Greedy Attribute Selection

Rich Caruana
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
caruana@cs.cmu.edu

Dayne Freitag
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
dayne@cs.cmu.edu

Abstract

Many real-world domains bless us with a wealth of attributes to use for learning. This blessing is often a curse: most inductive methods generalize worse given too many attributes than if given a *good* subset of those attributes. We examine this problem for two learning tasks taken from a calendar scheduling domain. We show that ID3/C4.5 generalizes poorly on these tasks if allowed to use all available attributes. We examine five greedy hillclimbing procedures that search for attribute sets that generalize well with ID3/C4.5. Experiments suggest hillclimbing in attribute space can yield substantial improvements in generalization performance. We present a caching scheme that makes attribute hillclimbing more practical computationally. We also compare the results of hillclimbing in attribute space with FOCUS and RELIEF on the two tasks.

1 INTRODUCTION

As machine learning is applied to real-world tasks, difficulties arise that do not occur in simpler textbook experiments. One such difficulty is selecting the best attributes to use for learning from a large set of candidate attributes. Ideally, a learning algorithm's generalization performance would improve when it is given the information supplied by additional attributes. Unfortunately, the opposite often occurs: additional attributes can interfere with other more useful attributes.

There are at least two ways excess attributes can hurt generalization. First, noise in the training sample is magnified by adding more noisy attributes. Given many noisy attributes, a learning algorithm is faced with the difficult task of separating signal from noise. The more noisy attributes available, the less likely the learning algorithm will be able to properly identify those attributes that lead to good generalization given a fixed training sample. Second, attributes sometimes interact with the learning algorithm's bias counterproductively. Such attributes might be called *deceptive* for that algorithm: the algorithm uses them because they appear to be useful but, when used, hurt generalization instead of helping it.

Langley and Sage examined the *effect* of irrelevant attributes on the generalization performance of ID3 [10][11]. Littlestone explored *selecting* attributes when many of the attributes are irrelevant [12]. Almuallim and Dietterich's FOCUS [1] and Kira and Rendell's RELIEF [6] are two algorithms that search for relevant attributes. For the tasks we consider here, all the attributes were designed to be relevant, so simply eliminating irrelevant attributes will not help. We examine five greedy hillclimbing algorithms for automatically selecting good performing subsets of attributes to use for decision tree induction [18] with ID3/C4.5 [15][17] when there are many relevant attributes from which to choose. There are many potential advantages of automating the attribute selection process:

- It is often difficult to determine what effects different attribute combinations will have on a learning procedure. Manually selecting attributes is challenging and frequently leads to inferior selection.
- It gives the system designer freedom to identify as many potentially useful attributes as possible and then let the system automatically determine which ones to use.
- It allows new attributes to be easily added to a system on the fly. New attributes might be added by the user, by the system designer, or by new sensing capabilities.
- In domains where the world changes, it allows the currently used attributes to be those best suited to the current state of the world.
- It allows the set of attributes used to change dynamically as the amount of training data changes.

Section 2 describes the attribute selection problem in the context of two learning problems drawn from the Calendar Apprentice (CAP) Project [4]. Section 3 presents five greedy methods that search for good performing sets of attributes by hillclimbing in attribute space. Section 4 describes a caching scheme that makes these methods more practical computationally. Section 5 shows the results of using the attribute hillclimbing methods on two CAP problems. Section 6 compares these results with the results of using improved versions of FOCUS [1] and RELIEF [6].

2 ATTRIBUTE SELECTION IN CAP

2.1 The Calendar Apprentice

The Calendar Apprentice (CAP) [4][9][13] is an apprentice system that learns to schedule meetings by watching real users do meeting scheduling on a day-to-day basis. CAP uses features such as the meeting attendees, the type of meeting, and the current state of the user's calendar to predict the meeting's LOCATION, DAY-OF-WEEK, START-TIME, and DURATION. CAP uses ID3/C4.5 [15][17] run on an accumulating database of previously scheduled meetings to predict these values for new meetings, thereby saving the user from having to determine and enter them.

The CAP domain is rich. Learning in CAP must deal with a very dynamic environment: new attendees appear frequently, scheduling during the summer is very different from scheduling during the fall or at holidays, user habits may change radically with changes in outside factors such as classes or changes in job title, etc. CAP also must deal with a complex hierarchy of meeting types (e.g., advisor-advisee meeting or faculty meeting), temporal regularities (e.g., twice-a-week to once-in-a-lifetime occurrences), and attendee statuses (e.g., graduate student or university president). Because of this, there are many attributes potentially available from which to learn. CAP currently considers 32 attributes (of the thousands potentially available) for predicting DAY-OF-WEEK and START-TIME.

2.2 The Problem with Too Many Attributes

We observe the following when all 32 attributes available in CAP are used for learning:

1. ID3/C4.5 often prefers attribute tests that appear good in isolation but which are suboptimal combined with the other tests that follow it.
2. The information gain splitting criterion favors high arity attributes (i.e., attributes which yield splits with many branches) because they break the training patterns into nearly pure classes.
3. The decision trees use a small subset of the available attributes (usually 10-15 of the available 32).
4. What is learned generalizes poorly to new instances.

The inability to select good attribute combinations (i.e., good *paths* instead of good *nodes*) is, of course, the price one pays for the efficiency of greedy methods like ID3/C4.5.[1] The bias in favor of high arity attributes is a well-known problem with the information gain heuristic we used in some of our experiments.[2] The fact that few attributes are used suggests that either there is redundancy in the attributes, that some attributes are not very useful, or that recursive splitting runs out of data at nodes before it is able to evaluate all potentially useful attributes. The final problem, poor generalization, probably results from problems 1–3.

The greedy attribute hillclimbing procedures we test in this paper partially mitigate these problems by selecting which attributes learning gets to use. Even partial mitigation yields substantial improvements in generalization performance. For example, restricting the attributes learning is allowed to consider more than doubles generalization performance on the DAY-OF-WEEK task.

3 ATTRIBUTE HILLCLIMBING

This section describes five hillclimbing methods that greedily search for attribute subsets that generalize well when given to a learning procedure. The methods differ *only* in the particular hillclimbing strategy they employ.

3.1 Performance Criterion

To do hillclimbing, we first need a metric to define the hills. We use the simplest (conceptually, not computationally!) performance metric: the generalization performance of the learning algorithm given the attribute set. John et al. [8] refer to this as the *wrapper* model for feature selection: the attribute selection procedure is *wrapped* around the induction algorithm that will use the selected attributes. Specifically, we break the training data into a training set and test set, give it to the learning procedure (ID3/C4.5 in our experiments), and measure the generalization performance of what is learned on the held-out test set. The sizes of the training and test sets are specified by the user.

In our experiments, hillclimbing is done using training data collected from recent scheduling events. The result of attribute selection—the set of attributes that performed best on the part of the training set used as the hillclimbing test set—is then used for subsequent learning. Thus in our domain attribute selection finds the attributes that would have performed best historically and then uses this set for future learning. For domains lacking this temporal character, feature selection and the final induction must both be done using a single set of data. Cross-validation is one way to do this: attribute selection is done using subsets of the data and then, once the attributes are chosen, they are used to learn from all the available data [14][8][2].

[1] Methods such as lookahead [3] might eliminate this—and possibly other—problems. Lookahead is expensive, but the attribute selection procedures we test here are also expensive.

[2] Quinlan developed *gain ratio* [17] to remedy this. We have run experiments with both information gain and with gain ratio. We ran some experiments with information gain specifically because we are interested in determining how effective attribute selection is at mitigating poor bias in the learning algorithm.

3.2 Hillclimbing Methods

The first four methods we present are similar to variable selection methods used commonly in statistics [5]. The simplest method is forward selection (FS). It starts with the empty set and greedily adds attributes one at a time until all attributes have been added. At each step FS adds the attribute that, when added to the current set, yields the learned structure that generalizes best. Once an attribute is added FS cannot later remove it. Backward elimination (BE) starts with all attributes in the attribute set and greedily removes them one at a time until no attributes remain. Like forward selection, backward elimination removes at each step the attribute whose removal yields a set of attributes that yields best generalization. Also like FS, once BE removes an attribute, it cannot later add it back to the set.

Irrevocably removing (adding) an attribute is risky—that attribute might later turn out to be important as other attributes are removed (added). A safer procedure is bidirectional hillclimbing: at each step allow an attribute to be added or deleted from the attribute set. Bidirectional hillclimbers can be initialized with an empty attribute set (like FS), with a complete attribute set (like BE), or with a random attribute set. We have experimented with the first two of these. Forward *stepwise* selection (FSS) is greedy attribute hillclimbing initialized with the empty attribute set. Backward *stepwise* elimination (BSE) is attribute hillclimbing initialized with the complete set of attributes. (Be careful: "selection" and "elimination" are misnomers in FSS and BSE, respectively; both methods can add or delete attributes at any step of the search other than the first.)

We also consider one more form of greedy attribute hillclimbing, backward stepwise elimination–SLASH (BSE-SLASH). BSE–SLASH is based on the observation that when there is a large number of attributes, some learning procedures (e.g., ID3/C4.5) frequently do not use many of them. BSE–SLASH starts with the full attribute set (like BSE), but, after taking a step, eliminates (*slashes*) any attributes not used in what was learned at that step. BSE–SLASH does this at every step during its search. This slash heuristic allows BSE–SLASH to quickly jump to regions of the search space where all the attributes in the active subset are playing a role in what is learned.

The three bidirectional hillclimbing methods, FSS, BSE, and BSE–SLASH, can loop if cycle prevention is not employed. The cycle prevention we use allows the bidirectional hillclimbers to exhaustively search the attribute space if not given a computational limit; when stopped they report the best performing attribute subset found so far.

4 CACHING TO SPEED SEARCH

The caching scheme described in this section is applicable whenever a deterministic learning procedure is employed that uses or does not use each attribute given it. Thus the caching scheme is suitable for use with methods such as ID3 [15], C4.5[17], and FOIL [16], but would require significant modification to work with methods such as Winnow [12], backpropagation networks, and nearest neighbor. In the experiments reported here we use ID3/C4.5 to grow decision trees, convert the decision trees to rules, and then prune the rules before testing [17].

4.1 Decision Trees Grown From Large Attribute Sets

When the training sample is limited, decision trees grown from large sets of attributes often do not use all the available attributes. Where this is the case, if decision trees repeatedly will be grown from similar sets of attributes, it is possible to cache results from the trees grown to reduce the cost of growing subsequent trees.

For example, if ID3 is given attributes {A,B,C,D,E} but does not use attributes D and E in the decision tree it generates, then it is not necessary later to grow decision trees given attributes {A,B,C}, {A,B,C,D}, or {A,B,C,E}—ID3 will ignore attributes D and E if either is given with {A,B,C} and will generate identical trees.[3] If we cache decision trees (or just their generalization performance) and record both the attributes given to ID3 and the attributes used in the learned decision tree, rapid lookup can be used before new decision trees are grown to see if the outcome is determinable from prior experience.

4.2 The Caching Algorithm

We employ a discrimination net to store results from what is learned. Each node in the net represents an attribute actually used by the learner in a previously learned hypothesis (e.g., a decision tree). The procedure for insertion into the discrimination net accepts three arguments: a list of the attributes used by the learner, a list of those presented to the learner but not used, and the generalization performance of what is learned. Insertion involves searching for a path in the net corresponding to the attributes used—creating or extending one if no such path exists—and storing both the results and the list of attributes not used at the terminal node. If the terminal node contains results from a previous cache insertion, the attributes not used are merged with those already stored at the node.

It is important that all attributes that *might* affect what is learned be considered "used" for caching. For example, if pruning eliminates an attribute, the attribute that was learned should be considered "used" because it's presence in the pre-pruned hypothesis might affect what is pruned and therefore influence the final hypothesis. We consider all attributes used in the pre-pruned decision tree grown by ID3/C4.5 to be "used," even if some of these attributes are later pruned from the final rule sets.

The retrieval procedure accepts a single list of attributes and traverses branches in the net in search of terminal nodes. It searches for a terminal node in which the attributes not

[3] This is true only if the decision tree procedure is deterministic.

traversed by the path to the node are a subset of the unused attributes stored at the node. In this case it returns the results stored at the node. If it traverses all relevant branches without satisfying the attribute subset criterion, then no entry corresponding to the attribute set is in the cache and learning is invoked.

Because the insertion procedure *merges* unused attributes with those stored as unused in a cache entry, caching can return results for points in the attribute space which have *more* attributes than any point already evaluated. For example, if when given the attribute sets $\{A,B,C,D\}$ and $\{A,B,E,F\}$, the learner only uses attributes A and B, it must be learning the same structure in both cases. We can cache the results for these two sets and confidently predict what will be learned when presented with the attributes A and B plus any subset of $\{C,D,E,F\}$. In the best case, this simple merge operation squares the number of points in attribute space covered by either cache entry in isolation.[4]

In the limit, this caching scheme may require exponential space in the number of attributes. Properties of our domain, however, make this unlikely to pose a problem. First, the limited size of our data sets allows learning to learn structures that use small numbers of attributes, leaving many attributes unused when large attribute sets are evaluated. The caching scheme *covers* regions in the attribute space exponential in size in the number of unused attributes. Second, all our methods are one-pass hill-climbing algorithms; they carve narrow channels through attribute space. This increases the chance that cache entries will be found and reduces the number of cache updates. Finally, hillclimbing is costly enough even with caching that only a small fraction of the space is ever searched in practice.

4.3 The Speedup of Caching

Figure 1 shows the number of trees *grown* when caching as a function of the number of trees *considered* by BSE on the two test problems. Cache misses are common early, when the cache is relatively empty and when significant changes to the attribute sets are being made. As the searches progress, however, the number of cache hits increases dramatically because the caches contain more entries and because many of the points considered are similar to points previously explored. The number of trees grown differs for the two problems; cache hits are less frequent on the START-TIME problem than on the DAY-OF-WEEK problem. Note that on these problems both graphs begin to level off after about 500,000 trees have been *considered*. Since cache hits are several orders of magnitude faster than tree growing, the transition from the rising part of the curves to the level part represents a reduction in the average cost per hillclimbing step from seconds to milliseconds.

Tables 1 and 2 present the number of attribute sets con-

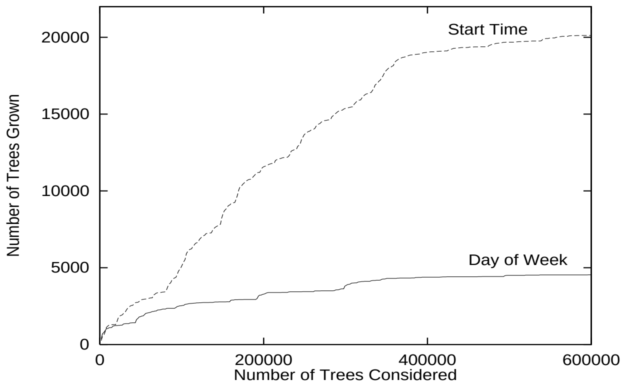

Figure 1: Number of Trees Grown with Caching

sidered and the number of decision trees grown (cache misses) for the two problems. Forward selection and backward elimination are naturally terminating; the entries in the table are the costs for complete runs. The three bidirectional methods, however, can exhaustively explore the search space if not limited. We chose an arbitrary stopping point of 100,000 trees considered. For these problems this represents a point in the searches where most (but not all) of the progress due to hillclimbing has been made and where caching is providing substantial benefit but has not yet reached the point where tree growing has virtually stopped. Caching significantly reduces the number of decision trees that must be grown for most of the methods. If the number of attributes were larger the speedup due to caching would be greater.

Search Method	Trees Considered	Trees Grown	Caching Speedup
FS	528	513	1.03
BE	528	133	3.97
FSS	100,000	3686	27.13
BSE	100,000	2,545	39.29
BSE	600,000	4,541	132.13
BSE–SLASH	100,000	12,435	8.04

Table 1: Speedup Due to Caching for Day of Week

Search Method	Trees Considered	Trees Grown	Caching Speedup
FS	528	528	1.00
BE	528	162	3.26
FSS	100,000	14,536	6.88
BSE	100,000	5,147	19.43
BSE	600,000	20,122	29.82
BSE–SLASH	100,000	15,418	6.49

Table 2: Speedup Due to Caching for Start Time

FS, which starts with no attributes and greedily adds them one-at-a-time, considers small attribute sets early in its search. Because a learner that was not allowed to consider using some attribute can predict nothing about the

[4] If two attribute sets which are equivalent in this way each contain n unused attributes, and the unused attribute sets are disjoint, then instead of covering $2 * 2^n$ points in the attribute space through two cache entries, we cover 2^{2n}.

result of learning with that attribute, caching is not very effective for FS. Because BE considers all attributes early in its search, however, it frequently benefits from caching because the early cache entries have considered large numbers of attributes, many of which are unused in what is learned.

Bidirectional hillclimbing starting from all attributes (BSE) is more similar in behavior to BE than to FS, so it benefits more from caching early in its search than bidirectional hillclimbing starting from no attributes (FSS). BSE–SLASH, which quickly jumps to regions of the search space that FSS would have to climb up to and that BSE would have to climb down to, sees caching benefits somewhere between these two methods early in its search. In later phases of their search, however, all three hillclimbing methods consider a large number of attribute sets similar to sets already tried because they search adjacent points in the space. Because of this, the bidirectional hillclimbing methods benefit more from caching in the long run than either forward selection or backward elimination. *This is important. It allows the bidirectional hillclimbing methods to do a more thorough search of the attribute space using computational cost comparable to that of FS and BE.*

5 EMPIRICAL ANALYSIS

Figure 2 shows the performance on the two problems for forward selection and backward elimination during their search. The horizontal axis is the number of attributes in the set. The vertical axis is the generalization performance of the attribute subset measured on the test set used for hillclimbing. The data was collected using a locally developed version of ID3/C4.5 that differs in some respects from Quinlan's C4.5 code and that uses information gain, not gain ratio. FS starts with the empty set. Its search starts at the left side of each graph and adds attributes one at a time until it terminates with all attributes in the set at the right side of the graphs. BE starts with the complete set of attributes; its search begins at the right side of the graphs and terminates at the left side.

The data suggest that on these problems FS performs better than BE, i.e., it finds sets of attributes that generalize a few percent better.[5] This is not, however, the case in general. Consider a learning problem with three attributes, A, B, and C, that have the information gains and generalization performances shown in Table 3 for ID3 grown trees.

FS first considers the attributes in isolation. Of these, attribute A yields the best generalization, so FS adds attribute A to the attribute set. On subsequent steps FS adds attribute B, and then C. The peak performance FS finds is 3.0 for {A,B}. BE, however, starts with {A,B,C}, considers eliminating each attribute, and finds that {B,C} is best. After then eliminating attribute B, BE terminates and returns {B,C} as the peak performing attribute set. FS never grows a decision tree using {B,C} because the information gain of the attributes in isolation is deceptive: ID3 will always use attribute A at the root of the decision tree given the opportunity. The true utility (to generalization, not to information gain) of the optimal attributes is not apparent when these attributes are considered in isolation and FS first adds attributes that have higher utility in isolation. In other words, FS gets stuck in a local minimum on this problem.

Attribute Set	**Information Gain**	**Generalization Performance**
A	2	2.0
B	1	1.5
C	1	1.0
A,B		3.0
A,C		2.5
B,C		4.0
A,B,C		2.5

Table 3: Deceptive Information Gain

The three bidirectional hillclimbing methods cannot easily be plotted on graphs like those in Figure 2 because they frequently consider many attribute sets of any given size. Moreover, although the graphs in Figure 2 make it easy to compare the progress of FS and BE, they are poor at presenting the computational cost of these algorithms when caching is used. Without caching, the cost of a complete FS run is identical to the cost of a complete BE run. But BE benefits far more from caching than FS because BE caches entries that contain many attributes early in the search.

Figure 3 presents the performance of each of the five algorithms on the two test problems as a function of computational cost. As in Figure 2, this data is collected using information gain, not gain ratio. The generalization performance at each point along the horizontal axis is the performance that would be returned by the search procedure if it were halted at that point and the best attribute set found so far reported. As in Figure 2, this is the performance measured on the test set used for hillclimbing, thus it is an optimistic measure of the generalization performance one would expect from using that attribute set for unseen instances. See Table 4, Section 6.3, for the performance on unseen instances.

The top graphs measure cost by counting the number of decision trees grown (cache misses). Counting the number of decision trees grown, however, can be misleading: it is usually more costly to grow decision trees if there are many attributes than if there are only a few. The bottom graphs in Figure 3 present the same data as the top graphs plotted against a more accurate measure of cost: the number of times an instance's attribute value is queried. The graphs in Figure 3 show the complete searches of the non-bidirectional methods (FS and BE), but only a small fraction (the very early part) of the bidirectional searches. This was done for three reasons:

[5] It might seem that generalization performances of 37% and 19% are so low that something is wrong. It isn't. These are *very* hard tasks. Achieving these levels of performance is not easy and—surprisingly—is actually useful.

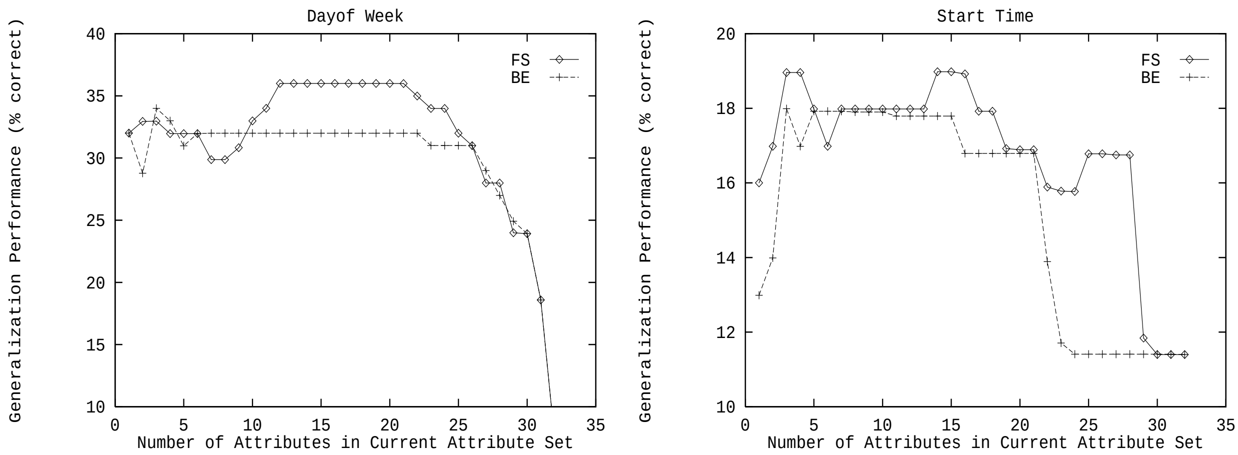

Figure 2: Generalization Performance vs. Number of Attributes for FS and BE

1. Most (but not all) of the progress of the bidirectional methods is made early in the searches.
2. The non-bidirectional methods (FS and BE) are fixed length and consider far fewer decision trees than the bidirectional runs were allowed to consider.
3. This data was collected when all our code was programmed in Lisp. Lisp was slow enough that it was awkward to collect more data than the graphs show.

With caching, and using Quinlan's C4.5 program, the bidirectional hillclimbers take roughly a day to consider 100,000–1,000,000 trees (but only growing about 5,000–20,000 trees) when running on a Sparc ELC given 32 attributes when trained on 100 training instances. Cache lookup time is negligible even after thousands of decision trees have been cached despite the fact that the cache code is written in Lisp and the tree induction code is written in C. The speedup due to caching is important; without it bidirectional hillclimbing would take months to actually grow a million trees. Note that while most of the progress occurs early in the bidirectional searches, some improvements do occur late in the searches.

The graphs suggest five things. First, there is no clear winner. BSE yields the best performance on Day of Week, but BSE-SLASH does best on Start Time. Second, there are clear losers—at least on these problems; the three bidirectional methods outperform both FS and BE in most regions of the graphs. Even if computational cost is an issue, there appears to be little advantage to using the naturally terminating methods FS and BE. Third, the method that is leading changes frequently, and apparently unpredictably, as the searches progress; it would be risky to assume that because one method has an early lead that it will perform best in the long run. The safest bet may be to run all three hillclimbing strategies interleaved. (Note that if more than one method will be run they can share one cache.) Fourth, the bidirectional methods allow a tradeoff to be made between the quality of the result and computational cost: they are easy to stop and restart and, given more cycles, may continue to find better answers. Finally, and perhaps most importantly, all the methods find attribute subsets that yield substantial increases in generalization performance. Clearly some form of attribute selection is useful.

The data in Figures 2 and 3 are collected with a locally developed version of ID3/C4.5 that uses information gain. Inspection of the attribute sets explored by these methods shows that all the selection procedures quickly eliminate the high arity attributes information gain is biased to prefer. It would be interesting to know what advantage, if any, attribute selection would confer to a "smarter" induction algorithm that did not have this bias. To test this, we ran similar experiments on the two problems using the default settings of Quinlan's C4.5 program, most notably gain ratio.

The results of these experiments were surprising. As expected, gain ratio was better than information gain at ignoring high arity attributes. This caused the performance of gain ratio given all the attributes to be better than the performance of information gain given all the attributes. But the performance with all the attributes using gain ratio was not as good as the performance of information gain combined with attribute selection. Thus attribute selection was able to at least partially mitigate the attribute arity problem of information gain and also confer additional benefit. We were surprised, however, to discover that the performance after attribute selection with information again and with gain ratio were virtually identical. For these test problems at least, it appears that all the advantage of using gain ratio could be acquired by using information gain and carefully selecting attributes. This suggests that a tradeoff exists between effort spent crafting the bias of the learning procedure and effort spent in selecting attributes. Some of the poor biases of the learning procedure can, in some domains, be successfully mitigated by careful attribute selection.

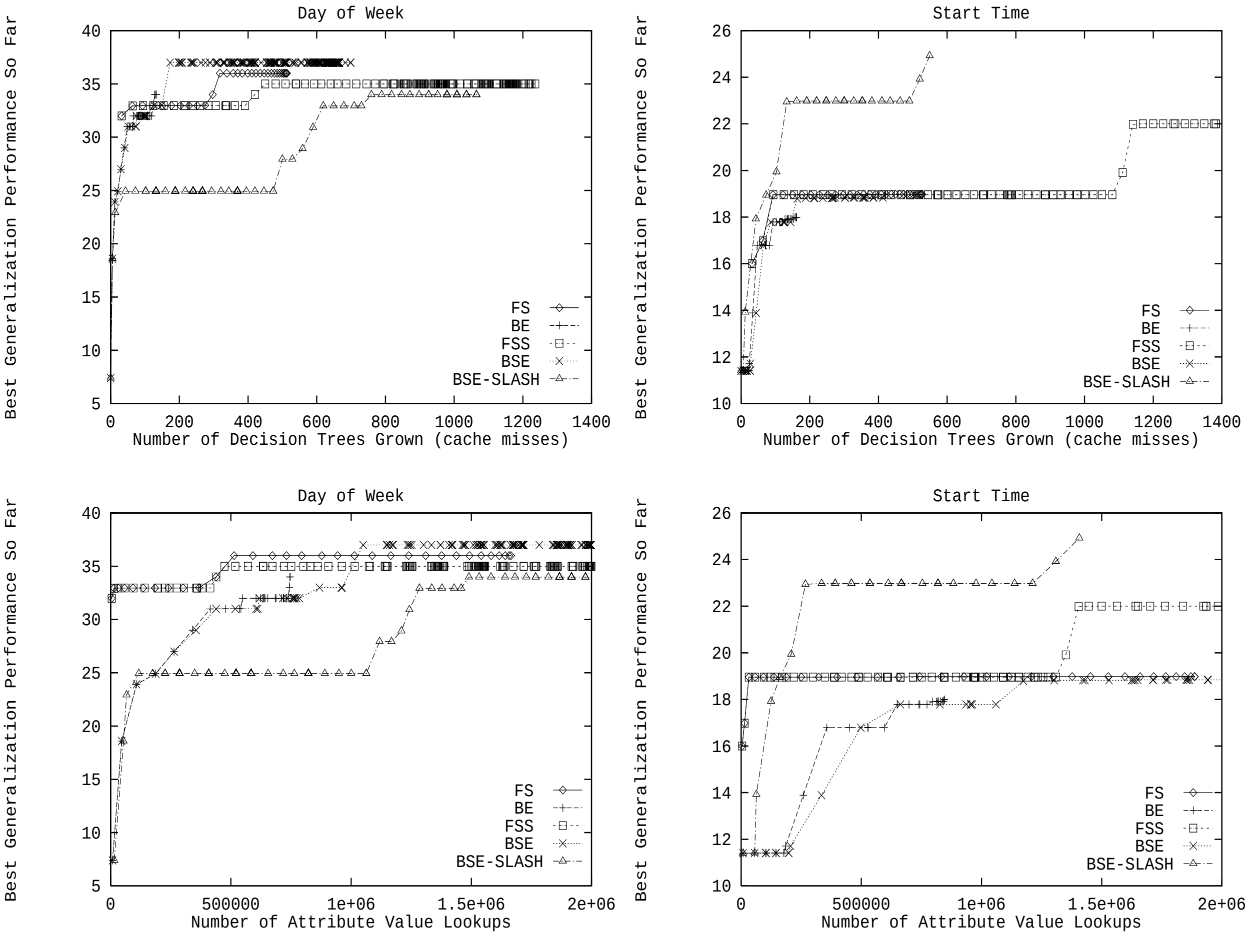

Figure 3: Generalization Performance of Attribute Sets vs. Computational Cost

6 FOCUS and RELIEF

FOCUS [1] and RELIEF [6], two procedures for attribute selection, both search for *relevant* attributes independent of the learning method that will use these attributes. This is quite different from the attribute hillclimbing methods above that embed the learning method within the performance metric of the attribute hillclimber.

6.1 FOCUS

FOCUS [1] searches for the minimum-size attribute subset sufficient to maintain consistency on the training data. There are two difficulties applying FOCUS in CAP. First, the CAP database is not consistent even when all attributes are present. Since a subset of attributes cannot be more consistent than its superset, FOCUS will fail to find a consistent set of attributes if run on a large enough sample from the CAP database. A simple generalization of FOCUS, however, eliminates this problem. If the original sample contains K inconsistencies with all attributes, then FOCUS should return the smallest subset of attributes that also yields exactly K inconsistencies. That is, FOCUS should return the subset of attributes that does not lead to *more* inconsistency in the database than originally present. This preserves the original spirit of FOCUS while significantly broadening its applicability.

The second difficulty is that FOCUS is prohibitively expensive if many attributes are needed to achieve consistency. FOCUS is an exhaustive search procedure that examines all possible subsets of a given size before going on to the next size. The combinatorial explosion resulting from examining the consistency of all subsets of even size 10 of 32 features is beyond current technology. We've enhanced FOCUS to improve its average case performance. We run two different FOCUS-like algorithms at the same time. One searches the smallest subsets first, exactly as in [1]. The other uses depth-first search starting with the largest subset. Running both at the same time has the advantage that if a small subset yields consistency, the first FOCUS will find it efficiently, but if a large subset is necessary, the second FOCUS will find it efficiently. (The worst case occurs when both methods find the same subset at the same time, but this only doubles the total cost.) Using the two also has the advantage that at any point the first FOCUS provides a lower bound on the number of attributes required while the second FOCUS provides an upper bound.

6.2 RELIEF

RELIEF [6] is an attribute relevance determination algorithm inspired by instance-based learning techniques. RELIEF associates with each attribute a weight indicating the relative relevance of that attribute to making class distinctions. RELIEF is attractive for three reasons. First, it is computationally efficient in most domains. Second, it is able to detect feature relevance even when features interact. Third, the relevance weights give a degree of relevance instead of returning only a boolean relevant/not-relevant distinction. This allows attributes to be ranked by relevance.

The main difficulty in applying RELIEF to the two CAP problems is that RELIEF is designed for boolean attributes. Many of the attributes in CAP are not boolean. It is not obvious how best to modify RELIEF to handle higher arity attributes. Kononenko [7] tests two extensions to RELIEF, RELIEF-E and RELIEF-F, to solve this problem and reports that RELIEF-F outperforms RELIEF-E on test problems. We tried both. RELIEF-F made it difficult to determine where to draw the line between what attributes to include and exclude because naturally occurring divisions between the ranked attributes' relevances were eliminated. We tried several dividing lines to find the results reported here for RELIEF-F.

6.3 COMPARISON OF RESULTS

Table 4 compares the generalization performance of the best performing attribute sets found with the five greedy hillclimbing strategies with the performance of FOCUS and RELIEF. FOCUS and RELIEF were allowed to use the entire training set for relevance determination; unlike the hillclimbing methods, they do not require that a subset of the training data be reserved as a hillclimbing test set. The entries in the table are the performances found using information gain when tested on *new,* i.e., previously unseen, instances. (The hillclimbers use the generalization performance on the test set to do hillclimbing. Because of this, the performances in Figures 2 and 3 are optimistic estimates, so tests on a second test set are necessary.) The data suggests that the greedy methods find attribute sets that perform better than FOCUS and RELIEF on these test problems. Additional experiments indicate that the greedy methods continue to outperform the relevance methods even when gain ratio is used.

Search Method	**Day of Week**	**Start Time**
ALL ATTRIBUTES	4	8
FS	23	14
BE	32	14
FSS	27	7
BSE	34	11
BSE–SLASH	14	11
FOCUS	4	7
RELIEF-E	10	7
RELIEF-F	7	3

Table 4: Percent Accuracy on Unseen Instances

The poor performance of FOCUS and RELIEF could be due to the inappropriateness to these tasks of their bias for small attribute sets.[6] Table 5 compares the number of attributes returned by the different methods. All the methods return comparably sized attribute sets, except for forward selection, which seems to need more. Thus the problem FOCUS and RELIEF have in this domain is not due to the inappropriateness of the bias for few features—there are small attribute sets that generalize well on these tasks.

Search Method	**Day of Week**	**Start Time**
ALL ATTRIBUTES	32	32
FS	12	14
BE	4	4
FSS	5	5
BSE	7	6
BSE–SLASH	5	6
FOCUS	9	6
RELIEF-E	8	7
RELIEF-F	6	9

Table 5: Number of Attributes Used by Each Method

The problem, then, must be the emphasis of FOCUS and RELIEF on *consistency* as a selection criterion. FOCUS searches for the smallest attribute set that yields consistency on the training set. RELIEF, which searches for the relevant attributes, defines relevance in terms of consistency. RELIEF searches for the subset of attributes that are most strongly positively correlated with consistency and most strongly negatively correlated with inconsistency in neighborhoods of the training set. This emphasis on consistency causes both FOCUS and RELIEF to select some attributes that, while relevant, are not useful for generalization, and to ignore other attributes that while less relevant than those selected, would be more useful to learning. In other words, the emphasis on consistency gives these algorithms a strong bias towards attributes and attribute combinations that shatter the space well but which do not necessarily generalize well. FOCUS, in particular, is prone to overfitting because it is driven to achieve complete consistency on the training set and will keep adding attributes to repair a single inconsistency even though statistically this would be unwarranted. (It would be interesting modify FOCUS to maintain the best attribute set for *each* degree of inconsistency. The generalization performance of these could then be plotted as a function of inconsistency. A similar graph could be

[6] The only size bias of the hillclimbing methods is to prefer the smallest attribute set when attribute sets of different size have equal performance.

collected for RELIEF by using successively larger subsets of attributes ordered by relevance.)

7 CONCLUSION

An ideal learning algorithm would use only attributes that lead to good generalization so there would be no need for attribute selection methods like FOCUS, RELIEF, and the hillclimbing strategies tested here. Unfortunately, many learning procedures have their own biases and weaknesses and, given excess attributes will often hang themselves.

Attribute selection procedures are one way of mitigating the inability of a learning procedure to properly deal with excess attributes. Selection procedures such as FOCUS and RELIEF attempt to do this *independent* of the learning algorithm and goal criterion. The greedy attribute evaluation strategies presented here take the learning algorithm into account by incorporating it into the attribute evaluation process and allow whatever performance metric is important to be used as the selection criterion. This flexibility, combined with the efficiency boost obtainable with caching, makes them a viable means of performing attribute selection for learning methods like ID3/C4.5.

Our experiments suggest attribute hillclimbing is effective at finding good performing subsets of attributes. The results also indicate that bidirectional hillclimbing in attribute space (the three *stepwise* methods) is more effective than either forward or backward selection. Surprisingly, the results suggest that attribute selection can sometimes mitigate poor bias in the learning algorithm effectively enough that it yields performance comparable to what can be achieved with "smarter" learning algorithms that presumably have better bias. The results also indicate that selection procedures such as FOCUS and RELIEF that use relevance as a criterion instead of the learning procedure and learning criterion can lead to inferior attribute selection.

Acknowledgements

Thanks to David Zabowski for help in developing some of these ideas and for assistance in running the experiments, to Tom Mitchell and Andrew Moore for letting us bounce ideas off them, and to Ron Kohavi and the anonymous reviewers for many useful comments and suggestions.

Research sponsored by the Wright Laboratory, Aeronautical Systems Center and the Advanced Research Projects Agency under grant F33615-93-1-1330, and by a National Science Foundation Graduate Research Fellowship.

References

[1] H. Almuallim and T.G. Dietterich, "Learning With Many Irrelevant Features," *AAAI-91 proceedings, 9th National Conference on Artificial Intelligence,* 1991.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees,*, Wadsworth International Group, 1984.

[3] W. Buntine, and R. Caruana, "Introduction to IND and Recursive Partitioning," *NASA Ames TR:FIA-91-28*, 1991

[4] L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski, "A Personal Learning Apprentice," *Proceedings of 1992 National Conference on Artificial Intelligence*, 1992.

[5] N. Draper and H. Smith, *Applied Regression Analysis,*, John Wiley & Sons, 2nd edition, 1981.

[6] K. Kira and L. Rendell, "The Feature Selection Problem: Traditional Methods and a New Algorithm," *AAAI-92 proceedings, 10th International Conference on Artificial Intelligence,*, 1992.

[7] I. Kononenko, "Estimating Attributes: Analysis and Extensions of Relief," to appear in *Proceedings of the European Conference on Machine Learning,* 1994.

[8] G. John, R. Khavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," to appear in *The Proceedings of the Eleventh International Conference on Machine Learning,*, Rutgers, NJ., July 1994.

[9] J. Jourdan, L. Dent, J. McDermott, T. Mitchell, and D. Zabowski, "Interfaces that Learn: A Learning Apprentice for Calendar Management," CMU Technical Report CMU-CS-91-135, Carnegie Mellon University, 1991.

[10] P. Langley, S. Sage, "Scaling to Domains with Many Irrelevant Features," presented at the workshop on Computational Learning and Natural Learning, 1993.

[11] P. Langley, S. Sage, "Oblivious Decision Trees and Abstract Cases," to be presented at the AAAI94 Workshop on Case-Based Reasoning, Seattle, WA., 1994.

[12] N. Littlestone, "Learning quickly when irrelevant attributes abound," *Machine Learning,* **2**:4, pp. 285–318

[13] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski, "Experience With a Personal Learning Assistant," to appear in Communications of the ACM, 1994.

[14] A. Moore and M. Lee, "Efficient Algorithms for Minimizing Cross Validation Error,", to appear in *The Eleventh International Conference on Machine Learning,* Rutgers, NJ., July, 1994.

[15] J.R. Quinlan, "Induction of Decision Trees," *Machine Learning,* **1,** pp. 81-106, 1986.

[16] J.R. Quinlan, "Determining Literals in Inductive Logic Programming," *Proceedings of the Eighth International Workshop on Machine Learning,* 1991.

[17] J.R. Quinlan, *C4.5: Programs for Machine Learning,* Morgan Kaufman, 1993.

[18] S. Salzberg, "Improving Classification Methods via Feature Selection," John Hopkins TR, 1992.