5% error rate and 35 features. Feature selection improved the error rate by 10.9% and reduced the number of features from 35 to 14.

VI.D.5 Heart Disease Database

Fig. 42 compares the performance of the search algorithms on the heart disease database when the decision tree classifier is the evaluation function.

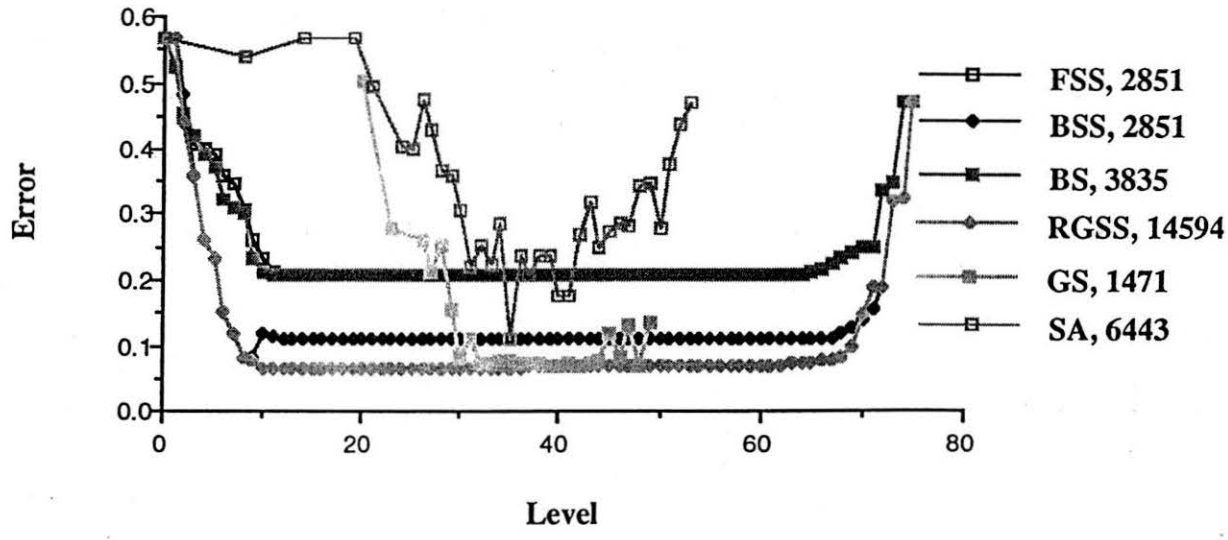

Fig. 42. Decision tree classifier / heart disease database.

RGSS and BSS are clearly superior to the other search algorithms on the heart disease database. Of these two, Fig. 43 shows the superiority of RGSS over BSS.

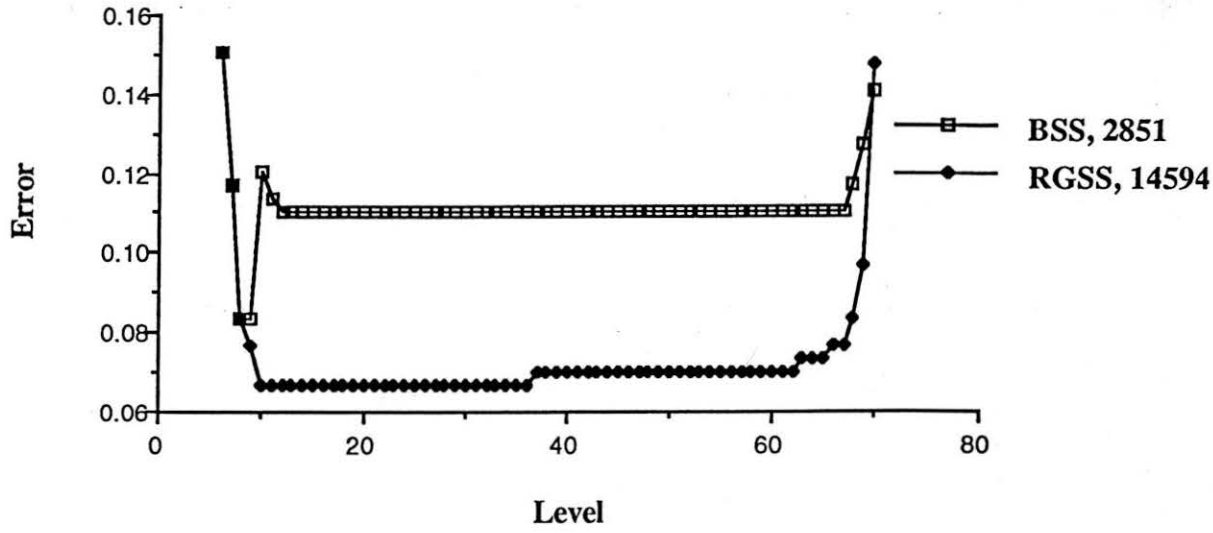

Fig. 43. Decision tree classifier / heart disease database / BSS and RGSS.

The best feature subset found by BSS has an error rate of 8.4% at level eight, but RGSS found a subset with an error rate of only 6.7% at level ten. The subset found by RGSS is

< 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 >

The features included in this subset are described in Section 3 of the Appendix Machine Learning Database Features.

Feature 38, which is included, says that whether or not you feel pain in your chest when exercising is an important factor in determining heart disease. Also, the year in which cardiac catheterization[14] was performed seems to be a valuable feature. The fact that the process of catheterization, an observatory procedure, is linked to the presence of heart disease should dictate that it be used only when absolutely necessary. The most interesting result, however, is that eight of the ten most important features are whether or not certain vessels have experienced greater than a 50% narrowing in diameter. The implication is that the majority of tests that are performed and questions that are asked in an attempt to discover heart disease may not be worthwhile. A simple subset of these tests, consisting primarily of the determination of narrowing in certain vessels, appears to be sufficient in order to make predictions about heart disease.

Comparing with the full feature subset, which had an error rate of 46.8%, we have improved the error rate by 40.1% and reduced the number of features from 75 to ten. This is truly an outstanding result and attests to the need for effective feature selection algorithms.

VI.D.6 Remarks

---

[14] Cardiac catheterization is a procedure which begins by pushing a tube through a vein or artery into the heart. Once in the heart, a dye is injected through the tube for the purpose of taking an angiogram, an Xray view of the radiopaque dye. The Xray is examined for any anomalies such as a faulty valve or irregular blood flow.

A decision tree classifier is a wise choice for an evaluation function. Given a good set of features, the C4 algorithm invariably produced a low error rate. The error rate of the classifier also reflected poor sets of features by producing high error rates.

VII. Conclusions

- *backward sequential selection had the best overall performance of the search algorithms.*

In the eight different comparisons of the search algorithms (five using the decision tree classifier and three using the Bayesian classifier), BSS outperformed all the other algorithms four times. It was also very close to the performance of the best algorithms in the remaining comparisons. What is perhaps most surprising is that BSS obtains such good results examining such a small fraction of the search space ($n^2/2^n$). Except for GS on the attack and heart disease databases, BSS and FSS examine the fewest number of states of the search algorithms.

The reason for the effectiveness of BSS is not clearly understood. One possible explanation for this is that, since BSS begins its search from the full feature set, it has information on all of the features from the outset. In other words, BSS is able to analyze the effect of removing a feature from the full feature set which gives it immediate knowledge of how the feature performs *in combination* with all the other features. Contrast this to FSS which only analyzes the *individual* performance of features during the initial step of the algorithm. It therefore does not know how a feature would affect performance when used as part of a group of features.

The use of several types of evaluation functions, based upon both intrinsic properties of the data and classification error rate, lead to this observation:

- *evaluation functions based upon the error rate are far superior to those based upon intrinsic properties of the data*[15].

The experiments using either the Bhattacharyya distance or the Euclidean distance as the evaluation function demonstrated two major drawbacks to the intrinsic measures. The first is that, since the intrinsic measures are monotonic, the feature subset with the highest value will always be the full feature subset. In other words, there is no point at which adding more features actually reduces the value returned by the evaluation function. The lack of such a point forces the user of the feature selection algorithm to either guess the right size of the feature subset or base the size on his complexity constraints. The result is likely to be a feature set which is too small (i.e., is missing relevant features) or is too large (i.e., contains irrelevant features).

The second problem with the intrinsic measures is that the method of evaluating features may have little to do with how relevant the features are to the classification task. Recall from the experiments on the disjunction database that both the Bhattacharyya distance and the Euclidean distance found the feature subset containing all five *irrelevant* features as the best subset. The reason for this was that the difference of feature values between the two classes (the larger the difference the better the feature is supposed to be) had nothing to do with the learned function.

Those measures using the classification error rate, on the other hand, suffer from neither of these problems. The error rate tends to be lowest when all the relevant features are present and all the irrelevant ones are excluded. The reason for this is that classifiers tend to perform worse when irrelevant or noisy features are used in training and testing. This can give the user of the algorithm a clear point at which either adding or removing features increases the error rate. (The error rate may increase when relevant features are removed or

---

[15] This result is not surprising since the evaluation functions based upon intrinsic properties of the data can be thought of as approximating the error rate.

irrelevant features are added.) The feature subset at this point can then be chosen as the best features subset.

The intrinsic measures suffered from faulty assumptions about how to determine the worth of features or feature subsets. Classifiers make no such assumptions and only rely on how well the features aid the classification task. Since classification is precisely what the features are to be used for, this is intuitively (and the experimental results show that this is) the most effective method of evaluating feature subsets.

- *the search algorithm with the best performance depended on the database and the evaluation function.*

Although BSS outperformed all the search algorithms on four of the eight comparisons, RGSS, SA, BS, and GS all were the best or nearly the best with certain database / evaluation function combinations: RGSS was the best for two of the comparisons and nearly the best on two others; SA was nearly the best on the DNF database using the decision tree classifier; BS was the best on the attack database with the C4 algorithm; and GS was nearly the best on the voting records database when the decision tree algorithm was the evaluation function. The fact that different algorithms performed well in different situations implies that

- *several combinations of a search algorithm and an evaluation function should be used to determine the optimal feature subset for a particular database.*

It is recommended that at least BSS and RGSS be used with two different error rate evaluation functions to arrive at a reasonably optimal subset.

We would also like to infer from the experiments the appropriate search algorithms and evaluation functions to use when selecting features for IDSs. For this purpose, let us examine the results from the simulated attack database and the heart disease database. Using the Bayesian classifier on the attack database, RGSS was the top performer. When the C4 decision tree was the evaluation function on the same database, BS obtained the best feature subset. Finally, RGSS was the best search algorithm on the heart disease database using the C4 algorithm. This, in addition to the generally good performance of both BSS and the error rate evaluation functions, allows us to conclude that

- *random generation plus sequential selection, backward sequential selection, and beam search should be used with at least two different error rate evaluation functions to determine the optimal set of features to be used in an intrusion detection system.*

VIII. Future Directions

There are several areas for extending the work presented in this paper.

- *bias genetic and simulated annealing search algorithms towards small feature subsets.*

The main problem with GS and SA is not that they do not find subsets with low error rates. On the contrary, they almost always find feature subsets with the lowest or close to the lowest error rates. The problem is that they don't find feature subsets of low error rates which have a small number of features. In fact, the results shown in the figures clearly demonstrate that these algorithms focus their search in that part of the space where subsets contain approximately one-half of the total features. If the algorithms can be reworked so that the search leans toward those feature subsets with a small number of features, low error rates *and* small feature subsets may be obtained.

There are a couple of ways that GS can be biased towards small feature subsets. The first method would involve a slight modification to the mutation operator which, in the current implementation, flips 1's to 0's and 0's to 1's in feature subsets with equal probability (the mutation probability). The change would simply require having a higher probability of flipping 1's to 0's than 0's to 1's so that feature subsets would tend to decrease in size. Another way of installing the bias would be by creating an initial population consisting of primarily small feature subsets. This could be done by initializing the population to contain only empty feature subsets or subsets with only a few features. Currently, each bit in the initial population is '1' with 50% probability which on the average creates an equal number of 1's and 0's in the feature subsets.

SA might lend itself to this bias in the transform operator. The implementation used in our experiments randomly switched the bits of the feature subset, corresponding to the current state in the search, to generate a new state. The transform operator could be modified in a fashion similar to the mutation operator of genetic search so that feature subsets of smaller sizes are more likely to be produced by the transform operator.

- *modify beam search so that the underlying algorithm is backward instead of forward sequential selection.*

BS in our implementation expands its search from the state at the head of the queue by performing FSS. Each way of adding one feature to the subset at the head of the queue is attempted with the resulting states being inserted into their proper positions in the queue. If the queue length is set at one, BS performs *identically* to FSS. During the experiments, it became extremely apparent that BSS was a better choice of search algorithm than FSS as it performed better in all but one of the experiments. It is expected that the performance of BS would improve significantly if the heart of the algorithm were BSS instead of FSS. In this case, the worst performance of the algorithm would be when the queue length were one, in essence BSS. For larger queue sizes, it is expected that the performance of the algorithm would improve on (and certainly be no worse than) BSS.

Other methods of expanding the feature subset at the head of the queue are possible and should be tested as well. For instance, we might use something like the transform operator of SA which randomly flips bits in the current feature subset to expand the search.

- *test search algorithms on a real-world attack database.*

The conclusion referring to which search algorithms would work well on attack data were made based upon experiments on a simulated attack database and a heart disease database. Stronger statements about the proper search algorithms and evaluation functions to use for IDSs can be made if experiments are performed on actual attack data.

• *implement an estimated or incremental error rate evaluation function.*

Neither an estimated nor an incremental error rate evaluation function were implemented for the experiments. With such an implementation, it may be possible to use a near exhaustive search (e.g., B&B) and still obtain the benefits of a using an error rate evaluation function. The slowness of training and testing a classifier for each feature subset prohibits such a combination for standard error rate evaluation functions.

• *implement other search algorithms for a more complete comparison.*

BDS, classical B&B, AMB&B, and PQSS were not implemented[16]. Implementing these algorithms would allow a more complete testing of existing feature selection search algorithms.

---

[16] However, at least one algorithm from each of their respective generations was implemented and evaluated. For instance, BS belongs to the same generation as BDS (both are neo-classical algorithms) and was implemented.

Acknowledgments

The author wishes to thank Dr. Biswanath Mukherjee for his close supervision and helpful comments during the course of this work. Others whose contributions are gratefully acknowledged are Dr. Armand Prieditis and Dr. Karl Levitt. Thanks also goes to the following institutions for supporting this research: the University of California which provided support under Grant No. 91-110 of the MICRO Program, the Hewlett Packard Company, and the Department of Defense.

References

[1] Stephen E. Smaha, "Haystack: An intrusion Detection System", *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, Dec. 1988.

[2] Harold S. Javitz and Alfonso Valdes, "The SRI IDES Statistical Anomaly detector", *Proceedings of the 1991 IEEE Symposium on security and Privacy*, Oakland, CA, May 1991.

[3] Hank S. Vaccaro and Gunar E. Liepins, "Detection of Anomalous Computer Session Activity", *Proceedings of the 1989 Symposium on Research in Security and Privacy*, pp. 280-289, Oakland, CA, May 1989.

[4] L. Todd Heberlein, Biswanath Mukherjee, Karl Levitt and Gihan Dias, "Towards Detecting Intrusions in a Networked Environment", *Proceedings of the 14th Department of Energy Computer Security Group Conference*, pp. 47-65, Concord, CA, May, 1991.

[5] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur, "DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype", *Proceedings of the National Computer Security Conference*, Oct. 1991.

[6] Sholom Weiss and Casimir Kulikowski, *Computer Systems That Learn*, Morgan Kaufman Publishers, Inc., San Mateo, CA, 1991.

[7] Peter C. Jurs and Thomas L. Isenhour, *Chemical Applications of Pattern Recognition*, John Wiley and Sons, 1975.

[8] Kou-Yuan Huang and King-Sun Fu, "Detection of Seismic Bright Spots Using Pattern Recognition Techniques", *Handbook of Geophysical Exploration, Section I, Seismic Exploration, Vol. 20, Pattern Recognition and Image Processing*, pp. 263-301, Geophysical Press Limited, London, UK, 1987.

[9] Tzay Y. Young and King-Sun Fu, editors, *Handbook of Pattern Recognition and Image Processing*, Academic Press, Inc., Orlando, FL, 1986.

[10] Terrance J. Goan, *Towards a Dynamic System for Accountability and Intrusion Detection in a Network Environment*, M.S. thesis, University of California at Davis, Graduate Group in Computer Science, 1992.

[11] Wojciech W. Siedlecki, *Feature Selection for Large Scale Problems*, Ph.D. dissertation, University of California at Irvine, 1988.

[12] Iman Foroutan, *Feature Selection for Piecewise Linear Classifiers*, Ph.D. dissertation, University of California at Irvine, 1985.

[13] William S. Meisel, *Computer Oriented Approaches to Pattern Recognition*, Academic Press, Inc., New York, NY, 1972.

[14] Theresa F. Lunt, John Van Horne, and Lawrence R. Halme, "Analysis of Computer System Audit Trails - Feature Identification and Selection", *Sytek Technical Report TR-85012*, Mountain View, CA, Dec. 1985.

[15] Manabu Ichino and Jack Sklansky, "Optimum Feature Selection by Zero-One Integer Programming", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-14, No. 5, pp. 737-746, September/October 1984.

[16] Paul Helman and Gunar E. Liepins, "Foundations of Intrusion Detection", to appear in *Proceedings of the 5th Computer Security Foundations Workshop*, Franconia.

[17] Kenji Kira and Larry A. Rendell, "The Feature Selection Problem: Traditional Methods and a New Algorithm", to appear in *Proceedings of the 1992 AAAI Conference*, San Jose, CA, July 12-17, 1992.

[19] Wojciech W. Siedlecki and Jack Sklansky, "A Note on Genetic Algorithms for Large-Scale Feature Selection", *Pattern Recognition Letters 10*, pp. 335-347, Elsevier Science Publishers, B.V., North-Holland, Nov. 1989.

[19] Christopher J. Matheus, "Adding Domain Knowledge to SBL through Feature Construction", *Proceedings of the 8th National Conference on AI*, Vol. 2, pp. 803-808, 1990.

[20] Iman Foroutan, "Feature Selection for Automatic Classification of Non-Gaussian Data", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-17, No. 2, pp. 187-198, March/April 1987.

[21] Giulia Pagallo, "Learning DNF by Decision Trees", *Proceedings of the 11th International Joint Conference on AI*, Vol. 1, pp. 639-644, 1989.

[22] Wojciech W. Siedlecki and Jack Sklansky, "On Automatic Feature Selection", *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 2, No. 2, pp. 197-220, 1988.

[23] Patrenahalli M. Narendra and Keinosuke Fukunaga, "A Branch and Bound Algorithm for Feature Subset Selection", *IEEE Transactions on Computers*, Vol. C-26, No. 9, pp. 917-922 Sept. 1977.

[24] Giulia Pagallo and Davis Haussler, "Boolean Feature Discovery in Empirical Learning", *Machine Learning*, Vol. 5, No. 1, pp. 71-99, Kluwer Academic Publishers, Boston, USA, 1990.

[25] Wojciech W. Siedlecki and Jack Sklansky, "Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition", obtained from authors at University of California at Irvine.

[26] Nick Littlestone, "Learning Quickly when Irrelevant Attributes Abound: a New Linear-Threshold Algorithm", *Machine Learning*, Vol. 2, No. 4, pp. 285-318, Kluwer Academic Publishers, Boston, USA, 1988.

[27] Mike James, *Classification Algorithms*, Collins Professional and Technical Books, London, UK, England, 1985.

[28] George R. Robertson, "Population Size in Classifier Systems", *Proceedings of the Fifth International Conference on Machine Learning*, pp. 142-152, 1988.

[29] David E. Goldberg, "Sizing Populations for Serial and Parallel Genetic Algorithms", *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pp. 70-79, 1989.

[30] Gilbert Syswerda, "Schedule Optimization Using Genetic Algorithms", *Handbook of Genetic Algorithms*, pg. 347, Van Nostrand Reinhold, New York, NY, 1991.

[31] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection and Genetics*, MIT Press, Cambridge, Massachusetts, 1992.

[32] John J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16, No. 1, pp. 122-128, Jan./Feb. 1986.

[33] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of State Calculation by Fast computing Machines", *Journal of Chemical Physics*, Vol 21, pp. 1087-1092, 1953.

[34] K. A. DeJong, *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. dissertation, Department of Computer and Communication Sciences, University of Michigan, 1975.

[35] Wray Buntine,"About the IND Tree Package", RIACS, NASA Ames Research Center, Moffet Field, CA, 1991.

[36] *Congressional Quarterly Almanac,* 98th Congress, Second Session, 1984, Volume XL, Congressional Quarterly Incorporated, Washington, D.C., 1985.

[37] R.S. Michalski and R. L. Chilausky, *Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis*, Internationaly Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.

[38] J. R. Quinlan, "Induction of Decision Trees", *Machine Learning*, Vol. 1, No. 1, pp. 81-106, 1986.

[39] R. S. Michalski and R. Stepp, "Learning from Observation: Conceptual Clustering", R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Editors), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufman Publishers, Inc., San Mateo, CA, 1983.

[40] Phil Laird, "TDAG: An Algorithm for Learning to Predict Discrete Sequences", *Proceedings of the Computational Learning and Natural Learning Workshop*, 1991.

[41] Ken Mock, private communication, June 1992.

[42] Armand Prieditis, private communication, January 1992.

[43] J. R. Quinlan, "Simplifying Decision Trees", B. Gaines and J. Boose (Editors), *Knowledge Acquisition for Knowledge Based Systems*, pp. 239-252, Academic Press, London, 1988.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, pp. 671-680, May 1983.

[45] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, 1989.

[46] B. Hajek, "Cooling Schedules for Optimal Annealing", *Mathematics of Operations Research*, Vol. 13, pp. 311-329, 1988.

Appendix

Machine Learning Database Features

This appendix describes the classes and features in the various machine learning databases. Also shown are the features included in the best feature subset found by the search algorithms.

1. Congressional Voting Records Database

This dataset contains votes on key issues, as determined by the Congressional Quarterly Almanac (CQA), for each of the U.S. House of Representatives Congressman during the 98th Congress, Second Session, 1984.

Classes: republican and democrat.

Features:

1. handicapped infants: y or n
2. water project cost sharing: y or n
3. adoption of the budget resolution: y or n
4. physician fee freeze: y or n
5. El Salvador aid: y or n
6. religious groups in schools: y or n
7. anti-satellite test ban: y or n
8. aid to Nicaraguan contras: y or n
9. MX-Missile: y or n
10. immigration: y or n
11. synfuels corporation cutback: y or n
12. education spending: y or n
13. superfund right to sue: y or n
14. crime: y or n
15. duty free exports: y or n
16. export administration act South Africa: y or n

2. Soybean Diseases Database

Classes: diaporthe stem canker, charcoal rot, rhizoctonia root rot, phytophthora rot, brown stem rot, powdery mildew, downy mildew, brown spot, bacterial blight, bacterial pustule, purple seed stain, anthracnose, phyllosticta leaf spot, alternarialeaf spot, frog eye leaf spot, diaporthe pod & stem blight, cyst nematode, 2 4 d injury, and herbicide injury.

Features: Note that the value of 'dna' means does not apply.

1. date: april, may, june, july, august, september, october, ?.
2. plant stand: normal, lt normal, ?.
3. precipitation: lt norm, norm, gt norm, ?.
4. temperature: lt norm, norm, gt norm, ?.
5. hail: yes, no, ?.

6. crop history: diff lst year, same lst yr, same lst two yrs, same lst sev yrs, ?.
7. area damaged: scattered, low areas, upper areas, whole field, ?.
8. severity: minor, pot severe, severe, ?.
9. seed tmt: none, fungicide, other, ?.
10. germination: 90-100%, 80-89%, lt 80%, ?.
11. plant growth: norm, abnorm, ?.
12. leaves: norm, abnorm.
13. leafspots halo: absent, yellow halos, no yellow halos, ?.
14. leafspots marg: w s marg, no w s marg, dna, ?.
15. leafspot size: lt 1/8, gt 1/8, dna, ?.
16. leaf shread: absent, present, ?.
17. leaf malf: absent, present, ?.
18. leaf mild: absent, upper surf, lower surf, ?.
19. stem: norm, abnorm, ?.
20. lodging: yes, no, ?.
21. stem cankers: absent, below soil, above soil, above sec nde, ?.
22. canker lesion: dna, brown, dk brown blk, tan, ?.
23. fruiting bodies: absent, present, ?.
24. external decay: absent, firm and dry, watery, ?.
25. mycelium: absent, present, ?.
26. int discolor: none, brown, black, ?.
27. sclerotia: absent, present, ?.
28. fruit pods: norm, diseased, few present, dna, ?.
29. fruit spots: absent, colored, brown w/blk specks, distort, dna, ?.
30. seed: norm, abnorm, ?.
31. mold growth: absent, present, ?.
32. seed discolor: absent, present, ?.
33. seed size: norm, lt norm, ?.
34. shriveling: absent, present, ?.
35. roots: norm, rotted, galls cysts, ?.

Features included in the best feature subset found by the feature selection algorithms:

2, 3, 8, 11, 15, 16, 19, 21, 24, 27, 30, 31, 32, and 35.

3. Heart Disease Database

Institutions and Principal Investigators:

1. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
2. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.

3. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
4. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Classes: 0, 1, 2, 3, and 4. The absence of heart disease is '0' and severe heart disease is '4'. This class feature is known as the angiographic disease status.

Features:

1 id: patient identification number

2 ccf: social security number (I replaced this with a dummy value of 0)

3 age: age in years

4 sex: sex (1 = male; 0 = female)

5 painloc: chest pain location (1 = substernal; 0 = otherwise)

6 painexer (1 = provoked by exertion; 0 = otherwise)

7 relrest (1 = relieved after rest; 0 = otherwise)

8 pncaden (sum of 5, 6, and 7)

9 cp: chest pain type

-- Value 1: typical angina

-- Value 2: atypical angina

-- Value 3: non-anginal pain

-- Value 4: asymptomatic

10 trestbps: resting blood pressure (in mm Hg on admission to the hospital)

11 htn

12 chol: serum cholestoral in mg/dl

13 smoke: I believe this is 1 = yes; 0 = no (is or is not a smoker)

14 cigs (cigarettes per day)

15 years (number of years as a smoker)

16 fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

17 dm (1 = history of diabetes; 0 = no such history)

18 famhist: family history of coronary artery disease (1 = yes; 0 = no)

19 restecg: resting electrocardiographic results

-- Value 0: normal

-- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)

-- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria

20 ekgmo (month of exercise ECG reading)

21 ekgday(day of exercise ECG reading)

22 ekgyr (year of exercise ECG reading)

23 dig (digitalis used furing exercise ECG: 1 = yes; 0 = no)

24 prop (Beta blocker used during exercise ECG: 1 = yes; 0 = no)

25 nitr (nitrates used during exercise ECG: 1 = yes; 0 = no)

26 pro(calcium channel blocker used during exercise ECG: 1 = yes; 0 = no)

27 diuretic (diuretic used used during exercise ECG: 1 = yes; 0 = no)

28 proto: exercise protocol

1 = Bruce

2 = Kottus

3 = McHenry

4 = fast Balke

5 = Balke

6 = Noughton

7 = bike 150 kpa min/min (Not sure if "kpa min/min" is what was written!)

8 = bike 125 kpa min/min

9 = bike 100 kpa min/min

10 = bike 75 kpa min/min

11 = bike 50 kpa min/min

12 = arm ergometer

29 thaldur: duration of exercise test in minutes

30 thaltime: time when ST measure depression was noted

31 met: mets achieved

32 thalach: maximum heart rate achieved

33 thalrest: resting heart rate

34 tpeakbps: peak exercise blood pressure (first of 2 parts)

35 tpeakbpd: peak exercise blood pressure (second of 2 parts)

36 dummy

37 trestbpd: resting blood pressure

38 exang: exercise induced angina (1 = yes; 0 = no)

39 xhypo: (1 = yes; 0 = no)

40 oldpeak = ST depression induced by exercise relative to rest

41 slope: the slope of the peak exercise ST segment

-- Value 1: upsloping

-- Value 2: flat

-- Value 3: downsloping

42 rldv5: height at rest

43 rldv5e: height at peak exercise

44 ca: number of major vessels (0-3) colored by flourosopy
45 restckm: irrelevant
46 exerckm: irrelevant
47 restef: rest raidonuclid (sp?) ejection fraction
48 restwm: rest wall (sp?) motion abnormality
    0 = none
    1 = mild or moderate
    2 = moderate or severe
    3 = akinesis or dyskmem (sp?)
49 exeref: exercise radinalid (sp?) ejection fraction
50 exerwm: exercise wall (sp?) motion
51 thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
52 thalsev: not used
53 thalpul: not used
54 earlobe: not used
55 cmo: month of cardiac cath (sp?) (perhaps "call")
56 cday: day of cardiac cath (sp?)
57 cyr: year of cardiac cath (sp?)
Attributes 58 through 67 are vessels.
    -- Value 0: < 50% diameter narrowing
    -- Value 1: > 50% diameter narrowing
58 lmt
59ladprox
60 laddist
61 diag
62 cxmain
63 ramus
64 om1
65 om2
66 rcaprox
67 rcadist
68 lvx1: not used
69 lvx2: not used
70 lvx3: not used
71 lvx4: not used
72 lvf: not used

73 cathef: not used

74 junk: not used

75 name: last name of patient (replaced with the dummy string "name")

Features included in the best feature subset found by the feature selection algorithms:

38, 57, 58, 59, 60, 62, 63, 64, 6 6, and 67.