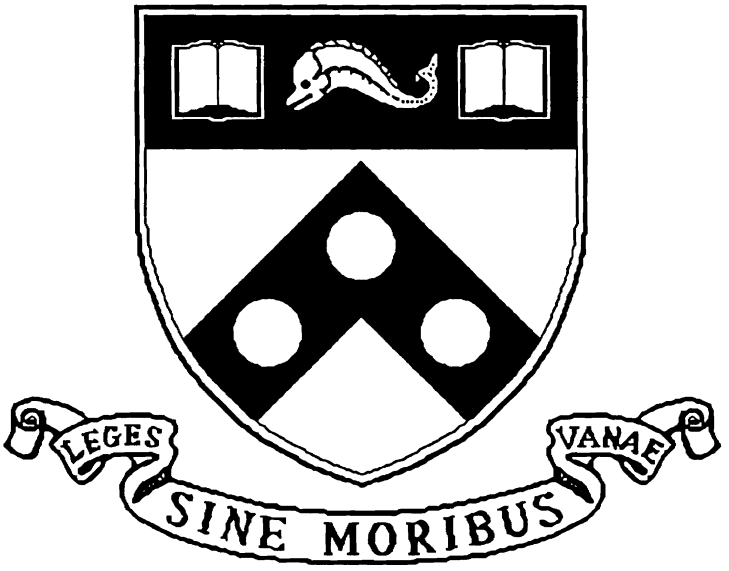
# An SE-tree based Characterization of the Induction Problem

MS-CIS-93-42
LINC LAB 248

Ron Rymon




University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

April 1993

# An SE-tree based Characterization of the Induction Problem

**Ron Rymon**
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
rymon@linc.cis.upenn.edu

(Proceedings Machine Learning Conference, Amherst MA, 1993)

## Abstract

Many induction programs use decision trees both as the basis for their search, and as a representation of their classifier solution. In this paper we propose a new structure, called SE-tree, as a more general alternative.

## 1 INTRODUCTION

Many learning algorithms use decision trees as an underlying framework for search and as a representation of their classifier solutions (e.g. ID3 [Quinlan, 86], CART [Breiman *et al.*, 84]). This framework, however, is known to mix *search bias* (introduced when the algorithm decides on the order in which attributes are to be used in splitting) with *hypotheses-space bias*. To avoid being trapped by this bias, several researchers have suggested averaging over multiple trees (e.g. [Buntine, 91]). In this paper, still within a recursive partitioning framework, we propose using an alternative data structure called SE-tree [Rymon, 92]. On one hand, since the new framework shares many of the features of decision tree-based algorithms, we should be able to adopt many sub-techniques developed for the latter. On the other hand, an SE-tree embeds a large number of decision trees, thereby providing a more expressive, more flexible, representation for classifiers. Importantly, SE-tree-based algorithms can eliminate almost completely the search bias, admitting instead a *user-specified* hypotheses-space preference criterion.

Section 2 outlines a formal theory of induction where classifiers take the form of collections of rules. Sections 3 and 4 present the SE-tree, and render it useful in searching and representing such collections (the *learning* phase), and in subsequently using them for *classification*. Incorporation of user-specified bias in either stage, or in both, is described in Sections 4 and 5. Section 6 presents *general* results relating the SE-tree to decision trees, with some algorithmic implications.

## 2 A THEORY FOR INDUCTION

Formalizing the induction problem, we will examine collections of production rules that best model the function (concept) represented by the training data. Rules provide a common denominator for decision trees on one hand, and SE-trees on the other, since there is an obvious one-to-one mapping between rules and leaves of such trees.

Let us introduce a few useful definitions first: Let $\text{ATTRS} \stackrel{\text{def}}{=} \{A_i\}_{i=1}^{n}$ be a set of *attributes* (also called features or variables), where each attribute $A_i$ can take values from a finite unordered discrete domain denoted $Dom(A_i)$. A *partial description* is a subset of ATTRS, each instantiated from its own domain. An *object* is a complete partial instantiation, i.e. one in which all attributes are instantiated. By UNIVERSE we refer to the collection of all objects. Consider, for example, a space of 3 binary attributes (A,B,C), hereafter called 3BIN. In 3BIN, {A=0,C=1} is a partial description. {A=0,B=0,C=1} is an object. UNIVERSE is 3BIN itself; it is made of a total of 8 objects. A training set (TSET), consisting of objects labeled by their correct *class* ($\pi$), makes an *induction problem instance.*

**Example 2.1** *(The Checkers Problem)*

Consider a universe defined by two 3-valued attributes (A,B), and a set of four classes $(\alpha, \beta, \gamma, \delta)$. The following figure depicts a training data, and an illustration of UNIVERSE.

| A | B | $\pi$ |
|---|---|---|
| 1 | 2 | $\alpha$ |
| 2 | 1 | $\beta$ |
| 2 | 3 | $\gamma$ |
| 3 | 2 | $\delta$ |

| A \ B | 1 | 2 | 3 |
|---|---|---|---|
| 1 | | $\alpha$ | |
| 2 | $\beta$ | | $\gamma$ |
| 3 | | $\delta$ | |

□

Having defined a problem instance, we shall try to characterize a solution. Conceptually, we assume the existence of a function (*target*) from UNIVERSE to the set of classes, and that the training data agree with this function. Our goal is to approximate *target*

over the complete universe, using conjunctive rules as our elementary building blocks.

A *rule*, R, is simply a partial description such that all objects in TSET which agree with it are equally classified, i.e. for every t,t'∈TSET, if R⊆t,t' then $\pi(t) = \pi(t')$. To avoid irrelevant rules, we add the additional requirement that an object matched by a rule is provided in TSET. As a partial description, a rule defines an equivalence class within the universe, namely $[R] \stackrel{\text{def}}{=} \{t \in \text{UNIVERSE} \mid R \subseteq t\}$. Moreover, since all objects in TSET∩[R] agree on their class, we can define $\pi([R])$ to be that class, and write a production rule of the form $R \Rightarrow \pi([R])$. Thus, from here on, we shall interchangeably talk about a rule as a set of instantiated attributes, as a region in UNIVERSE, and as a conjunction of antecedents. To model a target function, we use *collections* of rules, interpreted disjunctively for each class. In general, there may possibly be many such collections. The Checkers problem, for instance, admits 8 rules and thus $2^8$ collections. The purpose of an inductive theory is to characterize *desirable* features of candidate collections. Bias, or preference, expresses the relative desirability of one collection versus another.

Our theory has a single bias: for the most part, we will prefer rules that are syntactically simpler. By *kernel rules* we refer to rules that are most-general (minimal set-wise). Other bias, necessary to distinguish equally simple hypotheses, is deliberately left out of the theory. Our algorithms will *modularly* implement a *user-specified* preference criterion. Consider the Checkers problem again. Only four of the eight rules are also kernel rules: (1) (A=1)⇒ $\alpha$, (2) (B=1)⇒ $\beta$, (3) (B=3)⇒ $\gamma$, and (4) (A=3)⇒ $\delta$. All other rules, e.g. (A=1)∧(B=2)⇒ $\alpha$, are subsumed by one or more of the kernel rules.

Let C be a collection of rules for a problem instance P, we use Kernel(C) to denote the collection of kernel rules for P that subsume rules in C. The collection of all kernel rules, denoted KRULES, is the target of our induction algorithms. Doing so, we avoid *overfitting* of the training data. We propose that *over-generalization* be dealt with in the classification phase via resolution methods based on the user's preference criterion. Intuitively, while learning, we adopt most-general principles. Rules that are *too* general will be in conflict with others, and will then be resolved.

**Definition 2.2** *Completeness*

A collection of rules C is said to be *complete* w.r.t. T⊆UNIVERSE if for every t∈T, there exists a rule R∈C such that R⊆t.

**Proposition 2.3**

1. Let C be a collection of rules that is complete w.r.t. some T⊆UNIVERSE, then Kernel(C) is also complete w.r.t. T;
2. KRULES is complete w.r.t. TSET, but is not necessarily complete w.r.t. UNIVERSE.

Thus, in the Checkers problem, {A=2,B=2} is not covered by any rule (including non-kernel!). In contrast, *any* decision tree *is* complete w.r.t. UNIVERSE. But is completeness desired at all? One may argue that incompleteness of KRULES is often a direct result of important incompleteness of the training data. SE-tree-based classification algorithms can, however, extend their coverage by relaxing the rule matching procedure.

**Definition 2.4** *Consistency*

A collection of rules C is said to be *consistent* w.r.t. T⊆UNIVERSE, if for every t∈T, and rules R,R'∈C, if R,R'⊆t, then $\pi([R]) = \pi([R'])$.

**Proposition 2.5**

1. Every collection of rules is consistent w.r.t. TSET (by definition), but KRULES may be inconsistent w.r.t. UNIVERSE;
2. Every collection of rules contains a consistent sub-collection.

Thus, in the Checkers problem, each of the "corner" objects is covered by two contradicting kernel rules (e.g. {A=1,B=1} is covered by (A=1)⇒ $\alpha$ and (B=1)⇒ $\beta$). As per Proposition 2.5(2), KRULES may have (possibly several) sub-collections, the latter may have lesser *coverage* than KRULES. In contrast, *any* decision tree *is* consistent w.r.t. UNIVERSE. But is consistency desirable at all? KRULES is inconsistent when two rules are over-general to the point in which they contradict one another on as yet unseen parts of UNIVERSE. While ideally, one or both rules need be specialized or removed, the training data alone does *not* provide us with any suitable preference criterion. An *external* preference criteria, or *bias* [Mitchell, 80], must be applied.

Bias can be defined as the set of all factors that collectively influence hypothesis selection [Utgoff, 86]. [Buntine, 90] divides such criteria into three separate classes: *hypothesis space bias* are those criteria which specify a preference for one classifier over another; *search bias* consists of criteria used to guide the actual search for such; and finally, bias may have an *application specific* component. Adopting Buntine's dichotomy, we believe that an ideal learning system must *eliminate* search bias. Put differently, bias should be stated by the *user*, *independently* from the particular algorithm used.

We believe SE-trees represent a step in that direction. So far, we have introduced a single bias – a prefer-

ence for kernel rules. Next, when presenting the *SE-Learn* family of learning algorithms, we defer the introduction of bias to the latest possible. A variety of user-defined preference criteria can be plugged into the learning and/or classification algorithms.

# 3 A LEARNING ALGORITHM

## 3.1 SET ENUMERATION TREES

Many problems in Computer Science were formalized to admit solutions in the form of *sets*, or in the form of partial instantiations of a set of variables. Typically, such sets are required to satisfy some problem-specific criterion which designates them as solutions. In addition, where multiple solutions may exist, they are often ranked by their plausibility, likelihood, or desirability. Regularly, such preference involves a minimality (or maximality) criterion, e.g. minimal entropy, maximum probability or utility, etc. *Set Enumeration (SE) trees* [Rymon, 92] were shown to be useful as the basis for a unifying search-based framework for such domains. SE-trees support complete, irredundant, and prioritized search; their special structure allows for efficient pruning and other optimizations.

Let $\text{ATTRS} \stackrel{\text{def}}{=} \{A_i\}_{i=1}^{n}$ be a set of attributes with domains $\text{Dom}(A_i)$ respectively, and let *ind*:ATTRS$\rightarrow\mathbb{N}$ be an indexing of the set of attributes. We define the *SE-tree View* of a partial description $S$ as follows:

$$\text{View(S)} \stackrel{\text{def}}{=} \{\text{A}\in\text{ATTRS} \mid \text{ind(A)}>\text{Max}_{A' \text{ in } S}\text{ind(A')}\}$$

**Definition 3.1** *Extended Set Enumeration Tree*

The *extended SE-tree* for a set of attributes ATTRS is defined as follows:

1. At its root is a node labeled with the empty set;
2. Recursively, let S be a node's label, It has children labeled as follows:

$$\{\ \text{S}\cup\{\text{A=v}\} \mid \text{A}\in\text{View(S)},\ \text{v}\in\text{Dom(A)}\}.$$

**Example 3.2** Figure 1 depicts an extended SE-tree for the *complete* 3BIN space. Note that restricting a node's expansion to its *View*, ensures that *every* member of 3BIN is *uniquely* explored within the tree. □

Representing all elements of a power-set, the complete SE-tree is clearly exponential in size. However, in a large class of problems, especially where solutions are monotonic with respect to set inclusion, the SE-tree can be used to induce a complete and yet often efficient search because it allows for systematic pruning [Rymon, 92].

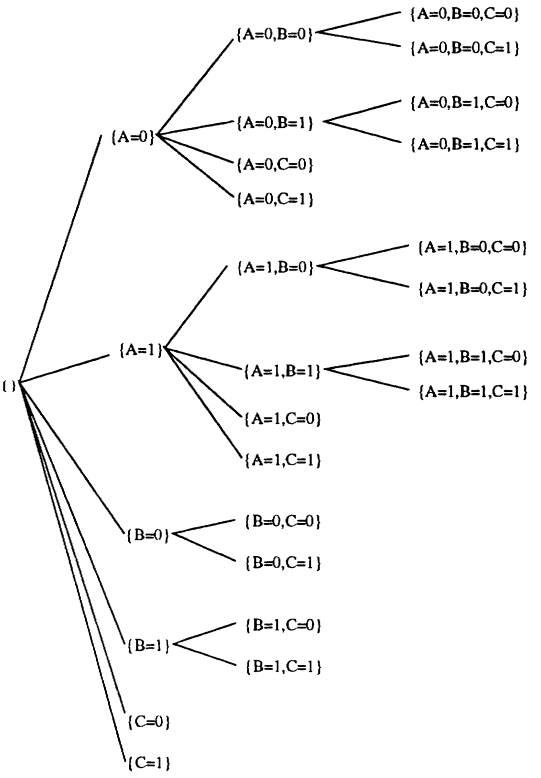



Figure 1: Complete SE-tree for 3 Binary Variables

## 3.2 SE-TREE-BASED LEARNING

Aimed at all kernel rules, SE-Learn (Algorithm 3.4) explores top-down an imaginary SE-tree. Nodes are explored by some predetermined priority function. In Sections 4 and 5, we show this prioritization useful in implementing various biases. In expanding open nodes, SE-Learn exploits the SE-tree structure to prune away nodes that cannot *lead* to kernel rules. SE-Learn's output is an SE-tree which leaves are labeled with kernel rules.

**Definition 3.3** *Candidate and Impotent Expansions*

Let S be a node, $\text{TSET(S)} \stackrel{\text{def}}{=} \{t \in \text{TSET} \mid S \subseteq t\}$. We say that $\text{S'} \stackrel{\text{def}}{=} \text{S}\cup\{\text{A=v}\}$ is a *candidate expansion* of S if $\text{A}\in\text{View(S)}$, $\text{v}\in\text{Dom(A)}$. However, S' is *impotent* if either

1. TSET(S') is empty; or
2. TSET(S')=TSET(S); or
3. all objects in TSET(S') agree on their assignment to attributes in $View(S')$, but there is not a complete agreement on the class (i.e. S' is not a rule).

**Algorithm 3.4**

*Program SE-Learn*

1. OPEN-NODES $\leftarrow \{\phi\}$;
2. Until OPEN-NODES is empty do
3. Expand (Extract-Min(OPEN-NODES))

*Procedure Expand(S)*

1. For every candidate expansion $R \stackrel{def}{=} S \cup \{A=v\}$ that is not impotent and that is not subsumed by a previously discovered rule do
2. If R is a rule then mark it as such; otherwise add it to OPEN-NODES.

The algorithm works by exploring nodes along the SE-tree's current fringe (OPEN-NODES) in a best-first fashion. For that purpose, nodes are cached in a priority queue and accessed via an *Extract-Min* operation. Candidate expansions that are not subsumed by previously discovered rules (step 1) are marked as rules if they satisfy the definition or otherwise marked for expansion and added to the queue for further consideration (step 2).

### 3.3 EXPLORATION POLICIES

An exploration policy is simply the priority function used in Algorithm 3.4 to determine the order in which nodes are explored. It is easy to verify that if nodes are explored by their cardinality (breadth-first exploration of the tree) then the algorithm is correct, i.e. it computes all and only kernel rules. As so far described, any monotonic function $\psi$, i.e. such that $S \subseteq S'$ implies $\psi(S) \leq \psi(S')$), results in Algorithm 3.4 being correct. A large class of interesting functions are monotonic, e.g. ones that are based on probability, utility, or information-gain measures. However, at some computational expense, SE-Learn can be modified to admit non-monotonic exploration policies as well. The sole purpose of the monotonicity restriction is to avoid recording non-minimal solutions; therefore, to remove it, we need to also check whether new rules subsume old ones.

Note however that, as so far presented, *all* exploration policies will result in the *same* tree structure. The variety of exploration policies allowed will become important next, in specifying and implementing preference criteria.

## 4 CLASSIFICATION ALGORITHMS

Given an SE-tree acquired as above, we want to be able to use it to classify *new* objects. As in decision tree-based classification algorithms, this is done by following matching paths from the root to class-labeled leaves (rules).

Recall however that in the SE-tree representation

1. there may be *no* such leaf (rule) (we called this incompleteness); or
2. there may be *multiple* rules (and thus leaves) matching a given object, and they may not always be equally labeled (we called this inconsistency).

The SE-tree incompleteness, we argued, is due to the incompleteness of the training data. One way to "complete" the SE-tree is to perform *partial* matching in cases where there are no perfectly matching rules.

The inconsistency property, on the other hand, gives the SE-tree its main power. Roughly, inconsistency reflects a variety of perspectives that could be adopted to logically partition the training data. In a decision tree, a *single* such perspective is decided upon at the learning phase in the choice of attribute for each branching point. Representing multiple perspectives is more expressive and allows more principled resolution. In particular, *hypotheses-space* preference, explicitly specified by the user, can be used to resolve conflicts.

Algorithm 4.1 uses such preferences directly: by searching the SE-tree best-first with respect to the specified preference, it picks the leaf which maximizes the specified preference from all those matching the object at hand.

**Algorithm 4.1** *Classification via SE-tree Search*

- **Input:** (1) an object; (2) an SE-tree; and (3) an exploration policy $\psi$ (bias).
- **Procedure:** Search SE-tree best-first (according to $\psi$), along paths matching the object. Stop when the first leaf is hit, or when the tree is exhausted.
- **Output:** If a leaf was hit, predict its class label. Otherwise, either respond "don't know", or guess, or re-search the tree allowing for partial matching.

A more general approach involves specifying a *resolution criterion*, e.g. weighted averaging or voting, which takes into account *all* rules matching a given object. The two approaches can, of course, be combined by applying the resolution criterion to a *subset* of the rules – those which rank highest by the preference criterion.

The following experiment, using the Monks benchmark [Thrun *et al.*, 91], demonstrates the importance of the particular choice of resolution criterion. In general, a preference and/or a resolution criterion should reflect some domain knowledge. However, given the artificial nature of the Monks problems, we experimented

with three generic weight functions: simple voting; quadratic (in the rule's size) weight voting, favoring more specific rules; and inverse quadratic, favoring more general rules. In the learning phase, we simply learned all kernel rules. In classification, when the rules were incomplete, we used partial matching. Conflicts were resolved using each of the three weight functions. Figure 2 compares accuracy obtained using each of the resolution criteria to each other; to the average reported for other decision tree-based programs and to the overall average reported for all methods. Note that SE-Learn's performance is crucially dependent on the resolution criterion used.

| | Monk1 | Monk2 | Monk3 |
|---|---|---|---|
| SE-Learn (inv. quad.) | 85.9% | 71.3% | 95.6% |
| SE-Learn (voting) | 72.0% | 69.0% | 88.4% |
| SE-Learn (quadratic) | 64.8% | 67.1% | 70.8% |
| Average decision trees | 84.2% | 67.6% | 86.9% |
| Average overall | 88.9% | 76.4% | 90.9% |

Figure 2: Various Resolution Criteria

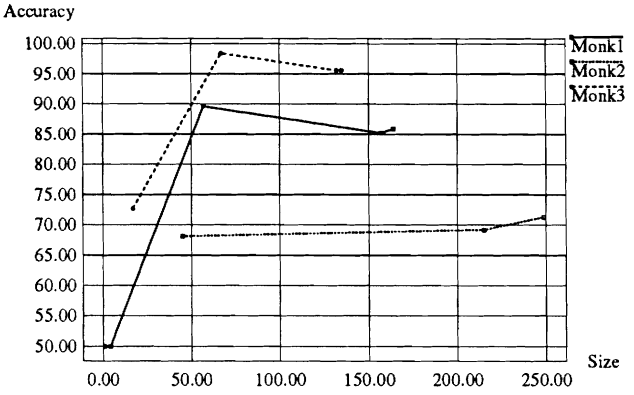



Figure 3: Complexity vs. Accuracy

# 5 BIAS IN THE LEARNING PHASE

## 5.1 PARTIALLY EXPLORED SE-TREES

It may often be intractable, or practically impossible, to explore all kernel rules. Exploration policies can then be used as early as the learning phase to prune away less promising parts of the SE-tree. Even when all kernel rules *can* be explored, added complexity may not pay in the margin. Worse, as with many other learning frameworks, more complex SE-trees can even have lower accuracy than their simpler subsets. In such instances, it is standard to use hill-climbing procedures and/or anytime algorithms which explore as time/space permit and return the best classifier seen so far. In SE-Learn, the SE-tree can be constructed gradually while testing to make sure that the added complexity of new rules is worth the marginal improvement in accuracy. When interrupted, or when it runs out of resources (particularly space) this procedure will return the best classifier it has seen so far. The particular exploration policy used plays an important role in this procedure since it determines the order in which rule nodes are seen. Using again the Monks problems, we ran an experiment in which an SE-tree was explored level by level. The change in complexity (measured by the number of rules) and in accuracy (using the inverse quadratic resolution criterion) is depicted in Figure 3.

## 5.2 SPECIAL COLLECTIONS OF RULES

In what follows, we briefly describe variations of SE-Learn that compute SE-trees corresponding to collections of rules with special features. Here too, the particular collection computed is determined by the exploration policy.

### Consistent Sub-Collections of KRULES

A collection of kernel rules is inconsistent w.r.t. UNIVERSE iff it has rules $R_1$, $R_2$ such that $\pi([R_1]) \neq \pi([R_2])$ and no attribute appears in both $R_1$–$R_2$, and $R_2$–$R_1$. Thus, SE-Learn could be modified not to retain rules which are inconsistent with previously discovered rules. Since the order in which nodes are explored determines which rules are retained, the particular exploration policy used defines a bias.

### Minimal Sub-Collections of KRULES

For TSET-completeness purposes, a rule R is redundant if every object in TSET that R matches is also matched by another rule R'. As before, one can modify SE-Learn so as not to retain rules deemed redundant by previously discovered rules. Another alternative is to restrict redundancy to $n$ rules per training instance, or to rules that satisfy some other acceptance criterion such as statistical significance. Again, the particular exploration policy defines a bias.

### Consistent and Complete Collections of Rules

The down side of discarding inconsistent rules, as suggested above, is that the collection of rules obtained may be incomplete even w.r.t. TSET. To avoid this, rather than discarding such rules, SE-Learn can be modified to further *expand* them. The collection of rules so obtained are guaranteed to be complete. However, individual rules may no longer be kernel.

### Minimal and Consistent Collections

By removing both inconsistent and redundant rules, one may get a minimal collection of rules that is both complete and consistent.

# 6 SE-TREE AND DECISION TREES

A number of decision tree based algorithms have had an impact on machine learning research. Part of our purpose here is to convince researchers to look at the SE-tree as a more general alternative to decision trees. We devote this section to a broader comparison of the two data structures.

## 6.1 A FOREST OF DECISION TREES

One way to view a decision tree is as an SE-tree in which every possible object has exactly one path along which it can be classified, i.e. an SE-tree that is consistent and complete w.r.t. UNIVERSE[1]. Conversely, one way to view an SE-tree is as a collection, or *forest*, of decision trees. A single SE-tree can be shown to embed a large number of decision trees. In particular, let $D$ be a decision tree in which attributes were chosen monotonically w.r.t. some indexing function. Let $S$ be an SE-tree constructed in accordance to same indexing function, then $S$ *embeds* $D$, i.e. there exists a subset of $S$'s edges which forms a tree that is topologically and semantically equivalent to $D$, and that is rooted at $S$'s root. One particular decision tree is the SE-tree's *primary decision tree*: the one in which each internal node is expanded with the first attribute in that node's SE-tree *View* that does not result in impotent expansions.

This result can be strengthened to make the SE-tree embed *any* single decision tree[2]. In particular, let $D$ be a decision tree that is constructed by any ID3-like procedure. To create an SE-tree that embeds $D$ we may have to slightly alter the definition of an SE-tree to allow for *dynamic re-indexing*. In particular, we will develop an indexing as we create the tree:

1. At first, we will choose an initial indexing $ind_{root}$ in which the first attribute used in $D$ appears first;
2. Then, while at a node labeled $S$, let $ind_{parent(S)}$ be the indexing used in expanding $S$'s parent. In $S$, we use an indexing which coincides with $ind_{parent(S)}$ on all attributes *not* in $View(S)$, but may re-order attributes in $View(S)$ as we wish. In particular, if a node corresponding to $S$ appears in $D$, we will re-order attributes in $View(S)$ so that the first attribute used in $D$ to split that node appears first.

By construction, $D$ will be embedded in an SE-tree created as above as its primary decision tree. It is fairly easy to verify that the SE-tree remains complete and irredundant, and that SE-Learn remains correct.

[1] An SE-tree, however, can be consistent and complete without being a decision tree.

[2] *Not* all of them at once; rather a collection that includes a specific decision tree.

## 6.2 IMPROVING UPON A GIVEN DECISION TREE

An important corollary of the result above is that one can *construct* an exploration policy under which SE-Learn will start off with one's favorite decision tree, and then try to improve it by adding more rule nodes. (This exploration policy may be non-monotonic though.) Of course, rule nodes will only be added to the extent in which accuracy (as tested empirically on a separate training set) is improved.

We have tested this approach on the Monks benchmark. In each of the three problems, we started with a decision tree constructed by the information-gain criterion. Then, the rest of the SE-tree was explored breadth-first. Accuracy and complexity were recorded for the primary decision tree, and for each level of the tree in which rules were added (Figure 4).

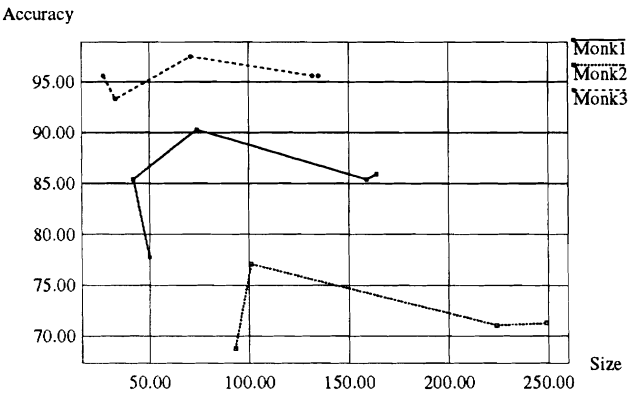



Figure 4: Starting from a Decision Tree

Note that in all three problems, the accuracy of the primary decision tree could be improved by adding SE-tree nodes, although this improvement is not monotonic. Also note that in Monk1, adding the SE-tree's first level has not only improved the accuracy, but has also *reduced* the number of rule nodes (some decision tree rules were pruned because they were subsumed by newly discovered rules).

## 6.3 HYPOTHESES EXPRESSIBILITY

Consider the following problem instance:

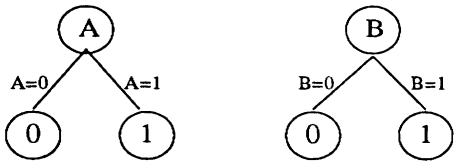
| A | B | Class |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |

| | | B | |
|---|---|---|---|
| | | 0 | 1 |
| A | 0 | 0 | ? |
| | 1 | ? | 1 |

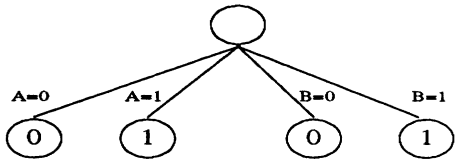
While *four* different hypotheses are consistent with this training data, there are only *two* ID3-style[3] de-

[3] There are more decision trees, but only these can be generated by an ID3-like procedure.

cision trees (Figure 5). The corresponding SE-tree contains (as subset of its arcs) both trees, and can be used to represent *all four* hypotheses depending on the particular exploration policy (bias) used in a given classification session.



(a) Decision Trees



(b) SE-Tree

Figure 5: SE-tree versus Decision Trees

Consider, for example, an OR function (not modeled by either decision trees). In SE-Learn, if a search-based approach to classification is adopted (Algorithm 4.1), an OR function can be implemented using an exploration policy that assigns high priority to the arcs A=1 and B=1. Generalizing this problem to $n$ attributes, each taking its values from $\{0 \ldots n-1\}$, we are given a training set with the $n$ cases in which all attributes, and the class, are equally labeled. Now, we consider a function that takes the most frequent value among its attributes, with bias towards higher values in case of equality (for $n = 2$, we get the OR function). Such function cannot be modeled by any of the ID3-style decision trees[4], but can easily be modeled using an SE-tree with a resolution criterion based on simple voting.

### 6.4 COMPUTING KERNEL RULES

Considering the goal of computing all kernel rules, three problems may arise in a decision tree-based search framework:

1. The *minimality* problem – rules will often not be discovered in their minimal (kernel) form;
2. The *multiplicity* problem – each kernel rule may be discovered multiply, disguised in a number of its non-minimal supersets; and
3. The *incompleteness* problem – some kernel rules may not be discovered at all.

Both the minimality problem and the multiplicity phenomenon result from the fact that attributes used high in the tree are necessary for some, but not all, the rules. The minimality problem is often addressed by subsequently pruning the rules extracted from the decision tree (e.g. [Quinlan, 87]). The replication problem, a special case of multiplicity in which whole subtrees are replicated, has been addressed by several researchers (e.g. [Rivest, 87, Pagallo & Haussler, 90]). Incompleteness, which is only a problem if one is really interested in all kernel rules, results from the insisted mutual exclusivity of any decision tree's leaves (see [Weiss & Indurkhya, 91]). None of these problems occurs in the SE-tree-based framework:

1. Rules are always discovered in their kernel form;
2. Kernel rules are always discovered uniquely; and
3. All kernel rules are discovered.

### 6.5 COMPLEXITY

The SE-tree's exhaustiveness and large *initial* branching may be deceiving. Let us first compare its worst-case complexity to that of a decision tree, *independently* of their use.

**Proposition 6.1** If all attributes are $b$-valued, then the number of nodes in a complete decision tree is $b^n + b^{n-1} + \cdots b + 1 > b^n$. The size of a complete SE-tree is $(b+1)^n$. In sharp contrast, the size of a super-tree containing all decision trees is significantly larger: $b^n \cdot n!$.

Within an induction framework, however, one rarely explores a complete decision tree (nor a complete SE-tree for that matter). In an ID3-like framework, the size of a decision tree is *linear* in the size of the training data. This is *not* true of SE-Learn! Kernel rules are close relatives of prime-implicants, and as such we know of pathological examples in which the number of kernel rules is exponential in the size of the training data. On the other hand, as just explained, one does not have to explore the entire SE-tree and one can always have the first nodes explored be those of one's favorite decision tree.

## 7 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

We have proposed an inductive learning framework which uses an SE-tree as a basis for search and classifier representation and have presented a family of algorithms for SE-tree induction and for SE-tree-based classification. We have shown that as a representation for classifiers, SE-trees generalize decision trees in two ways: first, a decision tree is a special case of an SE-tree, and second, an SE-tree contains many decision trees. An SE-tree can also be built by improving upon one's favorite decision tree. However, unlike decision trees, most of the *search bias* can be eliminated

[4] In fact, a decision tree modeling this function is necessarily exponential.

in SE-tree-based algorithms; an independently specified *hypothesis-space bias* can be used instead.

Importantly, the SE-tree-based framework can borrow from techniques developed for decision trees. In particular

1. More expressive representation languages can be adopted, e.g. ordered and hierarchical variables, multi-variable tests, class probability trees, etc. Discretization techniques, and criteria developed for selecting a splitting test can be used to handle ordered variables; averaging and smoothing techniques can be used in conjunction with class probabilities representation.
2. Pruning techniques developed for decision trees, e.g. using statistical significance tests, can also be used in SE-trees.
3. Entropy-minimization and other criteria developed for selecting the next splitting attribute in decision trees will likely be useful in selecting an indexing function for an SE-tree which will minimize the number of nodes that have to be explored.

More research, however, is needed to figure ways in which these techniques can be deployed *effectively*.

Other areas of future research include general and domain-specific exploration policies and resolution criteria, termination criteria suitable for various tradeoffs between accuracy and time/space, and an incremental version of SE-Learn.

Recent advances in search algorithms lend themselves to improved implementation of the SE-tree-based framework, e.g. linear-space best-first search algorithms [Korf, 92, Russell, 92] and a SIMD version of IDA* [Powley *et al.*, 93].

### Acknowledgements

The idea of using the SE-trees to learn rules originated at a talk by Tom Mitchell — I thank him for that, as well as for later suggestions. I also thank Kevin Atteson, Russ Greiner, Haym Hirsh, Alon Luss, Teow-Hin Ngair, Michael Niv, Greg Provan, Philip Resnik, Nick Short, Scott Weinstein, and anonymous reviewers for commenting on previous drafts. This work was supported in part by a graduate fellowship ARO grant DAAL03-89-C0031PRI.

## References

[Breiman *et al.*, 84] Breiman, L., Friedman, J., Olshen, R., and Stone, C., *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[Buntine, 90] Buntine, W., Myths and Legends in Learning Classification Rules. *Proc. AAAI-90*, Boston, MA, pp. 736-742, 1990.

[Buntine, 91] Buntine, W., Learning Classification Trees. Technical Report, NASA Ames Research Center, 1991.

[Korf, 92] Korf, R. E., Linear-Space Best-First Search: Summary of Results. *Proc. AAAI-92*, San Jose CA, 1992.

[Mitchell, 80] Mitchell, T. M., The Need for Biases in Learning Generalizations. *Technical Report 5-110*, Rutgers University, 1980.

[Pagallo & Haussler, 90] Pagallo, G., and Haussler, D., Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5, pp. 71-99, 1990.

[Powley *et al.*, 93] Powley, C., Ferguson, C., and Korf, R. E., Depth-First Heuristic Search on a SIMD Machine. *Artificial Intelligence*, 60, 1993, pp. 199-242.

[Quinlan, 86] Quinlan, J. R., Induction of Decision Trees. *Machine Learning*, 1(1):81-106, 1986.

[Quinlan, 87] Quinlan, J. R., Generating Production Rules from Decision Trees. *Proc. IJCAI-87*, pp. 304-307, 1987.

[Rivest, 87] Rivest, R., Learning Decision Lists. *Machine Learning*, 2, pp. 229-246, 1987.

[Russell, 92] Russell, S., Efficient Memory-Bounded Search Algorithms. *Proc. ECAI-92*, Vienna, Austria, 1992.

[Rymon, 92] Rymon, R., Search through Systematic Set Enumeration. *Proc. KR-92*, Cambridge MA, 1992.

[Thrun *et al.*, 91] Thrun, S. B., Bala, J., Bloedron, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S. E., Fisher, D., Hammann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuzinger, J., Michalski, R. S., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., Van de Welde, W., Wenzel, W., Wnek, J., Zhang, J., The MONK's Problems – A Performance Comparison of Different Learning Algorithms. *Technical Report CMU-CS-91-197*, December 1991.

[Utgoff, 86] Utgoff, P. E., *Machine Learning of Inductive Bias.* Kluwer Academic, Boston MA, 1986.

[Weiss & Indurkhya, 91] Weiss, S. M., and Indurkhya, N., Reduced Complexity Rule Induction. *Proc. IJCAI-91*, pp. 678-684, Sydney, Australia, 1991.