Journal of Machine Learning Research 8 (2007) 891-933 Submitted 1/06; Revised 12/06; Published 5/07

Anytime Learning of Decision Trees

Saher Esmeir ESAHER@CS.TECHNION.AC.IL
Shaul Markovitch SHAULM@CS.TECHNION.AC.IL
Department of Computer Science
Technion—Israel Institute of Technology
Haifa 32000, Israel

Editor: Claude Sammut

Abstract

The majority of existing algorithms for learning decision trees are greedy—a tree is induced top-down, making locally optimal decisions at each node. In most cases, however, the constructed tree is not globally optimal. Even the few non-greedy learners cannot learn good trees when the concept is difficult. Furthermore, they require a fixed amount of time and are not able to generate a better tree if additional time is available. We introduce a framework for anytime induction of decision trees that overcomes these problems by trading computation speed for better tree quality. Our proposed family of algorithms employs a novel strategy for evaluating candidate splits. A biased sampling of the space of consistent trees rooted at an attribute is used to estimate the size of the minimal tree under that attribute, and an attribute with the smallest expected tree is selected. We present two types of anytime induction algorithms: a *contract* algorithm that determines the sample size on the basis of a pre-given allocation of time, and an *interruptible* algorithm that starts with a greedy tree and continuously improves subtrees by additional sampling. Experimental results indicate that, for several hard concepts, our proposed approach exhibits good anytime behavior and yields significantly better decision trees when more time is available.

Keywords: anytime algorithms, decision tree induction, lookahead, hard concepts, resource-bounded reasoning

1. Introduction

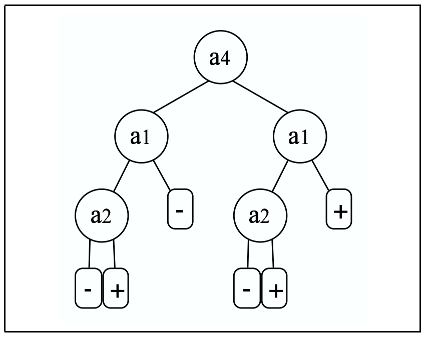
Assume that a medical center has decided to use medical records of previous patients in order to build an automatic diagnostic system for a particular disease. The center applies the C4.5 algorithm on thousands of records, and after few seconds receives a decision tree. During the coming months, or even years, the same induced decision tree will be used to predict whether patients have or do not have the disease. Obviously, the medical center is willing to wait much longer to obtain a better tree—either more accurate or more comprehensible.

Consider also a planning agent that has to learn a decision tree from a given set of examples, while the time at which the model will be needed by the agent is not known in advance. In this case, the agent would like the learning procedure to learn the best tree it can until it is interrupted and queried for a solution.

In both of the above scenarios, the learning algorithm is expected to exploit additional time allocation to produce a better tree. In the first case, the additional time is allocated in advance. In the second, it is not. Similar resource-bounded reasoning situations may occur in many real-life

a_1	a_2	a_3	a_4	label
1	0	0	1	+
0	1	0	0	+
0	0	0	0	-
1	1	0	0	-
0	1	1	1	+
0	0	1	1	-
1	0	1	1	+

(a) A set of training instances.



(b) ID3's performance.

Figure 1: Learning the 2-XOR concept $a_1 \oplus a_2$, where a_3 and a_4 are irrelevant

applications such as game playing, planning, stock trading and e-mail filtering. In this work, we introduce a framework for exploiting extra time, preallocated or not, in order to learn better models.

Despite the recent progress in advanced induction algorithms such as SVM (Vapnik, 1995), *decision trees* are still considered attractive for many real-life applications, mostly due to their interpretability (Hastie et al., 2001, chap. 9). Craven (1996) lists several reasons why the understandability of a model by humans is an important criterion for evaluating it. These reasons include, among others, the possibility for human validation of the model and generation of human-readable explanations for the classifier predictions. When classification cost is important, decision trees may be attractive in that they ask only for the values of the features along a single path from the root to a leaf. In terms of accuracy, decision trees have been shown to be competitive with other classifiers for several learning tasks.

The majority of existing methods for decision tree induction build a tree top-down and use local measures in an attempt to produce small trees, which, by Occam's Razor (Blumer et al., 1987), should have better predictive power. The well-known C4.5 algorithm (Quinlan, 1993) uses the gain ratio as a heuristic for predicting which attribute will yield a smaller tree. Several other alternative local greedy measures have been developed, among which are ID3's information gain, Gini index (Breiman et al., 1984), and chi-square (Mingers, 1989). Mingers (1989) reports an empirical comparison of several measures, and concludes that the predictive accuracy of the induced trees is not sensitive to the choice of split measure and even random splits do not significantly decrease accuracy. Buntine and Niblett (1992) present additional results on further domains and conclude that while random splitting leads to inferior trees, the information gain and Gini index measures are statistically indistinguishable.

The top-down methodology has the advantage of evaluating a potential attribute for a split in the context of the attributes associated with the nodes above it. The local greedy measures, however, consider each of the remaining attributes independently, ignoring potential interaction between different attributes (Mingers, 1989; Kononenko et al., 1997; Kim and Loh, 2001). We refer to the family of learning tasks where the utility of a set of attributes cannot be recognized by examining only subsets of it as tasks with a *strong interdependency*. When learning a problem with a strong interdependency, greedy measures can lead to a choice of non-optimal splits. To illustrate the above, let us consider the 2-XOR problem $a_1 \oplus a_2$ with two additional irrelevant attributes, a_3 and a_4. Assume that the set of examples is as listed in Figure 1(a). We observe that gain-1 of the irrelevant attribute a_4 is the highest:

$$0.13 = \text{gain}_1(a_4) > \text{gain}_1(a_1) = \text{gain}_1(a_2) = \text{gain}_1(a_3) = 0.02,$$

and hence ID3 would choose attribute a4 first. Figure 1(b) gives the decision tree as produced by ID3. Any positive instance with value 0 for a_4 would be misclassified by this decision tree. In the general case of parity concepts, the information gain measure is unable to distinguish between the relevant and irrelevant attributes because neither has a positive gain. Consequently, the learner will grow an overcomplicated tree with splits on irrelevant variables that come either in addition to or instead of the desired splits.

The problem of finding the smallest consistent tree[1] is known to be NP-complete (Hyafil and Rivest, 1976; Murphy and McCraw, 1991). In many applications that deal with hard problems, we are ready to allocate many more resources than required by simple greedy algorithms, but still cannot afford algorithms of exponential complexity. One commonly proposed approach for hard problems is anytime algorithms (Boddy and Dean, 1994), which can trade computation speed for quality. Quinlan (1993, chap. 11) recognized the need for this type of anytime algorithm for decision tree learning: “What is wanted is a resource constrained algorithm that will do the best it can within a specified computational budget and can pick up threads and continue if this budget is increased. This would make a challenging thesis topic!”

There are two main classes of anytime algorithms, namely *contract* and *interruptible* (Russell and Zilberstein, 1996). A contract algorithm is one that gets its resource allocation as a parameter. If interrupted at any point before the allocation is completed, it might not yield any useful results. An interruptible algorithm is one whose resource allocation is not given in advance and thus must be prepared to be interrupted at any moment. While the assumption of preallocated resources holds for many induction tasks, in many other real-life applications it is not possible to allocate the resources in advance. Therefore, in our work, we are interested both in contract and interruptible decision tree learners.

In this research, we suggest exploiting additional time resources by performing lookahead. Lookahead search is a well-known technique for improving greedy algorithms (Sarkar et al., 1994). When applied to decision tree induction, lookahead attempts to predict the profitability of a split at a node by estimating its effect on deeper descendants of the node. One of the main disadvantages of the greedy top-down strategy is that the effect of a wrong split decision is propagated down to all the nodes below it (Hastie et al., 2001, chap. 9). Lookahead search attempts to predict and avoid such non-contributive splits during the process of induction, before the final decision at each node is made.

Lookahead techniques have been applied to decision tree induction by several researchers. The reported results vary from *lookahead produces better trees* (Norton, 1989; Ragavan and Rendell, 1993; Dong and Kothari, 2001) to *lookahead does not help and can hurt* (Murthy and Salzberg, 1995). One problem with these works is their use of a uniform, fixed low-depth lookahead, therefore disqualifying the proposed algorithms from serving as anytime algorithms. Another problem is the data sets on which the lookahead methods were evaluated. For simple learning tasks, such as induction of conjunctive concepts, greedy methods perform quite well and no lookahead is needed. However, for more difficult concepts such as XOR, the greedy approach is likely to fail. Few other

1. A consistent decision tree is a tree that correctly classifies all training examples.

```
Procedure TDIDT($E$,$A$)
    If $E = \emptyset$ Return Leaf($nil$)
    If $\exists c$ such that $\forall e \in E$ Class($e$) $= c$
        Return Leaf($c$)
    $a \leftarrow$ CHOOSE-ATTRIBUTE($A$,$E$)
    $V \leftarrow$ domain($a$)
    Foreach $v_i \in V$
        $E_i \leftarrow \{e \in E \mid a(e) = v_i\}$
        $S_i \leftarrow$ TDIDT($E_i$,$A - \{a\}$)
    Return Node($a, \{\langle v_i, S_i\rangle \mid i = 1 \ldots |V|\}$)
```

Figure 2: Procedure for top-down induction of decision trees. E stands for the set of examples and A stands for the set of attributes.

non-greedy decision tree learners have been recently introduced (Bennett, 1994; Utgoff et al., 1997; Papagelis and Kalles, 2001; Page and Ray, 2003). These works, however, are not capable to handle high-dimensional difficult concepts and are not designed to offer anytime behavior.[2] The main challenge we face in this work is to make use of extra resources to induce better trees for hard concepts. Note that in contrast to incremental induction (Utgoff, 1989), we restrict ourselves in this paper to batch setup where all the training instances are available beforehand.

2. Contract Anytime Induction of Decision Trees

TDIDT (top-down induction of decision trees) methods start from the entire set of training examples, partition it into subsets by testing the value of an attribute, and then recursively call the induction algorithm for each subset. Figure 2 formalizes the basic algorithm for TDIDT. We focus first on consistent trees for which the stopping criterion for the top-down recursion is when all the examples have the same class label. Later, we consider pruning, which allows simpler trees at the cost of possible inconsistency with the training data (Breiman et al., 1984; Quinlan, 1993).

In this work we propose investing more time resources for making better split decisions. We first discuss tree size as a desired property of the tree to be learned and then we describe an anytime algorithm that uses sampling methods to obtain smaller trees.

2.1 Inductive Bias in Decision Tree Induction

The hypothesis space of TDIDT is huge and a major question is what strategy should be followed to direct the search. In other words, we need to decide what our *preference bias* (Mitchell, 1997, chap. 3) will be. This preference bias will be expressed in the CHOOSE-ATTRIBUTE procedure that determines which tree is explored next.

Ultimately, we would like to follow a policy that maximizes the accuracy of the tree on unseen examples. However, since these examples are not available, a heuristic should be used. Motivated by Occam's Razor, a widely adopted approach is to prefer smaller trees. The utility of this principle

2. In Section 5 we discuss related works in details.

to machine learning algorithms has been the subject of a heated debate. Several studies attempted to justify Occam's razor with theoretical and empirical arguments (Blumer et al., 1987; Quinlan and Rivest, 1989; Fayyad and Irani, 1990). But a number of recent works have questioned the utility of Occam's razor, and provided theoretical and experimental evidence against it.

Quinlan and Cameron-Jones (1995) provided empirical evidence that oversearching might result in less accurate rules. Experimental results with several UCI data sets indicate that the complexity of the produced theories does not correlate well with their accuracy, a finding that is inconsistent with Occam's Razor. Schaffer (1994) proved that no learning bias can outperform another bias over the space of all possible learning tasks. This looks like theoretical evidence against Occam's razor. Rao et al. (1995), however, argued against the applicability of this result to real-world problems by questioning the validity of its basic assumption about the uniform distribution of possible learning tasks. Webb (1996) presented C4.5X, an extension to C4.5 that uses similarity considerations to further specialize consistent leaves. Webb reported an empirical evaluation which shows that C4.5X has a slight advantage in a few domains and argued that these results discredit Occam's thesis.

Murphy and Pazzani (1994) reported a set of experiments in which all the possible consistent decision trees were produced and showed that, for several tasks, the smallest consistent decision tree had higher predictive error than slightly larger trees. However, when the authors compared the likelihood of better generalization for smaller vs. more complex trees, they concluded that simpler hypotheses should be preferred when no further knowledge of the target concept is available. The small number of training examples relative to the size of the tree that perfectly describes the concept might explain why, in these cases, the smallest tree did not generalize best. Another reason could be that only small spaces of decision trees were explored. To verify these explanations, we use a similar experimental setup where the data sets have larger training sets and attribute vectors of higher dimensionality. Because the number of all possible consistent trees is huge, we use a *Random Tree Generator* (RTG) to sample the space of trees obtainable by TDIDT algorithms. RTG builds a tree top-down and chooses the splitting attribute at random.

We report the results for three data sets: *XOR-5*, *Tic-tac-toe*, and *Zoo* (See Appendix A for detailed descriptions of these data sets). For each data set, the examples were partitioned into a training set (90%) and a testing set (10%), and RTG was used to generate a sample of 10 million consistent decision trees. The behavior in the three data sets is similar: the accuracy monotonically decreases with the increase in the size of the trees (number of leaves), confirming the utility of Occam's Razor. Further experiments with a variety of data sets indicate that the inverse correlation between size and accuracy is statistically significant (Esmeir and Markovitch, 2007).

It is important to note that smaller trees have several advantages aside from their probable greater accuracy, such as greater statistical evidence at the leaves, better comprehensibility, lower storage costs, and faster classification (in terms of total attribute evaluations).

Motivated by the above discussion, our goal is to find the smallest consistent tree. In TDIDT, the learner has to choose a split at each node, given a set of examples E that reach the node and a set of unused attributes A. For each attribute a, let $T_{min}(a,A,E)$ be the smallest tree rooted at a that is consistent with E and uses attributes from $A-\{a\}$ for the internal splits. Given two candidate attributes a_1 and a_2, we obviously prefer the attribute whose associated $T_{min}(a)$ is the smaller. For a set of attributes A, we define $\widetilde{A}$ to be the set of attributes whose associated $T_{min}(a)$ is the smaller. Formally, $\widetilde{A} = \arg\min_{a \in A} T_{min}(a)$. We say that a splitting attribute a is *optimal* if $a \in \widetilde{A}$. Observe that if the learner makes an optimal decision at each node, then the final tree is necessarily globally optimal.

```
Procedure ENTROPY-K$(E, A, a, k)$
    If $k = 0$
        Return $\mathrm{I}(\mathrm{P}_E(c_1), \ldots, \mathrm{P}_E(c_n))$
    $V \leftarrow \mathrm{domain}(a)$
    Foreach $v_i \in V$
        $E_i \leftarrow \{e \in E \mid a(e) = v_i\}$
        Foreach $a' \in A$
            $A' \leftarrow A - \{a'\}$
            $h_i(a') \leftarrow$ ENTROPY-K$(E_i, A', a', k-1))$
    Return $\sum_{i=1}^{|V|} \frac{|E_i|}{|E|} \min_{a' \in A}(h_i(a'))$

Procedure GAIN-K$(E, A, a, k)$
    Return $\mathrm{I}(\mathrm{P}_E(c_1), \ldots, \mathrm{P}_E(c_n)) -$
                ENTROPY-K$(E, A, a, k)$
```

Figure 3: Procedures for computing entropy$_k$ and gain$k$ for attribute a

2.2 Fixed-depth Lookahead

One possible approach for improving greedy TDIDT algorithms is to lookahead in order to examine the effect of a split deeper in the tree. ID3 uses entropy to test the effect of using an attribute one level below the current node. This can be extended to allow measuring the entropy at any depth k below the current node. This approach was the basis for the IDX algorithm (Norton, 1989). The recursive definition minimizes the $k-1$ entropy for each child and computes their weighted average. Figure 3 describes an algorithm for computing entropy$_k$ and its associated gain$_k$. Note that the gain computed by ID3 is equivalent to gain$_k$ for $k = 1$. We refer to this lookahead-based variation of ID3 as *ID3-k*. At each tree node, ID3-k chooses the attribute that maximizes gain$_k$.[3]

ID3-k can be viewed as a contract anytime algorithm parameterized by k. However, despite its ability to exploit additional resources when available, the anytime behavior of ID3-k is problematic. The run time of ID3-k grows exponentially as k increases.[4] As a result, the gap between the points of time at which the resulting tree can improve grows wider, limiting the algorithm's flexibility. Furthermore, it is quite possible that looking ahead to depth k will not be sufficient to find an optimal split. Entropy$_k$ measures the weighted average of the entropy in depth k but does not give a direct estimation of the resulting tree size. Thus, when $k < |A|$, this heuristic is more informed than entropy$_1$ but can still produce misleading results. In most cases we do not know in advance what value of k would be sufficient for correctly learning the target concept. Invoking exhaustive lookahead, that is, lookahead to depth $k = |A|$, will obviously lead to optimal splits, but its computational costs are prohibitively high. In the following subsection, we propose an alternative approach for evaluating attributes that overcomes the above-mentioned drawbacks of ID3-k.

3. If two attributes yield the same decrease in entropy, we prefer the one whose associated lookahead tree is shallower.
4. For example, in the binary case, ID3-k explores $\prod_{i=0}^{k-1}(n-i)^{2^i}$ combinations of attributes.

```
Procedure SID3-CHOOSE-ATTRIBUTE$(E, A)$
    Foreach $a \in A$
        $p(a) \leftarrow \text{gain}_1(E, a)$
    If $\exists a$ such that $\text{entropy}_1(E, a) = 0$
        $a^* \leftarrow$ Choose attribute at random from
                $\{a \in A \mid \text{entropy}_1(E, a) = 0\}$
    Else
        $a^* \leftarrow$ Choose attribute at random from $A$;
                for each attribute $a$, the probability
                of selecting it is proportional to $p(a)$
    Return $a^*$
```

Figure 4: Attribute selection in SID3

2.3 Estimating Tree Size by Sampling

Motivated by the advantages of smaller decision trees, we introduce a novel algorithm that, given an attribute a, evaluates it by estimating the size of the minimal tree under it. The estimation is based on Monte-Carlo sampling of the space of consistent trees rooted at a. We estimate the minimum by the size of the smallest tree in the sample. The number of trees in the sample depends on the available resources, where the quality of the estimation is expected to improve with the increased sample size.

One way to sample the space of trees is to repeatedly produce random consistent trees using the RTG procedure. Since the space of consistent decision trees is large, such a sample might be a poor representative and the resulting estimation inaccurate. We propose an alternative sampling approach that produces trees of smaller expected size. Such a sample is likely to lead to a better estimation of the minimum.

Our approach is based on a new tree generation algorithm that we designed, called *Stochastic ID3* (SID3). Using this algorithm repeatedly allows the space of "good" trees to be sampled semi-randomly. In SID3, rather than choose an attribute that maximizes the information gain, we choose the splitting attribute randomly. The likelihood that an attribute will be chosen is proportional to its information gain.[5] However, if there are attributes that decrease the entropy to zero, then one of them is picked randomly. The attribute selection procedure of SID3 is listed in Figure 4.

To show that SID3 is a better sampler than RTG, we repeated our sampling experiment (Section 2.1) using SID3 as the sample generator. Figure 5 compares the frequency curves of RTG and SID3. Relative to RTG, the graph for SID3 is shifted to the left, indicating that the SID3 trees are clearly smaller. Next, we compared the average minimum found for samples of different sizes. Figure 6 shows the results. For the three data sets, the minimal size found by SID3 is strictly smaller than the value found by RTG. Given the same budget of time, RTG produced, on average, samples that are twice as large as that of SID3. However, even when the results are normalized (dashed line), SID3 is still superior.

Having decided about the sampler, we are ready to describe our proposed contract algorithm, *Lookahead-by-Stochastic-ID3* (LSID3). In LSID3, each candidate split is evaluated by the estimated

5. We make sure that attributes with gain of zero will have a positive probability of being selected.

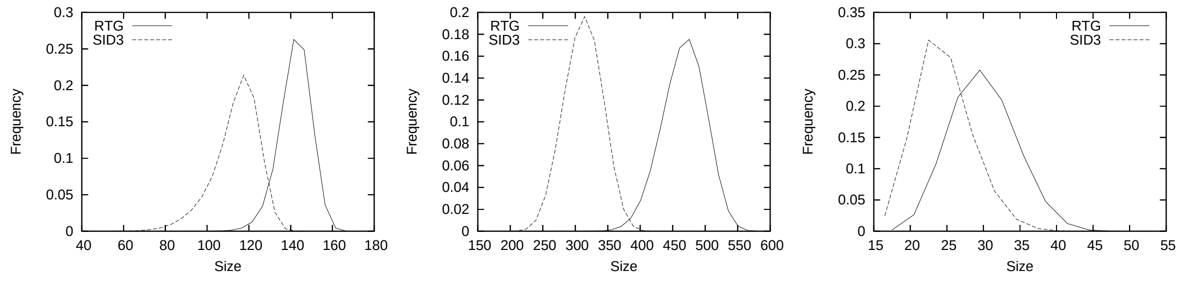


Figure 5: Frequency curves for the *XOR-5* (left), *Tic-tac-toe*, and *Zoo* (right) data sets

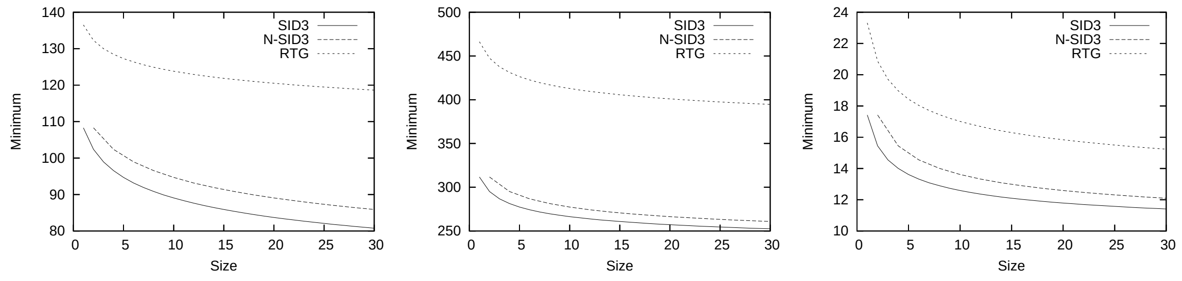


Figure 6: The minimum as estimated by SID3 and RTG as a function of the sample size. The data sets are *XOR5* (left), *Tic-tac-toe* and *Zoo* (right). The dashed line describes the results for SID3 normalized by time.

size of the subtree under it. To estimate the size under an attribute a, LSID3 partitions the set of examples according to the values a can take and repeatedly invokes SID3 to sample the space of trees consistent with each subset. Summing up the minimal tree size for each subset gives an estimation of the minimal total tree size under a.

LSID3 is a contract algorithm parameterized by r, the sample size. LSID3 with $r = 0$ is defined to choose the splitting attribute using the standard ID3 selection method. Figure 7 illustrates the choice of splitting attributes as made by LSID3. In the given example, SID3 is called twice for each subset and the evaluation of the examined attribute a is the sum of the two minima: $min(4,3) + min(2,6) = 5$. The method for choosing a splitting attribute is formalized in Figure 8.

To analyze the time complexity of LSID3, let m be the total number of examples and n be the total number of attributes. For a given node y, we denote by n_y the number of candidate attributes at y, and by m_y the number of examples that reach y. ID3, at each node y, calculates gain for n_y attributes using m_y examples, that is, the complexity of choosing an attribute is $O(n_y \cdot m_y)$. At level i of the tree, the total number of examples is bounded by m and the number of attributes to consider is $n - i$. Thus, it takes $O(m \cdot (n - i))$ to find the splits for all nodes in level i. In the worst case the tree will be of depth n and hence the total runtime complexity of ID3 will be $O(m \cdot n^2)$ (Utgoff, 1989). Shavlik et al. (1991) reported for ID3 an empirically based average-case complexity of $O(m \cdot n)$.

It is easy to see that the complexity of SID3 is similar to that of ID3. LSID3(r) invokes SID3 r times for each candidate split. Recalling the above analysis for the time complexity of ID3, we can

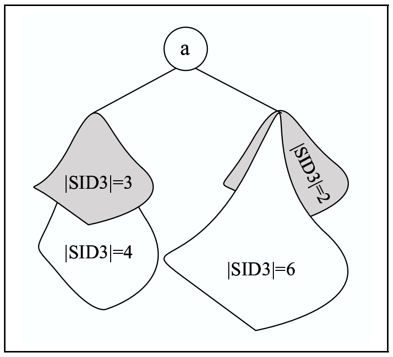


Figure 7: Attribute evaluation using LSID3. The estimated subtree size for a is $min(4,3)+min(2,6)=5$.

```
Procedure LSID3-CHOOSE-ATTRIBUTE$(E,A,r)$
    If $r=0$
        Return ID3-CHOOSE-ATTRIBUTE(E, A)
    Foreach $a \in A$
        Foreach $v_i \in$ domain$(a)$
            $E_i \leftarrow \{e \in E \mid a(e)=v_i\}$
            $min_i \leftarrow \infty$
            Repeat $r$ times
                $min_i \leftarrow \min(min_i, |\text{SID3}(E_i, A-\{a\})|)$
        $total_a \leftarrow \sum_{i=1}^{|\text{domain}(a)|} min_i$
    Return $a$ for which $total_a$ is minimal
```

Figure 8: Attribute selection in LSID3

write the complexity of LSID3(r) as

$$\sum_{i=0}^{n-1} r \cdot (n-i) \cdot O(m \cdot (n-i)^2) = \sum_{i=1}^{n} O(r \cdot m \cdot i^3) = O(r \cdot m \cdot n^4).$$

In the average case, we replace the runtime of SID3 by $O(m \cdot (n-i))$, and hence we have

$$\sum_{i=0}^{n-1} r \cdot (n-i) \cdot O(m \cdot (n-i)) = \sum_{i=1}^{n} O(r \cdot m \cdot i^2) = O(r \cdot m \cdot n^3). \quad (1)$$

According to the above analysis, the run-time of LSID3 grows at most linearly with r (under the assumption that increasing r does not result in larger trees). We expect that increasing r will improve the classifier's quality. To understand why, let us examine the expected behavior of the LSID3 algorithm on the 2-XOR problem used in Figure 1(b). LSID3 with $r=0$, which is equivalent to ID3, prefers to split on the irrelevant attribute a_4. LSID3 with $r \geq 1$ evaluates each attribute a by calling SID3 to estimate the size of the trees rooted at a. The attribute with the smallest estimation will be selected. The minimal size for trees rooted at a_4 is 6 and for trees rooted at a_3 is 7. For a_1 and a_2, SID3 would necessarily produce trees of size 4.[6] Thus, LSID3, even with $r = 1$, will succeed in learning the right tree. For more complicated problems such as 3-XOR, the space of SID3 trees under the relevant attributes includes trees other than the smallest. In that case, the larger the sample is, the higher the likelihood is that the smallest tree will be drawn.

2.4 Evaluating Continuous Attributes by Sampling the Candidate Cuts

When attributes have a continuous domain, the decision tree learner needs to discretize their values in order to define splitting tests. One common approach is to partition the range of values an attribute can take into bins, prior to the process of induction. Dougherty et al. (1995) review and compare several different strategies for pre-discretization. Using such a strategy, our lookahead algorithm can operate unchanged. Pre-discretization, however, may be harmful in many domains because the correct partitioning may differ from one context (node) to another.

An alternative approach is to determine dynamically at each node a threshold that splits the examples into two subsets. The number of different possible thresholds is at most $|E| - 1$, and thus the number of candidate tests to consider for each continuous attribute increases to $O(|E|)$. Such an increase in complexity may be insignificant for greedy algorithms, where evaluating each split requires a cheap computation (like information gain in ID3). In LSID3, however, the dynamic method may present a significant problem because evaluating the usefulness of each candidate split is very costly.

The desired method would reduce the number of splits to evaluate while avoiding the disadvantages of pre-discretization. We devised a method for controlling the resources devoted to evaluating a continuous attribute by Monte Carlo sampling of the space of splits. Initially, we evaluate each possible splitting test by the information gain it yields. This can be done in linear time $O(|E|)$. Next, we choose a sample of the candidate splitting tests where the probability that a test will be chosen is proportional to its information gain. Each candidate in the sample is evaluated by a single invocation of SID3. However, since we sample with repetitions, candidates with high information gain may have multiple instances in the sample, resulting in several SID3 invocations.

The resources allocated for evaluating a continuous attribute using the above method are determined by the sample size. If our goal is to devote a similar amount of resources to all attributes, then we can use r as the size of the sample. Such an approach, however, does not take into account the size of the population to be sampled. We use a simple alternative approach of taking samples with size $p \cdot |E|$ where p is a predetermined parameter, set by the user according to the available resources.[7] Note that p can be greater than one because we sample with repetition.

We name this variant of the LSID3 algorithm LSID3-MC, and formalize it in Figure 9. LSID3-MC can serve as an anytime algorithm that is parameterized by p and r.

2.5 Multiway vs. Binary Splits

The TDIDT procedure, as described in Figure 2, partitions the current set of examples into subsets according to the different values the splitting attribute can take. LSID3, which is a TDIDT algorithm, inherits this property. While multiway splits can yield more comprehensible trees (Kim and

6. Neither a_3 nor a_4 can be selected at the 2^{nd} level since the remaining relevant attribute reduces the entropy to zero.
7. Similarly to the parameter r, a mapping from the available time to p is needed (see Section 2.7).

```
**Procedure** MC-EVALUATE-CONTINUOUS-ATTRIBUTE($(E, A, a, p)$)
    **Foreach** $e \in E$
        $E_{\leq} \leftarrow \{e' \mid a(e') \leq a(e)\}$
        $E_{>} \leftarrow \{e' \mid a(e') > a(e)\}$
        $entropy_e \leftarrow \frac{|E_{\leq}|}{|E|}\text{entropy}(E_{\leq}) + \frac{|E_{>}|}{|E|}\text{entropy}(E_{>})$
        $gain_e \leftarrow \text{entropy}(E) - entropy_e$
    $sampleSize \leftarrow p \cdot |E|$
    $min \leftarrow \infty$
    **Repeat** $sampleSize$ **times**
        $e \leftarrow$ Choose an example at random from $E$; for each example $e$,
                the probability of selecting it is proportional to $gain_e$
        $E_{\leq} \leftarrow \{e' \mid a(e') \leq a(e)\}$
        $E_{>} \leftarrow \{e' \mid a(e') > a(e)\}$
        $total_e \leftarrow |\text{SID3}(E_{\leq}, A)| + |\text{SID3}(E_{>}, A)|$
        **If** $total_e < min$
            $min \leftarrow total_e$
            $bestTest \leftarrow a(e)$
    **Return** $\langle min, bestTest \rangle$
```

Figure 9: Monte Carlo evaluation of continuous attributes using SID3

Loh, 2001), they might fragment the data too quickly, decreasing its learnability (Hastie et al., 2001, chap. 9).

To avoid the fragmentation problem, several learners take a different approach and consider only binary splits (e.g., CART and QUEST, Loh and Shih, 1997). One way to obtain binary splits when an attribute a is categorical is to partition the examples using a single value v: all the examples with $a = v$ are bagged into one group and all the other examples are bagged into a second group. Therefore, for each attribute a there are $|\text{domain}(a)|$ candidate splits. While efficient, this strategy does not allow different values to be combined, and therefore might over-fragment the data. CART overcomes this by searching over all non-empty subsets of $\text{domain}(a)$. We denote by BLSID3 a variant of LSID3 that yields binary trees and adopts the CART style exhaustive search. Observe that the complexity of evaluating an attribute in BLSID3 is $O(2^{|\text{domain}(a)|})$ times that in LSID3.

A second problem with LSID3's multiway splits is their bias in attribute selection. Consider for example a learning task with four attributes: three binary attributes a_1, a_2, a_3, and a categorical attribute a_4 whose domain is $\{A, C, G, T\}$. Assume that the concept is $a_1 \oplus a_2$ and that a_4 was created from the values of a_2 by randomly and uniformly mapping every 0 into A or C and every 1 into G or T. Theoretically, a_2 and a_4 are equally important to the class. LSID3, however, would prefer to split a_2 due to its smaller associated total tree size.[8] BLSID3 solves this problem by considering the binary test $\in \{A, C\}$.

Loh and Shih (1997) pointed out that exhaustive search tends to prefer features with larger domains and proposed QUEST, which overcomes this problem by dealing with attribute selection and with test selection separately: first an attribute is chosen and only then is the best test picked.

8. Note that this bias is opposite to that of favoring attributes with a large domain (Quinlan, 1993, chap. 2).

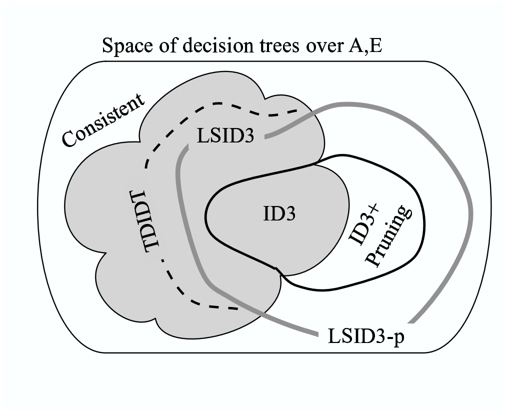


Figure 10: The space of decision trees over a set of attributes *A* and a set of examples *E*

This bias, and its reduction by QUEST, were studied in the context of greedy attribute evaluation, but can potentially be applied to our non-greedy approach.

2.6 Pruning the LSID3 Trees

Pruning tackles the problem of how to avoid overfitting the data, mainly in the presence of classification noise. It is not designed, however, to recover from wrong greedy decisions. When there are sufficient examples, wrong splits do not significantly diminish accuracy, but result in less comprehensible trees and costlier classification. When there are fewer examples the problem is intensified because the accuracy is adversely affected. In our proposed anytime approach, this problem is tackled by allotting additional time for reaching trees unexplored by greedy algorithms with pruning. Therefore, we view pruning as orthogonal to lookahead and suggest considering it as an additional phase in order to avoid overfitting.

Pruning can be applied in two different ways. First, it can be applied as a second phase after each LSID3 final tree has been built. We denote this extension by *LSID3-p* (and *BLSID3-p* for binary splits). Second, it is worthwhile to examine pruning of the lookahead trees. For this purpose, a post-pruning phase will be applied to the trees that were generated by SID3 to form the lookahead sample, and the size of the pruned trees will be considered. The expected contribution of pruning in this case is questionable since it has a similar effect on the entire lookahead sample.

Previous comparative studies did not find a single pruning method that is generally the best and conclude that different pruning techniques behave similarly (Esposito et al., 1997; Oates and Jensen, 1997). Therefore, in our experiments we adopt the same pruning method as in C4.5, namely error-based pruning (EPB), and examine post-pruning of the final trees as well as pruning of the lookahead trees.

Figure 10 describes the spaces of decision trees obtainable by LSID3. Let *A* be a set of attributes and *E* be a set of examples. The space of decision trees over *A* can be partitioned into two: the space of trees consistent with *E* and the space of trees inconsistent with *E*. The space of consistent trees includes as a subspace the trees inducible by TDIDT.[9] The trees that ID3 can produce are a subspace of the TDIDT space. Pruning extends the space of ID3 trees to the inconsistent space. LSID3

9. TDIDT is a strict subspace because it cannot induce trees that: (1) contain a subtree with all leaves marked with the same class, and (2) include an internal node that has no associated training examples.

can reach trees that are obtainable by TDIDT but not by the greedy methods and their extensions. LSID3-p expands the search space to trees that do not fit the training data perfectly, so that noisy training sets can be handled.

2.7 Mapping Contract Time to Sample Size

LSID3 is parameterized by r, the number of times SID3 is invoked for each candidate. In real-life applications, however, the user supplies the system with the time she is willing to allocate for the learning process. Hence, a mapping from the contract time to r is needed.

One way to obtain such a mapping is to run the algorithm with $r = 1$. Since the runtime grows linearly with r, one can now, given the remaining time and the runtime for $r = 1$, easily find the corresponding value of r. Then, the whole induction process is restarted with the measured r as the contract parameter. The problem with this method is that, in many cases, running the algorithm with $r = 1$ consumes many of the available resources while not contributing to the final classifier.

Another possibility is mapping time to r on the basis of previous experience on the same domain or a domain with similar properties. When no such knowledge exists, one can derive the runtime from the time complexity analysis. However, in practice, the runtime is highly dependent on the complexity of the domain and the number of splits that in fact take place. Hence it is very difficult to predict.

3. Interruptible Anytime Learning of Decision Trees

As a contract algorithm, LSID3 assumes that the contract parameter is known in advance. In many real-life cases, however, either the contract time is unknown or mapping the available resources to the appropriate contract parameter is not possible. To handle such cases, we need to devise an interruptible anytime algorithm.

We start with a method that converts LSID3 to an interruptible algorithm using the general sequencing technique described by Russell and Zilberstein (1996). This conversion, however, uses the contract algorithm as a black box and hence cannot take into account specific aspects of decision tree induction. Next, we present a novel iterative improvement algorithm, *Interruptible Induction of Decision Trees* (IIDT), that repeatedly replaces subtrees of the current hypothesis with subtrees generated with higher resource allocation and thus expected to be better.

3.1 Interruptible Induction by Sequencing Contract Algorithms

By definition, every interruptible algorithm can serve as a contract algorithm because one can stop the interruptible algorithm when all the resources have been consumed. Russell and Zilberstein (1996) showed that the other direction works as well: any contract algorithm $\mathcal{A}$ can be converted into an interruptible algorithm $\mathcal{B}$ with a constant penalty. $\mathcal{B}$ is constructed by running $\mathcal{A}$ repeatedly with exponentially increasing time limits. This general approach can be used to convert LSID3 into an interruptible algorithm. LSID3 gets its contract time in terms of r, the sample size. When $r = 0$, LSID3 is defined to be identical to ID3. Therefore, we first call LSID3 with $r = 0$ and then continue with exponentially increasing values of r, starting from $r = 1$. Figure 11 formalizes the resulting algorithm. It can be shown that the above sequence of runtimes is optimal when the different runs are scheduled on a single processor (Russell and Zilberstein, 1996).

```
Procedure SEQUENCED-LSID3($E$,$A$)
    $T \leftarrow$ ID3($E$,$A$)
    $r \leftarrow 1$
    While not-interrupted
        $T \leftarrow$ LSID3($E$,$A$,$r$)
        $r \leftarrow 2 \cdot r$
    Return $T$
```

Figure 11: Conversion of LSID3 to an interruptible algorithm by sequenced invocations

One problem with the sequencing approach is the exponential growth of the gaps between the times at which an improved result can be obtained. The reason for this problem is the generality of the sequencing approach, which views the contract algorithm as a black box. Thus, in the case of LSID3, at each iteration the whole decision tree is rebuilt. In addition, the minimal value that can be used for τ is the runtime of LSID3($r = 1$). In many cases we would like to stop the algorithm earlier. When we do, the sequencing approach will return the initial greedy tree. Our interruptible anytime framework, called IIDT, overcomes these problems by iteratively improving the tree rather than trying to rebuild it.

3.2 Interruptible Induction by Iterative Improvement

Iterative improvement approaches that start with an initial solution and repeatedly modify it have been successfully applied to many AI domains, such as CSP, timetabling, and combinatorial problems (Minton et al., 1992; Johnson et al., 1991; Schaerf, 1996). The key idea behind iterative improvement techniques is to choose an element of the current suboptimal solution and improve it by local repairs. IIDT adopts the above approach for decision tree learning and iteratively revises subtrees. It can serve as an interruptible anytime algorithm because, when interrupted, it can immediately return the currently best solution available.

As in LSID3, IIDT exploits additional resources in an attempt to produce better decision trees. The principal difference between the algorithms is that LSID3 uses the available resources to induce a decision tree top-down, where each decision made at a node is final and does not change. IIDT, however, is not allocated resources in advance and uses extra time resources to modify split decisions repeatedly.

IIDT receives a set of examples and a set of attributes. It first performs a quick construction of an initial decision tree by calling ID3. Then it iteratively attempts to improve the current tree by choosing a node and rebuilding its subtree with more resources than those used previously. If the newly induced subtree is better, it will replace the existing one. IIDT is formalized in Figure 12.

Figure 13 illustrates how IIDT works. The target concept is $a_1 \oplus a_2$, with two additional irrelevant attributes, a_3 and a_4. The leftmost tree was constructed using ID3. In the first iteration, the subtree rooted at the bolded node is selected for improvement and replaced by a smaller tree (surrounded by a dashed line). Next, the root is selected for improvement and the whole tree is replaced by a tree that perfectly describes the concept. While in this particular example IIDT first chooses to rebuild a subtree at depth 2 and then at depth 1, it considers all subtrees, regardless of their level.

```
Procedure IIDT($E$,$A$)
    $T \leftarrow$ ID3($E$,$A$)
    While not-interrupted
        $node \leftarrow$ CHOOSE-NODE($T$,$E$,$A$)
        $t \leftarrow$ subtree of $T$ rooted at $node$
        $A_{node} \leftarrow \{a \in A \mid a \notin$ ancestor of $node\}$
        $E_{node} \leftarrow \{e \in E \mid e$ reaches $node\}$
        $r \leftarrow$ NEXT-R($node$)
        $t' \leftarrow$ REBUILD-TREE($E_{node}$,$A_{node}$,$r$)
        If EVALUATE($t$) > EVALUATE($t'$)
            replace $t$ with $t'$
    Return $T$
```

Figure 12: Interruptible induction of decision trees

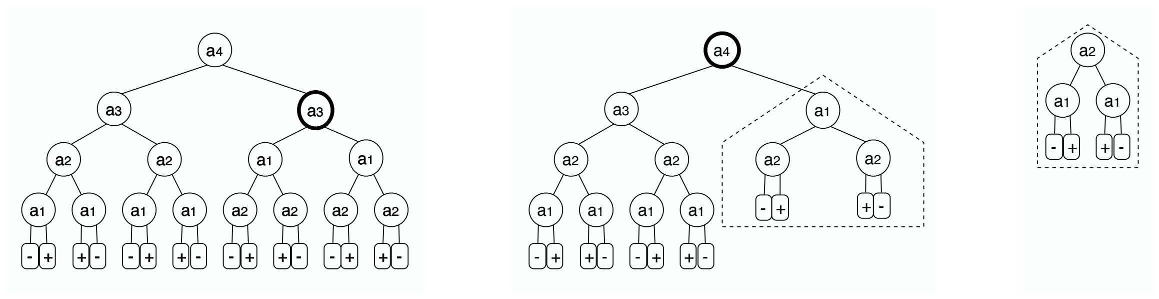


Figure 13: Iterative improvement of the decision tree produced for the 2-XOR concept $a_1 \oplus a_2$ with two additional irrelevant attributes, a_3 and a_4. The leftmost tree was constructed using ID3. In the first iteration the subtree rooted at the bolded node is selected for improvement and replaced by a smaller tree (surrounded by a dashed line). Next, the root is selected for improvement and the whole tree is replaced by a tree that perfectly describes the concept.

IIDT is designed as a general framework for interruptible learning of decision trees. It can use different approaches for choosing which node to improve, for allocating resources for an improvement iteration, for rebuilding a subtree, and for deciding whether an alternative subtree is better.

After the subtree to be rebuilt is chosen and the resources for a reconstruction iteration allocated, the problem becomes a task for a contract algorithm. A good candidate for such an algorithm is LSID3, which is expected to produce better subtrees when invoked with a higher resource allocation. In what follows we focus on the different components of IIDT and suggest a possible implementation that uses LSID3 for revising subtrees.

3.2.1 CHOOSING A SUBTREE TO IMPROVE

Intuitively, the next node we would like to improve is the one with the highest expected marginal utility, that is, the one with the highest ratio between the expected benefit and the expected cost (Hovitz, 1990; Russell and Wefald, 1989). Estimating the expected gain and expected cost of rebuilding a subtree is a difficult problem. There is no apparent way to estimate the expected improvement in terms of either tree size or generalization accuracy. In addition, the resources to be consumed by LSID3 are difficult to predict precisely. We now show how to approximate these values, and how to incorporate these approximations into the node selection procedure.

Resource Allocation. The LSID3 algorithm receives its resource allocation in terms of r, the number of samplings devoted to each attribute. Given a tree node y, we can view the task of rebuilding the subtree below y as an independent task. Every time y is selected, we have to allocate resources for the reconstruction process. Following Russell and Zilberstein (1996), the optimal strategy in this case is to double the amount of resources at each iteration. Thus, if the resources allocated for the last attempted improvement of y were $\text{LAST-R}(y)$, the next allocation will be $\text{NEXT-R}(y) = 2 \cdot \text{LAST-R}(y)$.

Expected Cost. The expected cost can be approximated using the average time complexity of the contract algorithm used to rebuild subtrees. Following Equation 1, we estimate $\text{NEXT-R}(y) \cdot m \cdot n^3$ to be the expected runtime of LSID3 when rebuilding a node y, where m is the number of examples that reach y and n is the number of attributes to consider. We observe that subtrees rooted in deeper levels are preferred because they have fewer examples and attributes to consider. Thus, their expected runtime is shorter. Furthermore, because each time allocation for a node doubles the previous one, nodes that have already been selected many times for improvement will have higher associated costs and are less likely to be chosen again.

Expected benefit. The whole framework of decision tree induction rests on the assumption that smaller consistent trees are better than larger ones. Therefore, the size of a subtree can serve as a measure for its quality. It is difficult, however, to estimate the size of the reconstructed subtree without actually building it. Therefore, we use instead an upper limit on the possible reduction in size. The minimal size possible for a decision tree is obtained when all examples are labelled with the same class. Such cases are easily recognized by the greedy ID3. Similarly, if a subtree were replaceable by another subtree of depth 1, ID3 (and LSID3) would have chosen the smaller subtree. Thus, the maximal reduction of the size of an existing subtree is to the size of a tree of depth 2. Assuming that the maximal number of values per attribute is b, the maximal size of such a tree is b^2. Hence, an upper bound on the benefit from reconstructing an existing tree t is $\text{SIZE}(t) - b^2$. Ignoring the expected costs and considering only the expected benefits results in giving the highest score to the root node. This makes sense: assuming that we have infinite resources, we would attempt to improve the entire tree rather than parts of it.

Granularity. Considering the cost and benefit approximations described above, the selection procedure would prefer deep nodes (that are expected to have low costs) with large subtrees (that are expected to yield large benefits). When no such large subtrees exist, our algorithm may repeatedly attempt to improve smaller trees rooted at deep nodes because these trees have low associated costs. In the short term, this behavior would indeed be beneficial but can be harmful in the long term. This is because when the algorithm later improves subtrees in upper levels, the resources spent on deeper

```
**Procedure** CHOOSE-NODE$(T, E, A, g)$
    $max\text{-}cost \leftarrow$ NEXT-R$(root) \cdot |E| \cdot |A|^3$
    **Foreach** $node \in T$
        $A_{node} \leftarrow \{a \in A \mid a \notin$ ancestor of $node\}$
        $E_{node} \leftarrow \{e \in E \mid e$ reaches $node\}$
        $r_{node} \leftarrow$ NEXT-R$(node)$
        $cost_{node} \leftarrow r_{node} \cdot |E_{node}| \cdot |A_{node}|^3$
        **If** $(cost_{node}/max\text{-}cost) > g$
            $l\text{-}bound \leftarrow (\min_{a \in A_{node}} |\text{DOMAIN}(a)|)^2$
            $\Delta q \leftarrow$ LEAVES$(node) - l\text{-}bound$
            $u_{node} \leftarrow \Delta q / cost_{node}$
    $best \leftarrow node$ that maximizes $u_{node}$
    **Return** $\langle best, r_{best} \rangle$

**Procedure** NEXT-R$(node)$
    **If** LAST-R$(node) = 0$
        **Return** 1
    **Else**
        **Return** $2 \cdot$ LAST-R$(node)$
```

Figure 14: Choosing a node for reconstruction

nodes will have been wasted. Had the algorithm first selected the upper level trees, this waste would have been avoided, but the time gaps between potential improvements would have increased.

To control the tradeoff between efficient resource use and anytime performance flexibility, we add a granularity parameter $0 \leq g \leq 1$. This parameter serves as a threshold for the minimal time allocation for an improvement phase. A node can be selected for improvement only if its normalized expected cost is above g. To compute the normalized expected cost, we divide the expected cost by the expected cost of the root node. Note that it is possible to have nodes with a cost that is higher than the cost of the root node, since the expected cost doubles the cost of the last improvement of the node. Therefore, the normalized expected cost can be higher than 1. Such nodes, however, will never be selected for improvement, because their expected benefit is necessarily lower than the expected benefit of the root node. Hence, when $g = 1$, IIDT is forced to choose the root node and its behavior becomes identical to that of the sequencing algorithm described in Section 3.1.

Figure 14 formalizes the procedure for choosing a node for reconstruction. Observe that IIDT does not determine g but expects the user to provide this value according to her needs: more frequent small improvements or faster overall progress.

3.2.2 EVALUATING A SUBTREE

Although LSID3 is expected to produce better trees when allocated more resources, an improved result is not guaranteed. Thus, to avoid obtaining an induced tree of lower quality, we replace an existing subtree with a newly induced alternative only if the alternative is expected to improve the quality of the complete decision tree. Following Occam's Razor, we measure the usefulness of a subtree by its size. Only if the reconstructed subtree is smaller does it replace an existing subtree. This guarantees that the size of the complete decision tree will decrease monotonically.

Another possible measure is the accuracy of the decision tree on a set-aside validation set of examples. In this case the training set is split into two subsets: a growing set and a validation set. Only if the accuracy on the validation set increases is the modification applied. This measure suffers from two drawbacks. The first is that putting aside a set of examples for validation results in a smaller set of training examples, making the learning process harder. The second is the bias towards overfitting the validation set, which might reduce the generalization abilities of the tree. Several of our experiments, which we do not report here, confirmed that relying on the tree size results in better decision trees.

4. Empirical Evaluation

A variety of experiments were conducted to test the performance and behavior of the proposed anytime algorithms. First we describe our experimental methodology and explain its motivation. We then present and discuss our results.

4.1 Experimental Methodology

We start our experimental evaluation by comparing our contract algorithm, given a fixed resource allocation, with the basic decision tree induction algorithms. We then compare the anytime behavior of our contract algorithm to that of fixed lookahead. Next we examine the anytime behavior of our interruptible algorithm. Finally, we compare its performance to several modern decision tree induction methods.

Following the recommendations of Bouckaert (2003), 10 runs of a 10-fold cross-validation experiment were conducted for each data set and the reported results averaged over the 100 individual runs.[10] For the *Monks* data sets, which were originally partitioned into a training set and a testing set, we report the results on the original partitions. Due to the stochastic nature of LSID3, the reported results in these cases are averaged over 10 different runs.

In order to evaluate the studied algorithms, we used 17 data sets taken from the UCI repository (Blake and Merz, 1998).[11] Because greedy learners perform quite well on easy tasks, we looked for problems that hide hard concepts so the advantage of our proposed methods will be emphasized. The UCI repository, nevertheless, contains only few such tasks. Therefore, we added 7 artificial ones.[12] Several commonly used UCI data sets were included (among the 17) to allow comparison with results reported in literature. Table 1 summarizes the characteristics of these data sets while Appendix A gives more detailed descriptions. To compare the performance of the different algorithms, we will consider two evaluation criteria over decision trees: (1) *generalization accuracy*, measured by the ratio of the correctly classified examples in the testing set, and (2) *tree size*, measured by the number of non-empty leaves in the tree.

10. An exception was the *Connect-4* data set, for which only one run of 10-fold CV was conducted because of its enormous size.
11. Due to the time-intensiveness of the experiments, we limited ourselves to 17 UCI problems.
12. The artificial data sets are available at http://www.cs.technion.ac.il/∼esaher/publications/datasets.

DATA SET	INSTANCES	ATTRIBUTES NOMINAL (BINARY)	ATTRIBUTES NUMERIC	MAX ATTRIBUTE DOMAIN	CLASSES
AUTOMOBILE - MAKE	160	10 (4)	15	8	22
AUTOMOBILE - SYMBOLING	160	10 (4)	15	22	7
BALANCE SCALE	625	4 (0)	0	5	3
BREAST CANCER	277	9 (3)	0	13	2
CONNECT-4	68557	42 (0)	0	3	3
CORRAL	32	6 (6)	0	2	2
GLASS	214	0 (0)	9	-	7
IRIS	150	0 (0)	4	-	3
MONKS-1	124+432	6 (2)	0	4	2
MONKS-2	169+432	6 (2)	0	4	2
MONKS-3	122+432	6 (2)	0	4	2
MUSHROOM	8124	22 (4)	0	12	2
SOLAR FLARE	323	10 (5)	0	7	4
TIC-TAC-TOE	958	9 (0)	0	3	2
VOTING	232	16 (16)	0	2	2
WINE	178	0 (0)	13	-	3
ZOO	101	16 (15)	0	6	7
NUMERIC XOR 3D	200	0 (0)	6	-	2
NUMERIC XOR 4D	500	0 (0)	8	-	2
MULTIPLEXER-20	615	20 (20)	0	2	2
MULTIPLEX-XOR	200	11 (11)	0	2	2
XOR-5	200	10 (10)	0	2	2
XOR-5 10% NOISE	200	10 (10)	0	2	2
XOR-10	10000	20 (20)	0	2	2

Table 1: Characteristics of the data sets used

4.2 Fixed Time Comparison

Our first set of experiments compares ID3, C4.5 with its default parameters, ID3-k$(k=2)$, LSID3$(r=5)$ and LSID3-p$(r=5)$. We used our own implementation for all algorithms, where the results of C4.5 were validated with WEKA's implementation (Witten and Frank, 2005). We first discuss the results for the consistent trees and continue by analyzing the findings when pruning is applied.

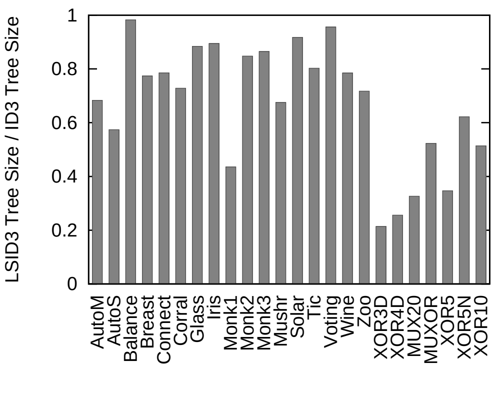
4.2.1 CONSISTENT TREES

Figure 15 illustrates the differences in tree size and generalization accuracy of LSID3(5) and ID3. Figure 16 compares the performance of LSID3 to that of ID3-k. The full results, including significance tests, are available in Appendix B.

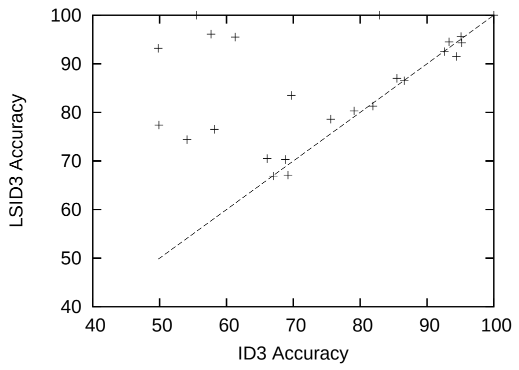
When comparing the algorithms that produce consistent trees, namely ID3, ID3-k and LSID3, the average tree size is the smallest for most data sets when the trees are induced with LSID3. In all cases, as Figure 15(a) implies, LSID3 produced smaller trees than ID3 and these improvements were found to be significant. The average reduction in size is 26.5% and for some data sets, such as *XOR-5* and *Multiplexer-20*, it is more than 50%. ID3-k produced smaller trees than ID3 for most but not all of the data sets (see Figure 16, a).

In the case of synthetic data sets, the optimal tree size can be found in theory.[13] For instance, the tree that perfectly describes the n XOR concept is of size 2^n. The results show that in this sense, the trees induced by LSID3 were almost optimal.

13. Note that a theoretically optimal tree is not necessarily obtainable from a given training set.

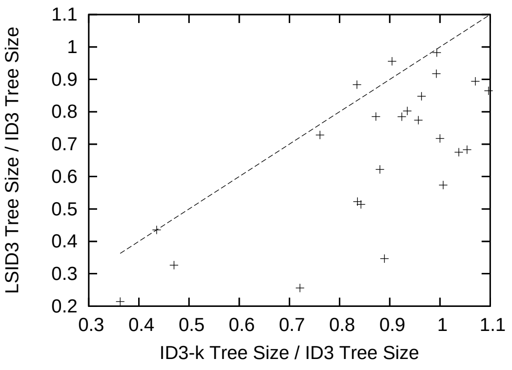


(a) Tree Size

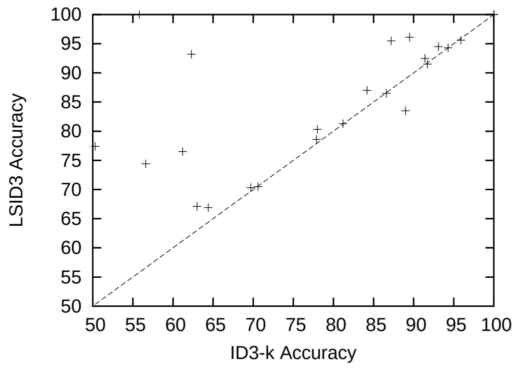


(b) Accuracy

Figure 15: Illustration of the differences in performance between LSID3(5) and ID3. The left-side figure gives the relative size of the trees produced by LSID3 in comparison to ID3. The right-side figure plots the accuracy achieved by both algorithms. Each point represents a data set. The *x*-axis represents the accuracy of ID3 while the *y*-axis represents that of LSID3. The dashed line indicates equality. Points are above it if LSID3 performs better and below it if ID3 is better.



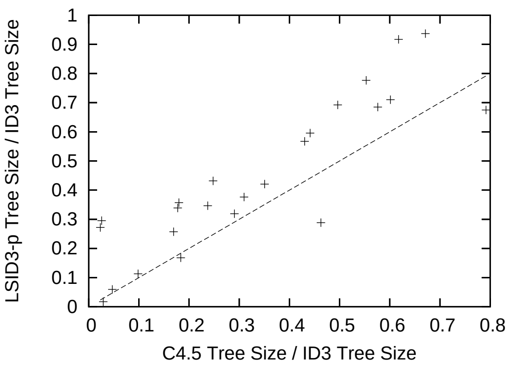
(a) Tree Size



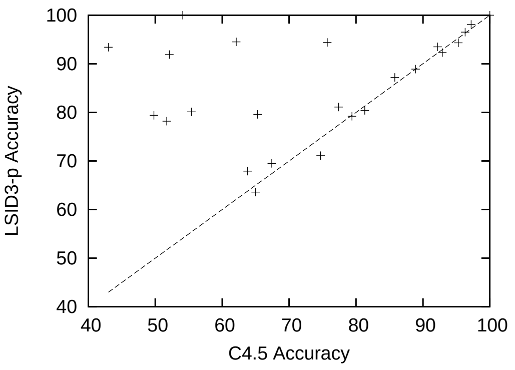
(b) Accuracy

Figure 16: Performance differences for LSID3(5) and ID3-k. The left-side figure compares the size of trees induced by each algorithm, measured relative to ID3 (in percents). The right-side figure plots the absolute differences in terms of accuracy.

Reducing the tree size is usually beneficial only if the associated accuracy is not reduced. Analyzing the accuracy of the produced trees shows that LSID3 significantly outperforms ID3 for most data sets. For the other data sets, the t-test values indicate that the algorithms are not significantly



(a) Tree Size



(b) Accuracy

Figure 17: Performance differences for LSID3-p(5) and C4.5. The left-side figure compares the size of trees induced by each algorithm, measured relative to ID3 (in percents). The right-side figure plots the absolute differences in terms of accuracy.

different. The average absolute improvement in the accuracy of LSID3 over ID3 is 11%. The Wilcoxon test (Demsar, 2006), which compares classifiers over multiple data sets, indicates that the advantage of LSID3 over ID3 is significant, with $\alpha = 0.05$.

The accuracy achieved by ID3-k, as shown in Figure 16(b), is better than that of ID3 on some data sets. ID3-k achieved similar results to LSID3 for some data sets, but performed much worse for others, such as *Tic-tac-toe* and *XOR-10*. For most data sets, the decrease in the size of the trees induced by LSID3 is accompanied by an increase in predictive power. This phenomenon is consistent with Occam's Razor.

4.2.2 PRUNED TREES

Pruning techniques help to avoid overfitting. We view pruning as orthogonal to our lookahead approach. Thus, to allow handling noisy data sets, we tested the performance of LSID3-p, which post-prunes the LSID3 trees using error-based pruning.

Figure 17 compares the performance of LSID3-p to that of C4.5. Applying pruning on the trees induced by LSID3 makes it competitive with C4.5 on noisy data. Before pruning, C4.5 outperformed LSID3 on the *Monks-3* problem, which is known to be noisy. However, LSID3 was improved by pruning, eliminating the advantage C4.5 had. For some data sets, the trees induced by C4.5 are smaller than those learned by LSID3-p. However, the results indicate that among the 21 tasks for which t-test is applicable,[14] LSID3-p was significantly more accurate than C4.5 on 11, significantly worse on 2, and similar on the remaining 8. Taking into account all 24 data sets, the overall improvement by LSID3-p over C4.5 was found to be statistically significant by a Wilcoxon test with $\alpha = 0.05$. In general, LSID3-p performed as well as LSID3 on most data sets, and significantly better on the noisy ones.

14. The t-test is not applicable for the Monk data sets because only 1 train-test partition was used.

These results confirm our expectations: the problems addressed by LSID3 and C4.5's pruning are different. While LSID3 allots more time for better learning of hard concepts, pruning attempts to simplify the induced trees to avoid overfitting the data. The combination of LSID3 and pruning is shown to be worthwhile: it enjoys the benefits associated with lookahead without the need to compromise when the training set is noisy.

We also examined the effect of applying error-based pruning not only to the final tree, but to the lookahead trees as well. The experiments conducted on several noisy data sets showed that the results of this extra pruning phase were very similar to the results without pruning. Although pruning results in samples that better represent the final trees, it affects all samples similarly and hence does not lead to different split decisions.

4.2.3 BINARY SPLITS

By default, LSID3 uses multiway splits, that is, it builds a subtree for each possible value of a nominal attribute. Following the discussion in Section 2.5, we also tested how LSID3 performs if binary splits are forced. The tests in this case are found using exhaustive search.

To demonstrate the fragmentation problem, we used two data sets. The first data set is *Tic-tac-toe*. When binary splits were forced, the performance of both C4.5 and LSID3-p improved from 85.8 and 87.2 to 94.1 and 94.8 respectively. As in the case of multiway splits, the advantage of LSID3-p over C4.5 is statistically significant with $\alpha = 0.05$. Note that binary splits come at a price: the number of candidate splits increases and the runtime becomes significantly longer. When allocated the same time budget, LSID3-p can afford larger samples than BLSID3-p. The advantage of the latter, however, is kept.

The second task is a variant on *XOR-2*, where there are 3 attributes a_1, a_2, a_3, each of which can take one of the values A, C, G, T. The target concept is $a_1^* \oplus a_2^*$. The values of a_i^* are obtained from a_i by mapping each A and C to 0 and each G and T to 1. The data set, referred to as *Categorial XOR-2*, consists of 16 randomly drawn examples. With multiway splits, both LSID3-p and C4.5 could not learn *Categorial XOR-2* and their accuracy was about 50%. C4.5 failed also with binary splits. BLSID3-p, on the contrary, was 92% accurate.

We also examined the bias of LSID3 toward binary attributes. For this purpose we used the example described in Section 2.5. An artificial data set with all possible values was created. LSID3 with multiway and with binary splits were run 10000 times. Although a_4 is as important to the class as a_1 and a_2, LSID3 with multiway splits never chose it at the root. LSID3 with binary splits, however, split the root on a_4 35% of the time. These results were similar to a_1 (33%) and a_2 (32%). They indicate that forcing binary splits removes the LSID3 bias.

4.3 Anytime Behavior of the Contract Algorithms

Both LSID3 and ID3-k make use of additional resources for generating better trees. However, in order to serve as good anytime algorithms, the quality of their output should improve with the increase in their allocated resources. For a typical anytime algorithm, this improvement is greater at the beginning and diminishes over time. To test the anytime behavior of LSID3 and ID3-k, we invoked them with successive values of r and k respectively. In the first set of experiments we focused on domains with nominal attributes, while in the second set we tested the anytime behavior for domains with continuous attributes.

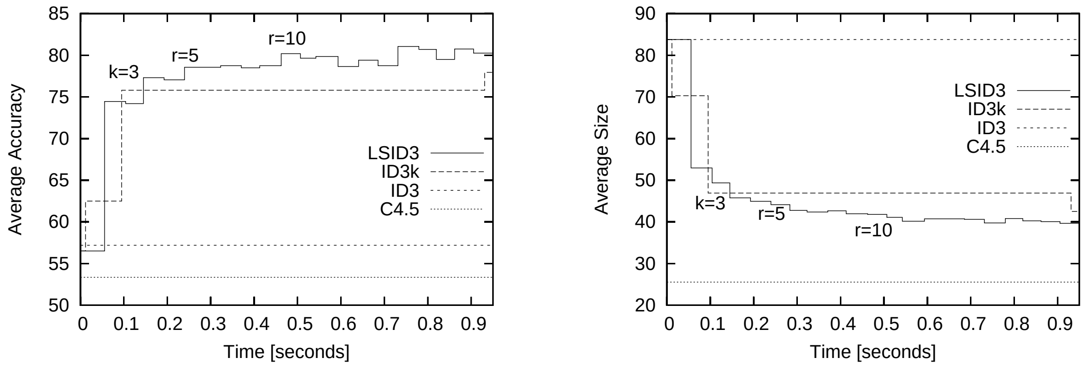


Figure 18: Anytime behavior of ID3-k and LSID3 on the *Multiplex-XOR* data set

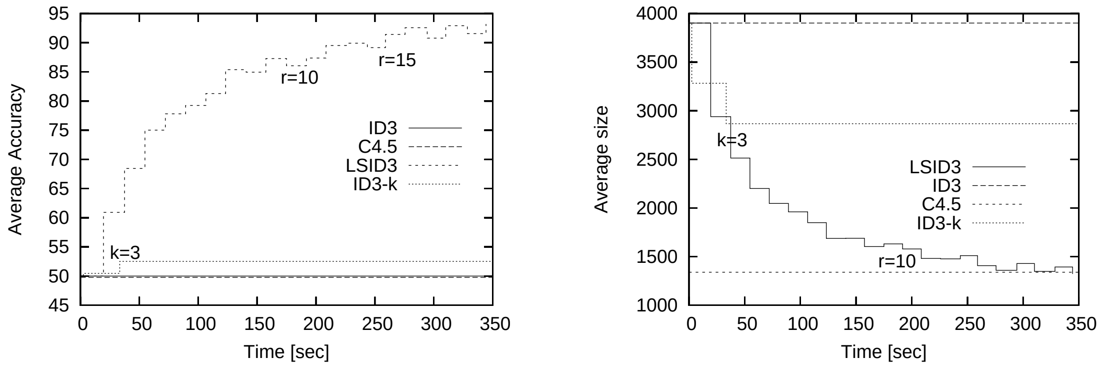


Figure 19: Anytime behavior of ID3-k and LSID3 on the *10-XOR* data set

4.3.1 NOMINAL ATTRIBUTES

Figures 18, 19 and 20 show the average results over 10 runs of 10-fold cross-validation experiments for the *Multiplex-XOR*, *XOR-10* and *Tic-tac-toe* data sets respectively. The *x*-axis represents the run time in seconds.[15] ID3 and C4.5, which are constant time algorithms, terminate quickly and do not improve with time. Since ID3-k with $k = 1$ and LSID3 with $r = 0$ are defined to be identical to ID3, the point at which ID3 yields a result serves also as the starting point of these anytime algorithms.

The graphs indicate that the anytime behavior of LSID3 is better than that of ID3-k. For ID3-k, the gaps between the points (width of the steps) increase exponentially, although successive values of *k* were used. As a result, any extra time budget that falls into one of these gaps cannot be exploited. For example, when run on the *XOR-10* data set, ID3-k is unable to make use of additional time that is longer than 33 seconds $(k = 3)$ but shorter than 350 seconds $(k = 4)$. For LSID3, the difference in the time required by the algorithm for any 2 successive values of *r* is almost the same.

For the *Multiplex-XOR* data set, the tree size and generalization accuracy improve with time for both LSID3 and ID3-k, and the improvement decreases with time. Except for a short period of time, LSID3 dominates ID3-k. For the *XOR-10* data set, LSID3 has a great advantage: while ID3-k

15. The algorithms were implemented in C++, compiled with GCC, and run on Macintosh G5 2.5 GHz.

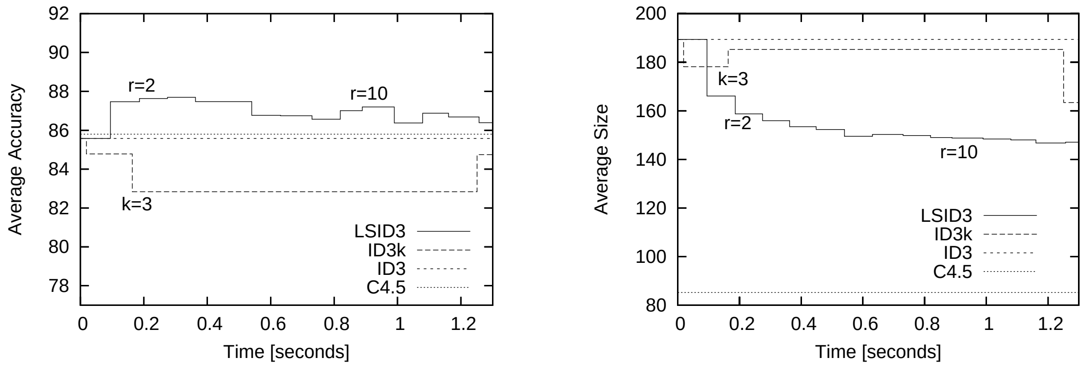


Figure 20: Anytime behavior of ID3-k and LSID3 on the *Tic-tac-toe* data set

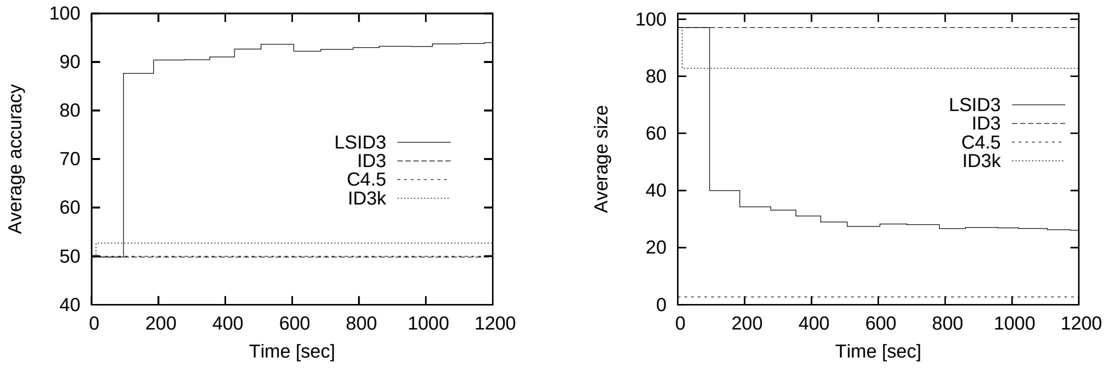


Figure 21: Anytime behavior of ID3-k, LSID3 on the *Numeric-XOR 4D* data set

produced trees whose accuracy was limited to 55%, LSID3 reached an average accuracy of more than 90%.

In the experiment with the *Tic-tac-toe* data set, LSID3 dominated ID3-k consistently, both in terms of accuracy and size. ID3-k performs poorly in this case. In addition to the large gaps between successive possible time allocations, a decrease in accuracy and an increase in tree size are observed at $k = 3$. Similar cases of pathology caused by limited-depth lookahead have been reported by Murthy and Salzberg (1995). Starting from $r = 5$, the accuracy of LSID3 does not improve over time and sometimes slightly declines (but still dominates ID3-k). We believe that the multiway splits prevent LSID3 from further improvements. Indeed, our experiments in Section 4.2 indicate that LSID3 can perform much better with binary splits.

4.3.2 CONTINUOUS ATTRIBUTES

Our next anytime-behavior experiment uses the *Numeric-XOR 4D* data set with continuous attributes. Figure 21 gives the results for ID3, C4.5, LSID3, and ID3-k. LSID3 clearly outperforms all the other algorithms and exhibits good anytime behavior. Generalization accuracy and tree size both improve with time. ID3-k behaves poorly in this case. For example, when 200 seconds are allocated, we can run LSID3 with $r = 2$ and achieve accuracy of about 90%. With the same allocation, ID3-k can be run with $k = 2$ and achieve accuracy of about 52%. The next improvement of

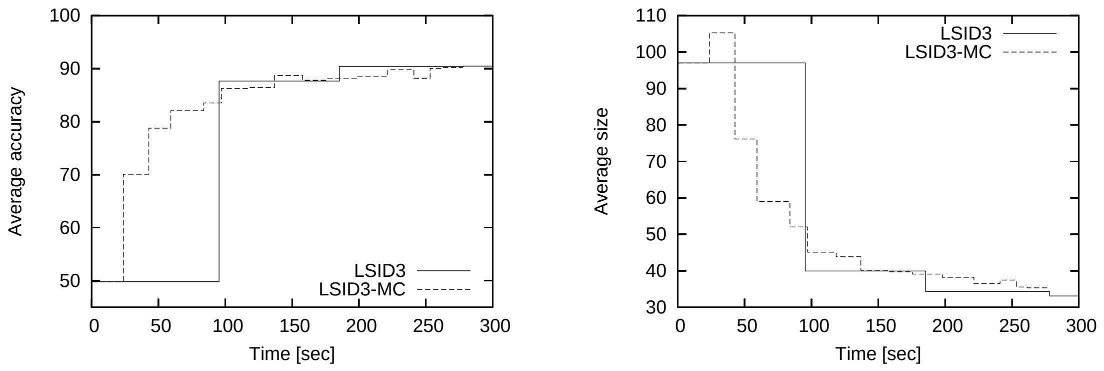


Figure 22: Anytime behavior of LSID3-MC on the *Numeric-XOR 4D* data set

ID3-k (with $k = 3$) requires 10,000 seconds. But even with such a large allocation (not shown in the graph since it is off the scale), the resulting accuracy is only about 66%.

In Section 2.4 we described the LSID3-MC algorithm which, instead of uniformly distributing evaluation resources over all possible splitting points, performs biased sampling towards points with high information gain. Figure 22 compares the anytime behavior of LSID3-MC to that of LSID3. The graph of LSID3 shows, as before, the performance for successive values of r. The graph of LSID3 shows the performance for $p = 10\%, 20\%, \ldots, 150\%$. A few significant conclusions can be drawn from these results:

1. The correlation between the parameter p and the runtime is almost linear: the steps in the graph are of almost constant duration.[16] We can easily increase the granularity of the anytime graph by smaller gaps between the p values.

2. It looks as if the runtime for LSID3-MC with $p = 100\%$ should be the same as LSID3($r = 1$) without sampling where all candidates are evaluated once. We can see, however, that the runtime of LSID3($r = 1$) is not sufficient for running LSID3-MC with $p = 50\%$. This is due to the overhead associated with the process of ordering the candidates prior to sample selection.

3. LSID3-MC with $p = 100\%$ performs better than LSID3($r = 1$). This difference can be explained by the fact that we performed a biased sample with repetitions, and therefore more resources were devoted to more promising tests rather than one repetition for each point as in LSID3($r = 1$).

4. When the available time is insufficient for running LSID3($r = 1$) but more than sufficient for running ID3, LSID3-MC is more flexible and allows these intermediate points of time to be exploited. For instance, by using only one-fifth of the time required by LSID3($r = 1$), an absolute accuracy improvement of 20% over ID3 was achieved.

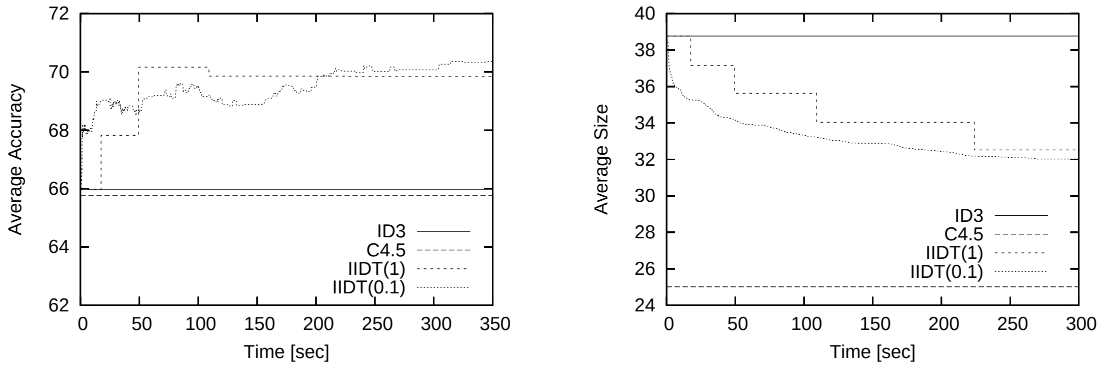


Figure 23: Anytime behavior of IIDT on the *Glass* data set

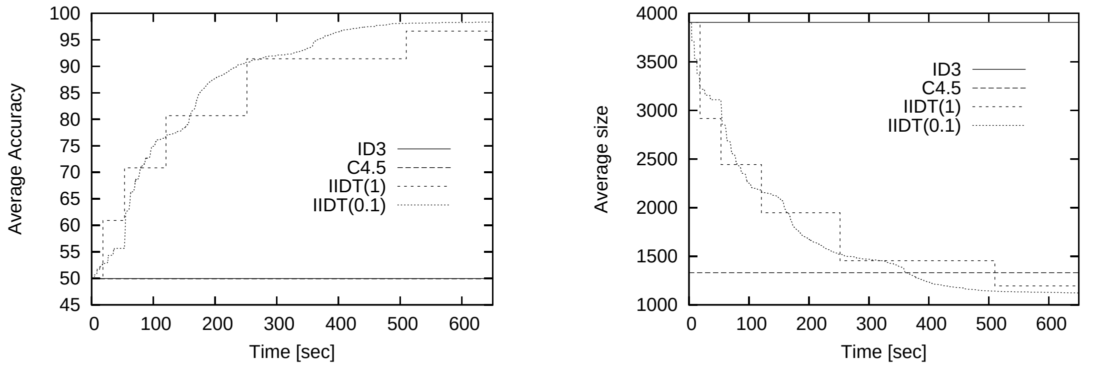


Figure 24: Anytime behavior of IIDT on the *10-XOR* data set

4.4 Anytime Behavior of IIDT

IIDT was presented as an interruptible decision tree learner that does not require advanced knowledge of its resource allocation: it can be stopped at any moment and return a valid decision tree. We tested two versions of IIDT, the first with granularity threshold $g = 0.1$ and the second with $g = 1$. Figures 23, 24, and 25 show the anytime performance of IIDT in terms of tree size and accuracy for the *Glass*, *XOR-10*, and *Tic-tac-toe* data sets. Each graph represents an average of 100 runs (for the 10×10 cross-validation). Unlike the graphs given in the previous section, these are interruptible anytime graphs, that is, for each point, the *y* coordinate reflects the performance if the algorithm was interrupted at the associated *x* coordinate. In the contract algorithm graphs, however, each point reflects the performance if the algorithm was initially allocated the time represented by the *x* coordinate.

In all cases, the two anytime versions indeed exploit the additional resources and produce both smaller and more accurate trees. Since our algorithm replaces a subtree only if the new one is smaller, all size graphs decrease monotonically. The most interesting anytime behavior is for the difficult *XOR-10* problem. There, the tree size decreases from 4000 leaves to almost the optimal size (1024), and the accuracy increases from 50% (which is the accuracy achieved by ID3 and C4.5)

16. Some steps look as though they require durations that are twice as long. However, these durations actually represent two *p* values with identical results.

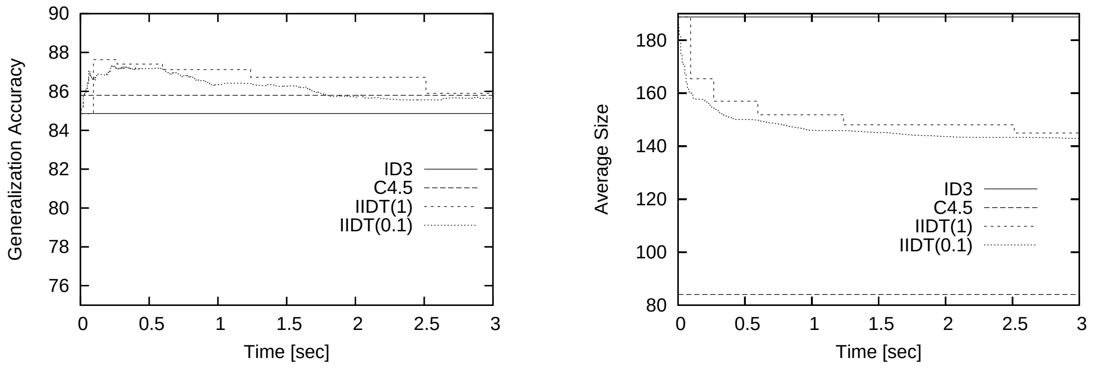


Figure 25: Anytime behavior of IIDT on the *Tic-tac-toe* data set

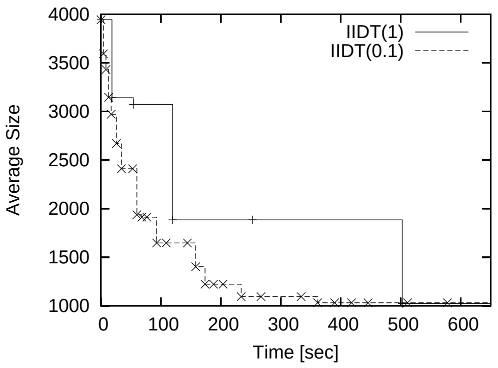


Figure 26: Time steps for a single run on *10-XOR*

to almost 100%. The shape of the graphs is typical to those of anytime algorithms with diminishing returns. The improvement in the accuracy of IIDT (at the latest point it was measured) over ID3 and C4.5 was found by t-test ($\alpha = 0.05$) to be significant for the *Glass* and *XOR-10* data sets. The performance of IIDT on *Tic-tac-toe* slightly degrades over time. We believe that similarly to LSID3, IIDT can perform much better if binary splits are used.

The difference in performance of the two anytime algorithms is interesting. IIDT(0.1), with the lower granularity parameter, indeed produces smoother anytime graphs (with lower volatility), which allows for better control and better predictability of return. Moreover, in large portions of the time axis, the IIDT(0.1) graph shows better performance than that of IIDT(1). This is due to more sophisticated node selection in the former. Recall that $g = 1$ means that the algorithm always selects the entire tree for improvement.

The smoothness of the IIDT(0.1) graphs is somehow misleading because it represents an average of 100 runs, with each step taking place at a different time (this is in contrast to the graph for IIDT(1), where the steps are at roughly the same times). Figure 26 illustrates the significance of this smoothing effect on a single anytime graph (out of the 100). We can see that although the IIDT(0.1) graph is less smooth than the average graph, it is still much smoother than the corresponding IIDT(1) graph.

4.5 Comparison with Modern Decision Tree Learners

During the last decade, several modern decision tree learners were introduced. Although these learners were not presented and studied as anytime algorithms, some of them can be viewed as such. In what follows we compare our proposed anytime framework to three such algorithms: bagging, skewing and GATree. We first give a brief overview of the studied methods and then compare their performance to that of our anytime approach.

4.5.1 OVERVIEW OF THE COMPARED METHODS

Page and Ray (2003) introduced *skewing* as an alternative to lookahead for addressing problematic concepts such as parity functions. At each node, the algorithm skews the set of examples and produces several versions of it, each with different weights for the instances. The algorithm chooses to split on the attribute that exceeds a pre-set gain threshold for the greatest number of weightings. Skewing was reported to perform well on hard concepts such as XOR-*n*, mainly when the data set is large enough relative to the number of attributes. Skewing can be viewed as a contract algorithm parameterized by *w*, the number of weightings. In order to convert skewing into an interruptible algorithm, we apply the general conversion method described in Section 3.1.

Two improvements to the original skewing algorithm were presented, namely *sequential skewing* (Page and Ray, 2004), which skews one variable at a time instead of all of them simultaneously, and *generalized skewing* (Ray and Page, 2005), which can handle nominal and continuous attributes. Nominal attributes are skewed by randomly reordering the possible values and assigning a weight for a value proportional to its rank. Continuous attributes are handled by altering the input distribution for every possible split point. We test the sequential skewing algorithm on the binary *XOR* and *Multiplexer* data sets. This version is not parameterized and hence is not anytime by nature. To convert it into an anytime algorithm, we added a parameter *k* that controls the number of skewing iterations. Thus, instead of skewing each variable once, we skew it *k* times. For the *Tic-tac-toe* data set, where the attributes are ternary, we used only the generalized skewing algorithm, parameterized by the number of random orderings by which the nominal attributes are reweighed.

Papagelis and Kalles (2001) presented *GATree*, a learner that uses genetic algorithms to evolve decision trees. The initial population consists of randomly generated 1-level depth decision trees, where both the test and the class labels are drawn randomly. Mutation steps choose a random node and replaces its test with a new random one. If the chosen node is a leaf, the class label is replaced by a random label. Crossover chooses two random nodes, possibly from different trees, and swaps their subtrees. When tested on several UCI data sets, GATree was reported to produce trees as accurate as C4.5 but of significantly smaller size. GATree was also shown to outperform C4.5 on the *XOR-2* and *XOR-3* problems. GATree can be viewed as an interruptible anytime algorithm that uses additional time to produce more and more generations. In our experiments we used the GATree full original version with the same set of parameters as reported by Papagelis and Kalles, with one exception: we allowed a larger number of generations.

The third algorithm we tested is *bagging* (Breiman, 1996). Bagging is an ensemble-based method, and as such, it is naturally an interruptible anytime learner. Additional resources can be exploited by bagging to generate larger committees. In our experiments we consider 3 different bagging methods that use ID3, C4.5, and RTG (Random Tree Generator) as base learners. In addition, we tested a committee of trees produced by our LSID3. Since the latter takes significantly more time to run, the LSID3 committees are expected to be smaller than the greedy committees for

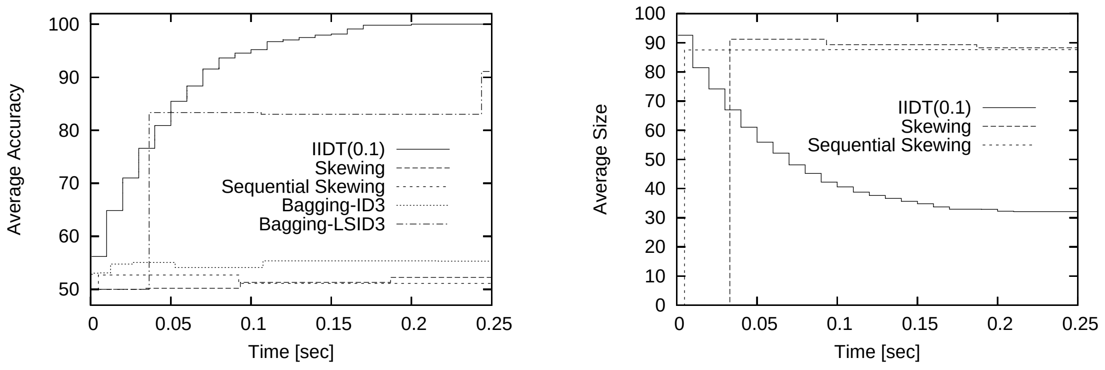


Figure 27: Anytime behavior of various modern algorithms on the *XOR-5* data set

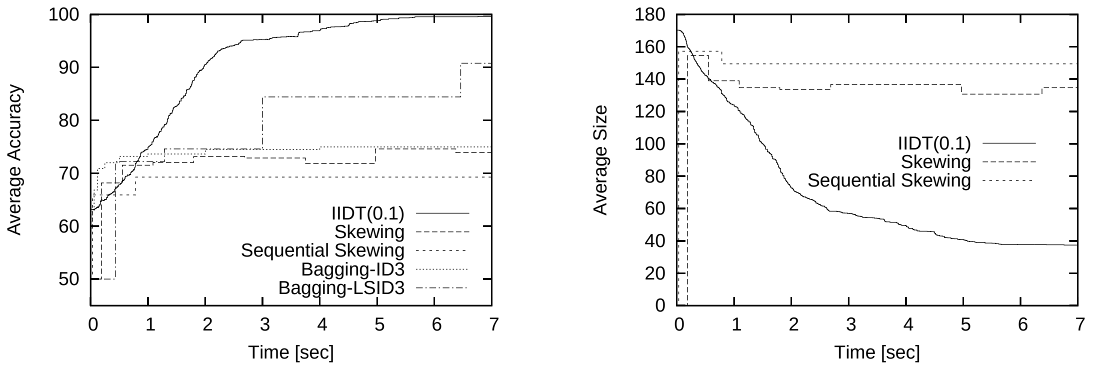


Figure 28: Anytime behavior of various modern algorithms on the *Multiplexer-20* data set

the same allocated time. Note that bagging is not a direct competitor to our method. We defined our goal as inducing a single "good" decision tree while bagging generates a set of trees. Generating a set of trees rather than a single good tree eliminates one of the greatest advantages of decision trees—their comprehensibility.

4.5.2 EMPIRICAL COMPARISON

We used our own implementation for IIDT, skewing, and bagging, and the commercial version for GATree.[17] The skewing and sequential skewing versions were run with linearly increasing parameters. The generalized skewing algorithm was run with exponentially increasing parameters. The performance of the ensemble method was tested for exponentially increasing committee sizes $(1, 2, 4, 8, \ldots)$.

Figures 27, 28, and 29 compare IIDT to bagging with ID3 as a base learner, bagging with LSID3($r = 1$), and skewing on the *XOR-5*, *Multiplexer-20,* and *Tic-tac-toe* tasks respectively. Note that the results for ID3 are identical to those of bagging-ID3 with a single tree in the committee and

17. The experiments with GATree were run on a Pentium IV 2.8 GHz machine with the Windows XP operating system. The reported times are as output by the application itself.

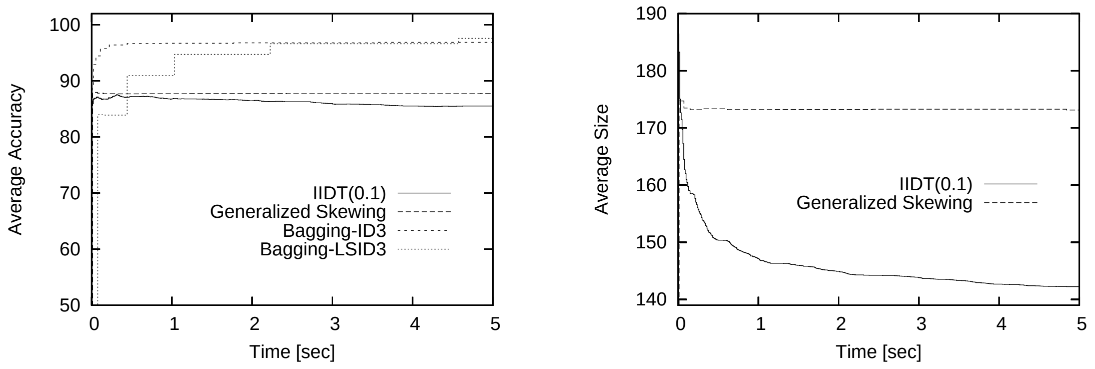


Figure 29: Anytime behavior of various modern algorithms on the *Tic-tac-toe* data set

hence are not plotted independently. Since GATree was run on a different machine, we report its results separately later in this section.

The graphs for the first 2 problems, which are known to be hard, show that IIDT clearly outperforms the other methods both in terms of tree size and accuracy. In both cases IIDT reaches almost perfect accuracy (99%), while bagging-ID3 and skewing topped at 55% for the first problem and 75% for the second.

The inferior performance of bagging-ID3 on the *XOR-5* and *Multiplexer-20* tasks is not surprising. The trees that form the committee were induced greedily and hence could not discover these difficult concepts, even when they were combined. Similar results were obtained when running bagging over C4.5 and RTG. However, when our LSID3($r = 1$) was used as a base learner, performance was significantly better than that of the greedy committees. Still, IIDT performed significantly better than bagging-LSID3, indicating that for difficult concepts, it is better to invest more resources for improving a single tree than for adding more trees of lower quality to the committee.

The inferior results of the skewing algorithms are more difficult to interpret, since skewing was shown to handle difficult concepts well. One possible explanation for this is the small number of examples with respect to the difficulty of the problem. To verify that this indeed explains the inferior results, we repeated the experiment with simpler XOR problems such as XOR-2 and XOR-3. In these cases skewing indeed did much better and outperformed ID3, reaching 100% accuracy (as IIDT). When we increased the size of the training set for the XOR-5 domain, skewing also performed better, yet IIDT outperformed it by more than 9%. For a deeper analysis of the difference between IIDT and skewing, see Section 5.

The average accuracy of GATree, after 150 generations, was 49.5%. It took more than 3 seconds on average to reach 150 generations. Thus, even when GATree was allocated much more time than IIDT, it could not compete with the latter. We repeated the experiment, allowing GATree to have a larger initial population and to produce more generations. The accuracy on the testing set, even after thousands of generations, remained very low. Similar results were obtained for the *Multiplexers-20* data set.

The above set of experiments was repeated on the much more difficult *XOR-10* data set. The advantage of IIDT over the other methods was even more evident. While IIDT was able to reach accuracy of 100%, bagging-ID3, skewing, and GATree performed as poorly as a random guesser, with accuracy of only 50%.

The next experiment was with the *Tic-tac-toe* data set. In this case, as shown in Figure 29, both ensemble-based methods have a significant advantage over the single tree inducers. We speculate that this is because ensemble methods were able to overcome the quick-fragmentation problem associated with multiway splits by combining several classifiers. We are still looking for ways to verify this hypothesis. Bagging-ID3 outperforms the other methods until the fifth second, where bagging-LSID3 overtakes it slightly. In contrast to the *XOR-5* domain, building larger committees is worthwhile in this case, even at the expense of less accurate base classifiers. However, if the time allocation permits, large ensembles of LSID3 trees are shown to be the most accurate. We believe that the general question of tradeoff between the resources allocated for each tree and the number of trees forming the ensemble should be addressed by further research with extensive experiments on various data sets. The performance of generalized skewing and IIDT was similar in this case, with a slight advantage for skewing in terms of accuracy and an advantage for IIDT in terms of tree size. GATree was run on the data set for 150 generations (30 seconds). The average accuracy was 76.42%, much lower than that of the other methods.

5. Related Work

While to the best of our knowledge no other work has tried to design an anytime algorithm for decision tree induction in particular, there are several related works that warrant discussion here. We consider first works that deal with decision tree inducers. We then discuss algorithms for induction of other models, and finally we consider methods that focus on aspects of the learning process other than the model-induction phase.

5.1 Single Tree Inducers

The goal of our research was to develop anytime algorithms for inducing a single decision tree. Several other algorithms for single decision-tree induction can either be considered anytime algorithms or can be converted into them with relative ease. We view the recently introduced skewing approach as the most relevant to our work. Therefore, we focus first on the differences between skewing and our anytime framework, and then we consider other decision tree learners.

5.1.1 SKEWING

The skewing approach was presented as an efficient alternative to lookahead (Page and Ray, 2003). In our discussion, we analyze the original skewing algorithm, which is applicable only to binary attributes. This analysis, however, holds also for the other versions of skewing (see Section 4.5), since they are based on the same ideas.

Skewing relies on the hypothesis that, when learning a hard concept, it may be easier for a greedy decision tree inducer to pick a relevant attribute if the distribution over the data is significantly different from the uniform distribution. Therefore, at each node, skewing repeatedly attempts to produce, from the existing set of examples, skewed versions that represent different distributions. Several different weight vectors are repeatedly assigned to the examples for this purpose. Then, the information gain of each attribute is calculated on the basis of each weight vector. The attribute that exceeds a pre-determined gain threshold for the greatest number of times is chosen. In order to generate the weight vectors, a *favored value* v_i is randomly drawn for each candidate attribute a_i. Then, the weight of each example in which a_i takes the value v_i is increased.

This process can be viewed as a stochastic clustering of the examples according to the values they take for a subset of the attributes: examples that most agree with the favored value for a large number of attributes are assigned the highest weight. Therefore, the calculation of the information gain will be most affected by sets of examples that share the same values for different subsets of the attributes.

In regular k-steps lookahead, for each tuple of k attributes we divide the set of examples into 2^k subsets, each of which is associated with a different k-tuple of binary values, that is, a different sub-path of length k in the lookahead tree. Then, we calculate the information gain for each subset and compute the average gain weighted by subset sizes.

In skewing, each iteration assigns a high weight to a few subsets of the examples that have common values for some of the attributes. These subsets have the greatest influence on the gain calculation at that skewing iteration. Hence, skewing iterations can be seen as sampling the space of lookahead sub-paths and considering a few of them at a time.

The great advantage of skewing over our proposed framework is efficiency. While LSID3 samples full decision trees, skewing samples only single sub-paths. When there are many examples but relatively few attributes, skewing improves greatly over the greedy learners at a low cost. The effectiveness of skewing, however, noticeably degrades when the concept hides a mutual dependency between a large number of attributes. When the number of attributes increases, it becomes harder to create large clusters with common values for some of the relevant ones, and hence the resulting lookahead is shallower.

Consider, for example, the *n-XOR* problem with n additional irrelevant attributes. For $n = 2$, a reweighting that assigns a high weight to examples that agree only on 1 of the 2 relevant attributes results in a high gain for the second attribute because it decreases the entropy of the cluster to 0. Nevertheless, in order to have a positive gain for a relevant attribute when $n = 10$, we should cluster together examples that agree on 9 of the 10 relevant attributes. Obviously, the probability for a good cluster in the first case is high while in the second case it is almost 0. The experiments reported in Section 4.5 provide an empirical backup for this argument.

5.1.2 OTHER SINGLE TREE INDUCERS

Papagelis and Kalles (2001) studied GATree, a learner that uses genetic algorithms for building decision trees. GATree does not adopt the top-down scheme. Instead, it starts with a population of random trees and uses a mutation operation of randomly changing a splitting test and a crossover operation of exchanging subtrees. Unlike our approach, GATree is not designed to generate consistent decision trees and searches the space of all possible trees over a given set of attributes. Thus, it is not appropriate for applications where a consistent tree is required. Like most genetic algorithms, GATree requires cautious parameter tuning and its performance depends greatly on the chosen setting. Comparing GATree to our algorithm (see Section 4.5) shows that, especially for hard concepts, it is much better to invest the resources in careful tuning of a single tree than to perform genetic search over the large population of decision trees.

Utgoff et al. (1997) presented DMTI, an induction algorithm that chooses an attribute by building a single decision tree under each candidate attribute and evaluates it using various measures. Several possible tree measures were examined and the MDL (Minimum Description Length) measure performed best. DMTI is similar to LSID3($r = 1$) but, unlike LSID3, it can only use a fixed amount of additional resources and hence cannot serve as an anytime algorithm. When the user can afford using more resources than required by DMTI, the latter does not provide means to improve the learned model further. Furthermore, DMTI uses a single greedy lookahead tree for attribute evaluation, while we use a biased sample of the possible lookahead trees. Our experiments with DMTI (as available online) show that while it can solve simpler XOR and multiplexer problems, its limited lookahead is not sufficient for learning complex concepts such as XOR-10: DMTI achieved an accuracy of 50%. IIDT and LSID3, by producing larger samples, overcame this problem and reached high accuracies.

Kim and Loh (2001) introduced CRUISE, a bias-free decision tree learner that attempts to produce more compact trees by (1) using multiway splits—one subnode for each class, and (2) examining pair-wise interactions among the variables. CRUISE is able to learn XOR-2 and Chess-board (numeric XOR-2) concepts. Much like ID3-k with $k = 2$, it cannot recognize more complex interactions.

Bennett (1994) presented GTO, a non-greedy approach for repairing multivariate decision trees. GTO requires as input an initial tree. The algorithm retains the structure of the tree but attempts to simultaneously improve all the multivariate decisions of the tree using iterative linear programming. GTO and IIDT both use a non-greedy approach to improve a decision tree. The advantage of GTO is its use of a well-established numerical method for optimization. Its disadvantages are its inability to modify the initial structure and its inability to exploit additional resources (beyond those needed for convergence).

5.2 Induction of Other Models

Ensemble-based methods, such as bagging and boosting (Schapire, 1999), can also be viewed as anytime algorithms. However, the classifiers constructed by the bagging and boosting algorithms consist of a committee of decision trees rather than a single tree. Therefore, the problem they face is very different from the problem we face in this work—that of learning a single tree. A major problem with ensemble-based methods is that the induced ensemble is often large, complex, and difficult to interpret (Freund and Mason, 1999). Therefore, these methods cannot be used when comprehensible models are required. Another problem is that greedy trees are unable to discover any knowledge about hard-to-learn target concepts. Therefore, combining them cannot improve performance. In our experiments, reported in Section 4.5, we provide empirical support for this claim by comparing our proposed anytime algorithms to bagging.

Dietterich (2000) presented the randomized-C4.5 algorithm, where a randomized version of C4.5 that chooses the split attribute at random from among the 20 best candidates is repeatedly invoked to produce an ensemble of decision trees. The experimental results indicate that ensembles of randomized-C4.5 were competitive with bagging but not as accurate as boosting. As in the case of the traditional bagging and boosting methods, our framework differs from randomized-C4.5 in that the latter produces an ensemble of trees that is obviously not as comprehensible as a single decision trees is.

The SVM algorithm usually depends on several parameters—kernel parameters for example. Several works, such as Chapelle et al. (2002), proposed iterative methods for automatic tuning of SVM parameters. These iterative methods can exploit additional time resources for better tuning.

Opitz (1995) introduced an anytime approach for theory refinement. This approach starts by generating a neural network from a set of rules that describe what is currently known about the domain. The network then uses the training data and the additional time resources to try to improve the resulting hypothesis.

5.3 Other Learning Components

In addition to the induction procedure, the learning process involves other components, where additional resources can be invested. Many algorithms for active learning (Lindenbaum et al., 2004), feature generation (Markovitch and Rosenstein, 2002), and feature selection (Last et al., 2001) have some level of anytime characteristics. All these methods are orthogonal to our induction methods and can complement them. An interesting research direction is to determine resource distribution between the various learning stages.

The related problem of *budgeted learning* was defined by Lizotte et al. (2003). There is a cost associated with obtaining each attribute value of a training example, and the task is to determine what attributes to test given a budget.

6. Conclusions

With the increased popularity of Machine Learning techniques, induction algorithms are being applied to more complex problems. Recently, much research effort has been spent on developing advanced induction algorithms, such as SVM. However, these algorithms do not provide comprehensible models and therefore are not appropriate when understandability is crucial, such as in medical tasks. Decision trees, on the contrary, are considered easy to interpret. In order to allow induction of better decision trees for hard-to-learn concepts, we presented a framework for anytime induction of decision trees that makes use of additional resources for producing better hypotheses.

Existing greedy algorithms for learning decision trees require a fixed small amount of time. We showed that for several real-world and synthetic data sets, our proposed anytime algorithms can exploit larger time budgets. This is shown to be worthwhile especially for hard concepts where existing greedy methods fail.

The major contributions of this paper are:

- *A better understanding of the space of consistent decision trees:* Using sampling techniques, we conducted experiments that support the application of Occam's Razor for decision trees and showed that smaller trees are clearly preferable.

- *LSID3:* We presented and empirically evaluated LSID3, a contract anytime algorithm for learning decision trees. On the data sets studied in this paper, LSID3 was shown to exhibit good anytime behavior and to produce better decision trees.

- *IIDT:* Motivated by the need for interruptible learners, we introduced IIDT, an interruptible anytime algorithm for inducing decision trees. In our experiments on hard concepts, IIDT produced significantly better decision trees when run for longer time.

- *Comparison with modern learners:* Our proposed framework was compared to several modern decision tree learners and shown to outperform them significantly. The advantage of LSID3 and IIDT is most evident when applied on problems with a strong interdependency.

When one can afford allocating extra resources, both LSID3 and IIDT are considered among the best choices for inducing a decision tree.

To the best of our knowledge, this is the first work that proposes anytime algorithms for learning decision trees. This research is only a first step in this direction. We are currently in the process of designing and implementing many variations of our proposed algorithms, including algorithms that combine ID3-k and LSID3, algorithms that allow different resource distribution for different sub-concepts, and algorithms that allow pruning as an improvement phase in the interruptible learner. In addition, we intend to adapt our anytime framework for incremental tasks, where new examples arrive after the learning process has been initiated. This will allow the classifier to exploit both additional time resources and additional data that becomes available with time.

Acknowledgments

This work was partially supported by funding from the EC-sponsored MUSCLE Network of Excellence (FP6-507752).

Appendix A. Data Sets

Below we give a more detailed description of the data sets used in our experiments:

1. *Automobile:* This problem, taken from the UCI Repository, consists of three types of entities: (1) the specification of an automobile, (2) its assigned insurance risk rating, and (3) its normalized losses in use as compared to other cars. Several of the attributes in the database could be used as class attributes: we chose to use both the make (model) and the symbolling (risk degree).

2. *Balance Scale:* This data set is taken from the UCI Repository and was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight and the right distance.

3. *Breast Cancer (Ljubljana):* This problem is taken from the UCI Repository. Each instance represents the characteristics of a patient and the class is whether or not there are recurrence events.

4. *Connect-4:* This data set, taken from the UCI Repository, contains all legal 8-ply positions in the connect-4 game in which neither player has won yet and the next move is not forced.

5. *Corral:* An artificial data set first used by John et al. (1994).

6. *Glass:* In this domain, taken from the UCI Repository, the goal is to determine the type of glass from its characteristics.

7. *Iris:* This data set is taken from the UCI Repository and contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

8. *Monks problems:* This set, taken from the UCI Repository, contains three problems. Each example is represented by 5 nominal attributes in the range $1,2,3,4$. The problems are:

   - *Monks-1:* $(a1 = a2) or (a5 = 1)$.
   - *Monks-2:* exactly two of $(a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1)$.
   - *Monks-3:* $((a5 = 3) and (a4 = 1)) or ((a5 \neq 4) and (a2 \neq 3))$, with an added 5% class noise.

   The original data sets are already partitioned into training and testing sets.

9. *Mushroom:* This data set, taken from the UCI Repository, includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family. Each species is identified as edible or poisonous.

10. *Solar Flare:* This problem is taken from the UCI Repository. Each instance represents captured features for one active region on the sun. Among the three possible classes, we considered the C-class flare where the instances are more distributed.

11. *Tic-Tac-Toe:* The problem, taken from the UCI Repository, deals with the classification of legal tic-tac-toe end games, as wins or non-wins for the x player. Each example is represented by 9 nominal attributes, which represent the slot values.

12. *Voting:* This data set, taken from the UCI repository, includes votes for each member of the U.S. House of Representatives on the 16 key votes identified by the CQA. The class of each record is Democrat or Republican.

13. *Wine:* This problem, taken from the UCI Repository, deals with the classification of wines into 3 class types. Each example is represented by 13 continuous attributes, which represent measures of chemical elements in the wine.

14. *Zoo:* In this domain, taken from the UCI Repository, the goal is to determine the type of the animal from several attributes.

15. *Multiplexer:* The multiplexer task was used by several researchers for evaluating classifiers, for example, Quinlan (1993). An instance is a series of bits of length $a + 2^a$, where a is a positive integer. The first a bits represent an index into the remaining bits and the label of the instance is the value of the indexed bit. In our experiments we considered the 20-Multiplexer ($a = 4$). The data set contains 500 randomly drawn instances.

16. *Boolean XOR:* Parity-like functions are known to be problematic for many learning algorithms. However, they naturally arise in real-world data, such as the Drosophila survival concept (Page and Ray, 2003). We considered XOR of five and ten variables with additional irrelevant attributes.

17. *Numeric XOR:* A XOR based numeric data set that has been used to evaluate learning algorithms, for example, Baram et al. (2003). Each example consists of values for x and y coordinates. The example is labelled 1 if the product of x and y is positive, and -1 otherwise. We generalized this domain for three and four dimensions and added irrelevant variables to make the concept harder.

18. *Multiplex-XOR:* Another XOR based concept that is defined over 11 binary attributes. The concept is a composition of two XOR terms, where the first attribute determines which one should be considered. The other 10 attributes are used to form the XOR terms. The size of each term is drawn randomly between 2 and 5.

Appendix B. Full Results

Table 2 shows the size of the trees induced by the above mentioned algorithms, lists the differences between the algorithms, and states whether these differences were found to have t-test significance with $\alpha = 0.05$. Similarly, Table 3 compares the produced trees in terms of their generalization accuracy.

DATA SET	ID3	C4.5	ID3-K ($k=2$)	LSID3 ($r=5$)	LSID3-P ($r=5$)	LSID3 VS. ID3 DIFF	SIG?	LSID3 VS. C4.5 DIFF	SIG?	LSID3-P VS. C4.5 DIFF	SIG?
AUTOS MAKE	53.6 ±4.2	26.6 ±1.4	56.5 ±2.4	36.6 ±2.0	37.1 ±2.0	-17.1 ±4.8	√	9.9 ±2.4	×	10.4 ±2.2	×
AUTOS SYM.	46.7 ±1.8	20.1 ±4.1	47.0 ±1.8	26.8 ±1.7	26.5 ±2.1	-19.9 ±2.2	√	6.6 ±4.1	×	6.3 ±4.3	×
BALANCE	353.9 ±6.9	34.8 ±5.5	351.6 ±7.3	347.7 ±7.7	40.1 ±6.5	-6.2 ±5.0	√	312.8 ±9.8	×	5.2 ±7.7	×
BR. CANCER	129.6 ±5.9	6.1 ±4.0	124.0 ±4.9	100.3 ±4.4	7.7 ±7.1	-29.3 ±5.6	√	94.2 ±5.4	×	1.7 ±8.5	∼
CONNECT-4	18507 ±139	3329 ±64	16143 ±44	14531 ±168	6614 ±173	-3976 ±265	√	11201 ±183	×	3284 ±186	×
CORRAL	9.2 ±2.1	5.3 ±1.2	7.0 ±0.6	6.7 ±0.5	6.3 ±0.8	-2.5 ±2.2	√	1.4 ±1.3	×	1.0 ±1.3	×
GLASS	38.7 ±2.3	23.9 ±2.6	32.3 ±2.3	34.2 ±2.1	35.5 ±2.4	-4.4 ±3.0	√	10.3 ±3.1	×	11.6 ±3.7	×
IRIS	8.5 ±1.0	4.7 ±0.5	9.1 ±0.9	7.6 ±0.8	6.6 ±1.6	-0.9 ±0.7	√	2.9 ±1.0	×	1.9 ±1.6	×
MONKS-1	62.0 ±0.0	11.0 ±0.0	27.0 ±0.0	27.0 ±0.0	21.0 ±0.0	-35.0 ±0.0	-	16.0 ±0.0	-	10.0 ±0.0	-
MONKS-2	109.0 ±0.0	20.0 ±0.0	105.0 ±0.0	92.4 ±3.4	18.3 ±5.0	-16.6 ±3.4	-	72.4 ±3.4	-	-1.7 ±5.0	-
MONKS-3	31.0 ±0.0	9.0 ±0.0	34.0 ±0.0	26.8 ±1.6	9.9 ±1.4	-4.2 ±1.6	-	17.8 ±1.6	-	0.9 ±1.4	-
MUSHROOM	24.0 ±0.0	19.0 ±0.2	24.9 ±0.2	16.2 ±0.9	16.2 ±0.9	-7.8 ±0.9	√	-2.8 ±0.9	√	-2.9 ±0.9	√
SOLAR-FLARE	68.9 ±2.9	2.0 ±1.0	68.4 ±3.2	63.2 ±2.9	1.2 ±0.8	-5.6 ±2.5	√	61.2 ±3.1	×	-0.8 ±1.4	√
TIC-TAC-TOE	189.0 ±13.6	83.4 ±7.8	176.7 ±9.0	151.7 ±5.6	112.6 ±8.9	-37.3 ±15.1	√	68.3 ±9.3	×	29.1 ±10.9	×
VOTING	13.6 ±2.2	2.8 ±1.5	12.3 ±1.6	13.0 ±2.0	3.5 ±2.1	-0.6 ±2.1	√	10.2 ±2.5	×	0.7 ±2.4	×
WINE	7.9 ±1.0	5.3 ±0.7	7.3 ±0.9	6.2 ±0.7	7.4 ±1.3	-1.7 ±1.2	√	0.9 ±1.0	×	2.1 ±1.6	×
ZOO	13.8 ±0.4	8.3 ±0.8	13.8 ±0.5	9.9 ±0.8	9.8 ±0.9	-3.9 ±0.9	√	1.6 ±1.2	×	1.5 ±1.2	×
NUMERIC XOR-3D	43.0 ±5.1	1.0 ±0.1	15.6 ±6.8	9.2 ±1.0	11.7 ±1.7	-33.8 ±5.1	√	8.2 ±1.0	×	10.7 ±1.7	×
NUMERIC XOR-4D	104.7 ±4.5	2.7 ±1.8	75.5 ±14.0	26.8 ±4.7	30.9 ±5.9	-77.9 ±6.0	√	24.1 ±4.8	×	28.1 ±6.3	×
MULTIPLEXER-20	142.8 ±8.3	66.1 ±6.5	67.1 ±29.0	46.6 ±20.0	41.2 ±16.8	-96.1 ±21.6	√	-19.4 ±20.4	√	-24.9 ±17.4	√
MULTIPLEX-XOR	84.0 ±5.6	26.0 ±5.2	70.2 ±5.3	43.9 ±5.7	31.6 ±4.8	-40.1 ±7.3	√	17.9 ±8.1	×	5.7 ±7.0	×
XOR-5	92.3 ±7.8	21.9 ±5.3	82.1 ±9.3	32.0 ±0.0	32.0 ±0.0	-60.3 ±7.8	√	10.1 ±5.3	×	10.1 ±5.3	×
XOR-5 NOISE	93.6 ±6.0	23.2 ±5.9	82.4 ±7.3	58.2 ±6.1	40.4 ±5.0	-35.4 ±8.3	√	35.0 ±8.9	×	17.2 ±8.1	×
XOR-10	3901 ±34	1367 ±39	3287 ±37	2004 ±585	1641 ±524	-1897 ±587	√	637 ±577	×	273 ±524	×

Table 2: The size of the induced trees on various data sets. The numbers represent the average and standard deviation over the individual runs. The last columns list the average differences between the algorithms and their t-test significance, with $\alpha = 0.05$ ($\surd$ indicates a significant advantage and $\times$ a significant disadvantage). The t-test is not applicable for the Monk data sets because only 1 train-test partition was used.

DATA SET	ID3	C4.5	ID3-K ($k = 2$)	LSID3 ($r = 5$)	LSID3-P ($r = 5$)	LSID3 VS. ID3 DIFF	SIG?	LSID3 VS. C4.5 DIFF	SIG?	LSID3-P VS. C4.5 DIFF	SIG?
AUTOS MAKE	79.1 ±9.8	79.4 ±10.5	78.0 ±9.7	80.3 ±9.9	79.2 ±10.2	1.2 ±9.9	∼	0.8 ±10.3	∼	-0.2 ±10.6	∼
AUTOS SYM.	81.9 ±9.8	77.4 ±10.6	81.2 ±9.5	81.3 ±10.2	81.1 ±9.5	-0.6 ±11.4	∼	3.9 ±12.3	√	3.7 ±12.2	√
BALANCE	68.8 ±5.4	63.8 ±4.9	69.7 ±5.0	70.3 ±5.7	67.9 ±5.5	1.5 ±4.5	√	6.5 ±5.6	√	4.1 ±5.5	√
BR. CANCER	67.0 ±8.8	74.7 ±8.1	64.4 ±8.4	66.9 ±9.1	71.1 ±8.2	-0.2 ±9.1	∼	-7.8 ±9.1	×	-3.6 ±6.0	×
CONNECT-4	75.6 ±0.5	81.3 ±0.4	77.9 ±0.5	78.6 ±0.5	80.4 ±0.5	3.0 ±0.6	√	-2.7 ±0.5	×	-0.9 ±0.4	×
CORRAL	69.7 ±25.4	65.3 ±19.8	89.0 ±19.9	83.5 ±20.9	79.6 ±19.8	13.8 ±25.4	√	18.2 ±25.1	√	14.2 ±24.1	√
GLASS	66.1 ±10.2	67.4 ±10.3	70.6 ±11.0	70.5 ±10.7	69.5 ±10.4	4.3 ±11.6	√	3.0 ±12.7	√	2.0 ±11.5	∼
IRIS	93.3 ±5.9	95.3 ±4.8	93.1 ±6.7	94.5 ±6.0	94.3 ±6.2	1.1 ±4.0	√	-0.8 ±7.8	∼	-0.9 ±7.9	∼
MONKS-1	82.9 ±0.0	75.7 ±0.0	100.0 ±0.0	100.0 ±0.0	94.4 ±0.0	17.1 ±0.0	-	24.3 ±0.0	-	18.8 ±0.0	-
MONKS-2	69.2 ±0.0	65.0 ±0.0	63.0 ±0.0	67.1 ±0.5	63.6 ±1.1	-2.2 ±0.5	-	2.0 ±0.5	-	-1.5 ±1.1	-
MONKS-3	94.4 ±0.0	97.2 ±0.0	91.7 ±0.0	91.5 ±1.5	98.1 ±1.3	-3.0 ±1.5	-	-5.7 ±1.5	-	0.8 ±1.3	-
MUSHROOM	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0	0.0 ±0.0	∼	0.0 ±0.0	∼	0.0 ±0.0	∼
SOLAR FLARE	86.6 ±6.1	88.9 ±5.2	86.6 ±6.0	86.5 ±6.4	88.9 ±5.1	-0.2 ±3.0	∼	-2.4 ±8.0	×	0.0 ±7.6	∼
TIC-TAC-TOE	85.5 ±3.8	85.8 ±3.3	84.2 ±3.4	87.0 ±3.2	87.2 ±3.1	1.5 ±4.5	√	1.2 ±4.8	√	1.4 ±4.6	√
VOTING	95.1 ±4.1	96.4 ±3.8	95.9 ±4.5	95.6 ±4.9	96.5 ±3.5	0.4 ±4.6	∼	-0.9 ±4.4	∼	0.1 ±1.8	∼
WINE	92.6 ±6.9	92.9 ±6.9	91.4 ±6.3	92.5 ±6.2	92.3 ±6.2	-0.1 ±7.4	∼	-0.4 ±8.7	∼	-0.6 ±10.0	∼
ZOO	95.2 ±7.7	92.2 ±8.0	94.3 ±7.9	94.3 ±6.6	93.5 ±6.9	-1.0 ±8.0	∼	2.1 ±10.7	∼	1.3 ±10.1	∼
NUMERIC XOR-3D	57.7 ±11.1	43.0 ±5.4	89.5 ±11.7	96.1 ±4.3	93.4 ±5.5	38.4 ±12.0	√	53.1 ±7.1	√	50.4 ±7.6	√
NUMERIC XOR-4D	49.8 ±7.0	52.1 ±4.7	62.3 ±11.8	93.2 ±4.4	91.9 ±4.7	43.4 ±7.8	√	41.1 ±6.2	√	39.8 ±5.9	√
MULTIPLEXER-20	61.3 ±7.2	62.1 ±7.5	87.2 ±14.2	95.5 ±8.5	94.5 ±9.5	34.1 ±11.8	√	33.4 ±11.0	√	32.3 ±12.1	√
MULTIPLEX-XOR	58.2 ±12.0	55.4 ±11.5	61.2 ±12.2	76.5 ±11.8	80.1 ±9.6	18.3 ±16.2	√	21.1 ±15.1	√	24.7 ±14.7	√
XOR-5	55.5 ±12.2	54.1 ±12.8	55.8 ±13.2	100.0 ±0.0	100.0 ±0.0	44.5 ±12.2	√	45.9 ±12.8	√	45.9 ±12.8	√
XOR-5 NOISE	54.1 ±12.5	51.7 ±11.9	56.6 ±12.6	74.4 ±11.4	78.2 ±14.0	20.3 ±17.2	√	22.7 ±15.9	√	26.5 ±18.2	√
XOR-10	49.9 ±1.8	49.8 ±1.8	50.3 ±1.6	77.4 ±14.9	79.4 ±16.5	27.5 ±14.7	√	27.7 ±14.8	√	29.6 ±16.7	√

Table 3: The generalization accuracy of the induced trees on various data sets. The numbers represent the average and standard deviation over the individual runs. The last columns list the average differences between the algorithms and their t-test significance, with $\alpha = 0.05$ ($\surd$ indicates a significant advantage and $\times$ a significant disadvantage). The t-test is not applicable for the Monk data sets because only 1 train-test partition was used.

References

Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 19–26, Washington, DC, USA, 2003.

K. Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, 26:156–160, 1994.

C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/∼mlearn/MLRepository.html`.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's Razor. *Information Processing Letters*, 24(6):377–380, 1987.

M. Boddy and T. L. Dean. Deliberation scheduling for problem solving in time constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 51–58, Washington, DC, USA, 2003.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, 1992.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

M. W. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Department of Computer Sciences, University of Wisconsin, Madison, 1996.

J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

M. Dong and R. Kothari. Look-ahead based fuzzy decision tree induction. *IEEE-FS*, 9:461–468, June 2001.

J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202, Tahoe City, California, USA, 1995.

S. Esmeir and S. Markovitch. Occam's Razor just got sharper. In *Proceedings of The Twentieth International Joint Conference on Artificial Intelligence*, India, 2007.

F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

U. M. Fayyad and K. B. Irani. What should be minimized in a decision tree? In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 749–754, Boston, Massachusetts, USA, 1990. AAAI Press / The MIT Press.

Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 124–133, Bled, Slovenia, 1999.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag, 2001.

E. Hovitz. *Computation and Action under Bounded Resources*. PhD thesis, Computer Science Department, Stanford University, 1990.

L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121–129, 1994.

D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

H. Kim and W. Loh. Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, 96:589–604, 2001.

I. Kononenko, E. Simec, and M. Robnik-Sikonja. Overcoming the myopia of inductive learning algorithms with RELIEFF. *Applied Intelligence*, 7(1):39–55, 1997.

M. Last, A. Kandel, O. Maimon, and E. Eberbach. Anytime algorithm for feature selection. In *Revised Papers from the Second International Conference on Rough Sets and Current Trends in Computing*, pages 532–539. Springer-Verlag, 2001.

M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine Learning*, 54(2):125–152, 2004.

D. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive bayes classifiers. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, Acapulco, Mexico, 2003.

W. Loh and Y. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

S. Markovitch and D. Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49:59–98, 2002.

J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learing*, 3(4):319–342, 1989.

S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

O. J. Murphy and R. L. McCraw. Designing storage efficient decision trees. *IEEE Transactions on Computers*, 40(3):315–320, 1991.

P. M. Murphy and M. J. Pazzani. Exploring the Decision Forest: An empirical investigation of Occam's Razor in decision tree induction. *Journal of Artificial Intelligence Research*, 1:257–275, 1994.

S. K. Murthy and S. Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1025–1033, Montreal, Canada, 1995.

S. W. Norton. Generating better decision trees. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 800–805, Detroit, Michigan, USA, 1989.

T. Oates and D. Jensen. The effects of training set size on decision tree complexity. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 254–262. Morgan Kaufmann, 1997.

D. Opitz. *An Anytime Approach to Connectionist Theory Refinement: Refining the Topologies of Knowledge-Based Neural Networks*. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, 1995.

D. Page and S. Ray. Skewing: An efficient alternative to lookahead for decision tree induction. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

D. Page and S. Ray. Sequential Skewing: An improved skewing algorithm. In *Proceedings of the Twenty-First International Conference on Machine Learning*, Banff, Canada, 2004.

A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 393–400, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, CA, 1993.

J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1019–1024, 1995.

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80(3):227–248, 1989.

H. Ragavan and L. Rendell. Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 252–259, Amherst, MA, USA, 1993.

R. Rao, D. Gordon, and W. Spears. For every generalization action, is there really an equal or opposite reaction? In *Proceedings of Twelfth International Conference on Machine Learning*, pages 471–479, 1995.

S. Ray and D. Page. Generalized Skewing for functions with continuous and nominal attributes. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, Bonn, Germany, 2005.

S. J. Russell and E. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 400–411, San Mateo, California, 1989.

S. J. Russell and S. Zilberstein. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.

U. K. Sarkar, P. Chakrabarti, S. Ghose, and S. C. DeSarkar. Improving greedy algorithms by lookahead search. *Journal of Algorithms*, 16:1–23, 1994.

A. Schaerf. Tabu search techniques for large high-school timetabling problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 363–368, OR, USA, 1996.

C. Schaffer. A conservation law for generalization performance. In *Proceedings of Eleventh International Conference on Machine Learning*, pages 259–265, 1994.

R. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406, Stockholm, Sweden, 1999.

Jude W. Shavlik, Raymond J. Mooney, and Geoffrey Towell G. Symbolic and neural learning algorithm: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

V. Vapnik. *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

G. I. Webb. Further experimental evidence against the utility of Occam's Razor. *Journal of Artificial Intelligence Research*, 4:397–417, 1996.

I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, CA, 2005.