Lookahead-based Algorithms for Anytime Induction 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

Abstract

The majority of the 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. Furthermore, the greedy algorithms require a fixed amount of time and are not able to generate a better tree if additional time is available. To overcome this problem, we present two lookahead-based algorithms for anytime induction of decision trees, thus allowing tradeoff between tree quality and learning time. The first one is depth-k lookahead, where a larger time allocation permits larger k. The second algorithm uses a novel strategy for evaluating candidate splits; a stochastic version of ID3 is repeatedly invoked to estimate the size of the tree in which each split results, and the one that minimizes the expected size is preferred. 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.

1. Introduction

Assume that a medical center has decided to acquire a classification system for cancer diagnosis. Few seconds after supplying the system with thousands of previous cases, a decision tree is produced. 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 for obtaining the most accurate decision tree available. However, most of the existing algorithms for induction of decision trees do not allow such a tradeoff: they cannot make use of additional time in order to generate better decision trees.

Despite the recent progress in developing advanced induction algorithms, such as SVM (Burges, 1998), *decision trees* are still considered attractive for many real-life applications where classifier comprehensibility is important. The majority of existing methods for decision tree induction, such as CART (Breiman et al., 1984), ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993), use local heuristics in attempt to produce small trees, which, by the Occam's Razor principle (Blumer et al., 1987), should have better predictive power. In most cases, these methods do not result in globally optimal trees. The possibility of constructing optimal trees has been theoretically examined by several researchers. Hyafil and Rivest (1976) showed that the problem of constructing from decision tables, decision trees that are optimal with respect to the expected classification cost, is NP-Complete. Murphy and McCraw (1991) showed that for most cases, the construction of a storage optimal decision tree is an NP-complete problem.

In many applications that deal with hard problems, we are ready to allocate much more resources than required by simple greedy algorithms, but still cannot afford using algorithms of exponential complexity. One of the commonly suggested approaches to handle such cases, is using anytime algorithms (Boddy & Dean, 1994) that can trade quality for time. Quinlan (1993) recognized the need for this type of 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...".

In this paper, we aim to fulfil this requirement by presenting lookahead-based anytime algorithms for constructing decision trees. Lookahead search is a well-known technique for improving greedy algo-

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004. Copyright 2004 by the authors.

rithms (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 decedents of the node. One of the main disadvantages of the greedy strategy, is the irreversibility of the damage caused by a wrong decision: once an attribute was chosen to split on, there is no provision for backtracking and choosing another attribute instead. Lookahead search attempts to predict and avoid such non-contributive splits during the process of induction, before the final decision at each node is taken. 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. The main challenge we face in this work is to make use of extra resources to induce better decision trees for such hard concepts.

In this work, we restrict the search space to trees *consistent* with the training set. Note that while pruning techniques (Breiman et al., 1984; Quinlan, 1993) also attempt to obtain small decision trees, their goals and their search space are different from ours. The main goal of pruning algorithms is to avoid overfitting the data, mainly in the presence of noise. The trees produced by pruning algorithms are not necessarily consistent and are restricted to those attainable from the "greedy" trees by a sequence of pruning and grafting operations. Therefore, in many cases, pruning is unable to recover from wrong decisions. In addition, the anytime behavior of pruning algorithms is limited: once all non-contributive subtrees are removed, the enhancing process is terminated and the constructed tree can no longer be improved.

2. Anytime Induction of Decision Trees

In this section we present two anytime algorithms for decision tree induction. There are two main classes of anytime algorithms namely, *contract* algorithms and *interruptible* algorithms (Russell & Zilberstein, 1991). A Contract algorithm is one that gets its resource allocation as a parameter. An Interruptible algorithm is one whose resource allocation is not given in advance and thus must be prepared to be interrupted and return a solution at any moment. In this work we are mainly interested in contract algorithms since we view their assumption as the most natural for induction tasks. Nevertheless, Russell and Zilberstein (1991) showed that any given contract algorithm can be converted to an interruptible one with a small constant penalty.

The two algorithms described in this section are based on top-down induction of decision trees (TDIDT). Under this framework, an attribute is chosen to partition the entire dataset into subsets, each of which is used to recursively build a subtree. Figure 1 lists the basic algorithm for TDIDT. The proposed algorithms differ in the way they select an attribute for splitting a node. The first, called ID3-k, generalizes ID3 by looking ahead to depth k. The second, our novel algorithm, uses a stochastic version of ID3 to perform lookahead.

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

2.1. ID3-k

Let E be a set of examples at a tree node t. Let $\mathrm{P}_E(c_i)$ be the probability of an example in E to belong to class c_i. We can calculate the entropy at t by using Shannon's information measure:

$$\text{entropy}(E) = \mathrm{I}(\mathrm{P}_E(c_1), \ldots, \mathrm{P}_E(c_n)).$$

When an attribute a with values $\{v_1, \ldots, v_m\}$ is used as the test attribute in a tree node, it partitions E into $\{E_1, \ldots, E_m\}$ where each E_i is a subset of E that contains only the examples for which the value of the attribute a is v_i. The new entropy is the weighted average of the entropies for each subset:

$$\text{entropy-1}(E, a) = \sum_{i=1}^{m} \frac{|E_i|}{|E|} \cdot \text{entropy}(E_i).$$

Thus, the expected information gain for a is given by:

$$\text{gain-1}(E, a) = \text{entropy}(E) - \text{entropy-1}(E, a).$$

The suffix "1"of entropy-1 indicates that the effect of using the attribute was tested one level below the current node. We can extend this definition to allow measuring the entropy at any depth k below the current node. The recursive definition minimizes the $k - 1$

```
Procedure ENTROPY-K(E, A, a, k)
  If k = 0
    Return I(P_E(c_1), ..., P_E(c_n))
  V ← domain(a)
  Foreach v_i ∈ V
    E_i ← {e ∈ E | a(e) = v_i}
    Foreach a' ∈ A
      A' ← A − {a'}
      h_i(a') ← ENTROPY-K(E_i, A', a', k − 1))
  Return Σ_{i=1}^{|V|} (|E_i|/|E|) min_{a'∈A} (h_i(a'))

Procedure GAIN-K(E, A, a, k)
  Return I(P_E(c_1), ..., P_E(c_n)) −
           ENTROPY-K(E, A, a, k)
```

Figure 2. Procedures for computing entropy-k and gain-k.

```
Procedure SID3-CHOOSE-ATTRIBUTE(E, A)
  Foreach a ∈ A
    p(a) ← gain-1(E, a)
  If ∃a such that entropy-1(E, a) = 0
    a* ← Choose attribute at random from
         {a ∈ A | entropy-1(E, a) = 0}
  Else
    a* ← Choose attribute at random from A;
         for each attribute a, the probability
         of selecting it is proportional to p(a)
  Return a*
```

Figure 3. Attribute selection in SID3.

entropy for each child and computes their weighted average. Figure 2 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 to split on the attribute that maximizes gain-k. In ID3, if two attributes yield the same decrease in entropy, we choose one randomly. In ID3-k we do the same only if both attributes were explored to the same depth. If, however, one of the two attributes has an associated shallower lookahead tree, we obviously prefer it over the other.

The contract time allocated to ID3-k is determined by the k parameter.[1] Despite its ability to exploit additional resources when available, the anytime behavior of ID3-k is problematic. First, the run time of ID3-k grows exponentially as k increases. As a result, the gap between the points of time at which the resulted tree can potentially improve gets wider, limiting the adjustability of the algorithm. Another problem with this approach is that it is quite possible that looking ahead to depth k is not sufficient to determine the usefulness of an attribute. To illustrate this problem, let us consider the n-XOR problem, where the concept is defined by $a_1(x) \oplus \ldots \oplus a_n(x)$. ID3-k with $k < n$ will not be able to realize the utility of the attributes $a_1, \ldots, a_n$. However, when looking ahead to depth n the optimal solution can be found.

2.2. Lookahead by Stochastic ID3

The goal of our induction process is to find a small tree consistent with the examples. ID3 uses information gain as a heuristic to estimate which attribute will yield a small tree. ID3-k extends this notion and tests the effect of a split further down the tree. Exhaustive lookahead will obviously lead to the smallest tree but its computational costs are prohibitively high. One way to reduce the costs is invoking shallower lookahead. Another way is performing a full lookahead but exploring only the most promising attribute at each level, i.e., using ID3 itself as the tree size estimator.

Given an attribute a, we partition the set of examples according to the different values a can take and call ID3 for each subset. Each such call results in a subtree. Summing up the size of each subtree gives an estimation of the total tree size in case a is chosen to split on. Since ID3 does not necessarily return the smallest tree, this heuristic might over-estimate but never under-estimates the optimal tree size. One problem with the above algorithm is that it is not a contract algorithm—it has a fixed execution time and cannot improve the solution quality using additional resources. To overcome this shortcoming and to produce a better estimate of the tree size, instead of calling ID3, we call *Stochastic-ID3* several times.

Stochastic-ID3 (SID3) is a TDIDT algorithm that we have designed to allow sampling the space of "good" trees produced by greedy algorithms. Instead of choosing the attribute that maximizes the information gain, the splitting attribute is drawn randomly with a likelihood that is proportional to the attribute's information gain.[2] However, if there are attributes that decrease the entropy to zero then one of them is picked randomly. The attribute selection procedure of the algorithm is listed in Figure 3. Since SID3 is not a deterministic algorithm, different runs of it might return different trees of different sizes.

[1] The mapping from time to the appropriate k can be estimated, for example, from experience.

[2] We make sure that attributes with gain of zero will have a positive probability to be selected.

Procedure LSID3-CHOOSE-ATTRIBUTE(E, A, r)
  If $r = 0$
    Return ID3-CHOOSE-ATTRIBUTE(E, A)
  Foreach $a \in A$
    Foreach $v_i \in \text{domain}(a)$
      $E_i \leftarrow \{e \in E \mid a(e) = v_i\}$
      $min_i \leftarrow \infty$
      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 4. Attribute selection in LSID3.

In *Lookahead-by-Stochastic-ID3* (LSID3) each candidate split is evaluated by summing up the estimated size of each subtree. For each subtree there are several estimations, and since each estimation is an upper bound, the minimal one is considered. LSID3 is a contract algorithm parameterized by r, the number of times SID3 is called for each candidate. We define LSID3 with $r = 0$ to be ID3. Figure 4 formalizes the choice of splitting attributes as made by LSID3.

The run-time of LSID3 grows linearly with r. We expect that increasing r will improve the classifier quality due to the better sampling. For example, consider the expected behavior of LSID3 on the concept $a_1(x) \oplus a_2(x) \oplus a_3(x)$. Let $A = \{a_1, a_2, a_3, a_4\}$ be the set of binary attributes and E the set of examples. Assume that entropy-1$(E, a) = 1$ for $a \neq a_4$ while entropy-1$(E, a_4) < 1$. 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 is selected. For a_4 this estimation must be 2^4 since all possible trees rooted at a_4 are of this size. The only way that a_4 will be mistakenly selected by LSID3 is that the estimation for each of the other 3 attributes will be also 2^4. This can happen only if SID3, when invoked to estimate the size of the trees rooted at $a \neq a_4$ will select a_4 for a's both decedents.[3] The probability of such event is low, and decreases with the increase in r. Therefore, calling LSID3 with higher values of r will increase the likelihood of finding the right tree.

3. Experiments

A variety of experiments were conducted to test the performance and behavior of both ID3-k and LSID3.

[3]It is easy to see that a_4 cannot be selected at lower levels.

The first set of experiments compares ID3, ID3-k with $k = 2$ and LSID3 with $r = 5$, in terms of generalization accuracy and size of the induced trees, measured by the number of leaves. We also show results for C4.5 with its default parameters except of using information gain rather than gain ratio. We use 12 well-known benchmark datasets from the UCI Machine Learning repository (Blake & Merz, 1998), additionally to the following relatively hard-concept datasets that were referred to in previous Machine Learning literature:

1. *Multiplexer:* The multiplexer task was used by several researchers for evaluating classifiers (e.g., 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 problem in which $a = 4$. The dataset contains 500 randomly drawn instances.

2. *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 & Ray, 2003). We used two XOR based datasets. The first is the 5-XOR problem with additional 5 irrelevant attributes. The second dataset, named Multiplex-XOR, is defined over 11 binary attributes. The concept is a composition of two XOR terms, where the first attribute determines which one of them 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. Both datasets contain 200 randomly drawn examples.

3. *Numeric XOR:* A XOR based numeric dataset that has been previously used to evaluate learning algorithms (e.g., 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 dimensions and added an irrelevant variable to make the concept harder. The dataset contains randomly drawn 200 examples.

4. *Shapes-Connectedness:* This dataset is based on the connectedness example Minsky and Papert (1969) gave to show the practical limitations of the perceptron. Four binary attributes represent the sides of the shape, to which we added three irrelevant attributes for obtaining a harder concept. The dataset contains all 2^7 possible assignments.

Table 1. The size of the induced trees for ID3, C4.5, ID3-k and LSID3 on various datasets. The numbers represent the average and standard deviation over the individual runs. The 5th column lists the average difference between LSID3 and ID3 while the 6th column states whether LSID3 is significantly better or worse than ID3 based on t-test with $p = 0.95$. The last two columns show the same for LSID3 vs. C4.5.

Dataset	ID3	C4.5	ID3-k ($k = 2$)	LSID3 ($r = 5$)	LSID3 vs. ID3 Diff	LSID3 vs. ID3 Sig?	LSID3 vs. C4.5 Diff	LSID3 vs. C4.5 Sig?
Autos-Make	53.7 ±4.0	36.6 ±2.1	56.7 ±1.9	36.8 ±1.8	-16.9 ±4.2	√	0.2 ±2.8	∼
Autos-Symbolling	46.6 ±1.8	31.8 ±1.3	47.1 ±1.9	26.7 ±1.9	-19.9 ±2.2	√	-5.1 ±2.5	√
Balance Scale	354.1 ±6.9	33.7 ±4.5	350.5 ±7.3	347.5 ±6.6	-6.6 ±5.3	√	313.9 ±8.1	×
Breast Cancer	130.4 ±5.2	11.5 ±3.2	124.4 ±5.1	99.7 ±4.3	-30.8 ±5.9	√	88.2 ±4.9	×
Connect4	18534 ±150	3384 ±65	16166 ±97	14622 ±299	-3912 ±406	√	11238 ±303	×
Iris	8.5 ±1.0	4.7 ±0.6	9.1 ±0.9	7.6 ±0.7	-0.9 ±0.7	√	3.0 ±0.7	×
Monks-1	70.0 ±24.2	28.2 ±2.4	27.0 ±0.0	27.0 ±0.0	-43.0 ±24.2	√	-1.2 ±2.4	√
Monks-2	278.7 ±6.8	3.7 ±10.0	265.2 ±6.6	256.6 ±6.2	-22.2 ±7.2	√	252.9 ±12.2	×
Monks-3	37.8 ±3.4	12.8 ±1.0	38.4 ±3.7	36.1 ±3.0	-1.7 ±1.4	√	23.3 ±3.0	×
Solar Flare	69.8 ±3.5	1.1 ±0.4	66.5 ±3.2	59.7 ±3.0	-10.1 ±3.1	√	58.6 ±3.0	×
Tic-Tac-Toe	188.5 ±15.3	82.5 ±7.5	176.6 ±8.6	151.2 ±4.9	-37.3 ±15.3	√	68.7 ±9.3	×
Wine	7.9 ±1.0	5.3 ±0.8	7.4 ±1.1	6.3 ±0.7	-1.6 ±1.1	√	1.0 ±1.2	×
5-XOR	91.7 ±7.7	22.6 ±5.4	78.5 ±10.5	32.2 ±1.8	-59.4 ±7.7	√	9.7 ±5.8	×
3-D XOR	43.6 ±5.2	26.1 ±6.1	17.3 ±8.2	9.3 ±1.0	-34.3 ±5.3	√	-16.9 ±6.2	√
20-Multiplexer	168.9 ±11.4	82.0 ±7.3	45.5 ±28.0	43.4 ±19.5	-125.6 ±21.0	√	-38.7 ±18.8	√
Multiplex-XOR	89.9 ±6.1	22.1 ±4.4	72.7 ±9.5	50.6 ±4.9	-39.2 ±7.2	√	28.5 ±7.0	×
Shapes	18.7 ±5.8	8.5 ±1.0	9.6 ±0.5	9.0 ±0.2	-9.7 ±5.8	√	0.5 ±1.0	×

Following the recommendations of Bouckaert (2003), 10 runs of 10-fold cross-validation experiment were conducted for each dataset, and the reported results are averaged over the 100 individual results.[4] Table 1 compares the size of the trees induced by the four algorithms. Among the three algorithms that force consistency with the training set, the results indicate that for all datasets, the average tree size is smallest when the trees are induced by LSID3. All the improvements in comparison to ID3 were found by t-test to be significant ($p = 0.95$). ID3-k is better than ID3 for most of the datasets, but not all of them. By allowing both pre and post pruning, C4.5 produces smaller trees than LSID3 in most cases. However, these trees are not necessarily consistent with the data.

Reducing the tree size is usually beneficial only if the associated accuracy is not reduced. Table 2 shows the generalization accuracy of the trees produced by the four algorithms. LSID3 significantly outperforms ID3 for 13 out of the 17 datasets. For the other 4 datasets the t-test values indicate that the algorithms are not significantly different. ID3-k is better than ID3 on some datasets. In comparison to LSID3, ID3-k performed much worse for several datasets. When examining both the accuracy and size of the induced trees, for most datasets, the decrease in the size of the trees induced by LSID3 is accompanied by an increase in the predictive power. This phenomenon is consistent with the principle of Occam's Razor.

For the majority of the datasets, LSID3 produces classifiers of higher accuracy than C4.5. For the more difficult concepts, the advantage of LSID3 is substantial. However, for some datasets, such as Breast Cancer and Monks-3, C4.5 produces trees that are both smaller and more accurate. These results confirm our expectations: the problems addressed by LSID3 and C4.5 are different. While LSID3 allows using more time for better learning of hard concepts, C4.5 attempts to simplify the induced trees to avoid the problem of overfitting the data. To exemplify the above let us consider two of the Monks datasets. In the Monks-2 problem all 6 attributes are interdependent and thus greedy strategies fail and pruning does not help and even hurts. In the case of the Monks-3 problem, that is considered relatively easy to learn but contains 5% noise, LSID3 does not improve ID3 while C4.5 produces a tree of higher accuracy that is more than twice smaller.

To test the anytime behavior of the LSID3 and ID3-k algorithms, we invoked LSID3 and ID3-k with different values of r and k. Figure 5 shows the average results over 10 runs of 10-fold cross validation experiment obtained for the Multiplex-XOR dataset. The X axis represents the run time in seconds.[5] ID3, that is a constant time algorithm, finishes in 0.003 second and does not improve with time. Since ID3-k with $k = 1$ and LSID3 with $r = 0$ are defined to be identi-

[4]Except of the Connect4 dataset, for which only one run of 10-fold CV was conducted due to its enormous size.

[5]The algorithms were implemented in C++, compiled by gcc and run on Macintosh G5 2.0 GHz.

Table 2. Classification accuracy for ID3, C4.5, ID3-k and LSID3 on various datasets. The numbers represent the average and standard deviation over the individual runs. The 5th column lists the average difference between LSID3 and ID3 while the 6th column states whether LSID3 is significantly better or worse than ID3 based on t-test with $p = 0.95$. The last two columns show the same for LSID3 vs. C4.5.

Dataset	ID3	C4.5	ID3-k ($k = 2$)	LSID3 ($r = 5$)	LSID3 vs. ID3 Diff	LSID3 vs. ID3 Sig?	LSID3 vs. C4.5 Diff	LSID3 vs. C4.5 Sig?
Autos-Make	78.9 ±9.8	71.0 ±11.4	77.7 ±8.7	80.9 ±9.8	2.0 ±9.7	√	9.9 ±11.8	√
Autos-Symbolling	83.0 ±9.2	76.6 ±10.6	81.9 ±9.2	83.7 ±8.4	0.7 ±9.4	∼	7.1 ±10.0	√
Balance Scale	68.7 ±5.2	64.1 ±5.3	68.6 ±4.8	70.2 ±5.3	1.6 ±4.8	√	6.1 ±4.9	√
Breast Cancer	67.1 ±8.7	72.7 ±9.5	65.1 ±8.4	67.0 ±9.3	-0.2 ±8.2	∼	-5.7 ±9.6	×
Connect4	75.5 ±0.5	78.2 ±0.5	78.0 ±0.7	78.5 ±0.5	3.0 ±0.5	√	0.4 ±0.5	∼
Iris	93.0 ±6.4	93.9 ±5.9	93.1 ±6.7	94.4 ±6.4	1.4 ±4.4	√	0.5 ±4.0	∼
Monks-1	98.1 ±2.9	98.9 ±2.4	100.0 ±0.0	100.0 ±0.0	1.9 ±2.9	√	1.1 ±2.4	√
Monks-2	69.7 ±5.5	65.2 ±5.7	70.6 ±5.9	76.6 ±5.0	6.9 ±5.4	√	11.4 ±7.3	√
Monks-3	97.0 ±2.1	98.9 ±1.4	96.9 ±2.1	96.9 ±2.2	-0.1 ±0.7	∼	-2.0 ±1.8	×
Solar Flare	84.1 ±6.1	88.9 ±5.0	85.4 ±6.2	84.9 ±6.4	0.8 ±3.4	√	-4.0 ±4.6	×
Tic-Tac-Toe	85.5 ±3.7	84.7 ±4.3	84.5 ±3.4	87.7 ±3.2	2.2 ±4.7	√	3.1 ±5.2	√
Wine	92.7 ±7.1	93.1 ±7.2	91.4 ±6.7	93.0 ±5.1	0.3 ±7.4	∼	-0.1 ±7.7	∼
5-XOR	54.4 ±11.9	53.4 ±11.7	56.8 ±15.0	99.9 ±0.7	45.5 ±11.8	√	46.5 ±11.7	√
3-D XOR	55.4 ±13.2	59.3 ±15.9	87.3 ±13.8	96.6 ±4.3	41.2 ±13.1	√	37.3 ±15.9	√
20-Multiplexer	63.3 ±6.9	62.5 ±7.0	96.5 ±8.7	98.1 ±5.0	34.8 ±8.4	√	35.5 ±8.3	√
Multiplex-XOR	50.8 ±11.7	52.3 ±12.4	57.9 ±13.8	74.8 ±10.6	24.0 ±15.0	√	22.5 ±15.7	√
Shapes	96.0 ±5.8	94.8 ±9.2	100.0 ±0.0	100.0 ±0.0	4.0 ±5.8	√	5.2 ±9.2	√

cal to ID3, the point at which ID3 yields a result is also the starting point of the proposed anytime algorithms. The average run time of C4.5 is 0.004 second.

We can see that for LSID3 and ID3-k both tree size and accuracy improve with time. This improvement is larger at the beginning and diminishes over time. The anytime behavior of LSID3 is better in comparison to ID3-k. For ID3-k, the gaps in time between the points grow 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, ID3-k is unable to make use of additional time that is longer than 0.15 second ($k = 3$) but shorter than 1.2 second ($k = 4$). For LSID3, the difference in the run-time for any 2 successive values of r is almost the same. Except of a small period of time, ID3-k is dominated by LSID3. This implies that when additional time is available LSID3 should be preferred over ID3-k for the task of learning this concept.

Figure 6 shows the performance of both algorithms when applied on the Tic-tac-toe dataset. In this case, the average run time of ID3 is 0.004 second, while C4.5 finished in 0.008 second. LSID3 dominates ID3-k at any point of time, both in terms of accuracy and size. ID3-k is clearly not well-behaved in this case. In addition to the problem of large gaps between successive possible time allocations mentioned above, a decrease in the accuracy and increase in the size of the tree is observed at $k = 3$. Similar cases of pathology caused by limited-depth lookahead have been reported by Murthy and Salzberg (1995).

4. Related Work

While, to our knowledge, no other work tried specifically to design anytime algorithm for decision tree induction, there are several related works that need to be discussed here. Kononenko and Kovacic (1992) applied stochastic search methods, such as stochastic hill climbing and simulated annealing for learning decision rules from examples: rules are randomly generated and iteratively improved until a local maxima is reached.

Lookahead techniques have been applied to decision tree induction by several researchers. The reported results vary from *lookahead produces better trees* (Norton, 1989; Ragavan & Rendell, 1993; Dong & Kothari, 2001) to *lookahead does not help and can hurt* (Murthy & Salzberg, 1995). One problem with these works is their use of a fixed low depth lookahead, therefore, disqualifying them from serving as anytime algorithms.

The boosting method (Schapire, 1999), which iteratively refines the constructed classifier by increasing the weight of misclassified instances, can be viewed as an anytime algorithm. However, unlike the problem we face in this work, the classifier constructed by the boosting algorithm consists of an ensemble of decision trees rather than a single one.

Page and Ray (2003) presented *Skewing* as an alternative to lookahead for addressing problematic concepts such as parity functions. At each node, the algorithm

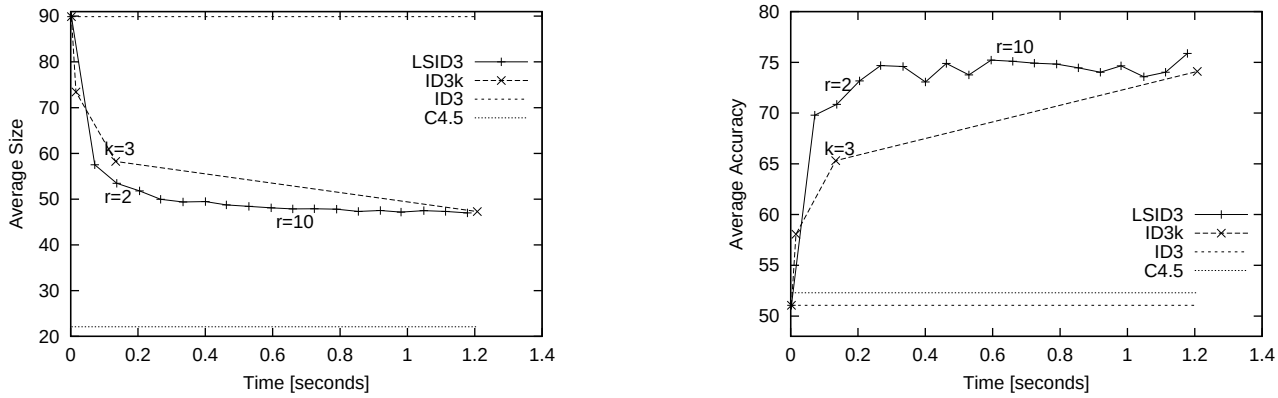

Figure 5. Average results over 10 runs of 10-fold cross validation experiment on the Multiplex-XOR dataset.

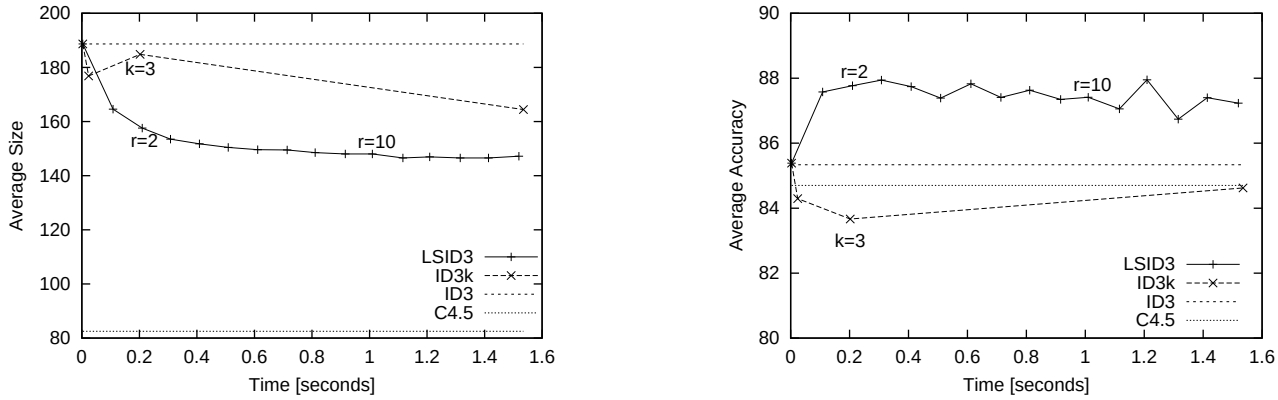

Figure 6. Average results over 10 runs of 10-fold cross validation experiment on the Tic-tac-toe dataset.

skews the set of examples and produces several versions of it, each with different weights to the instances. The attribute that exceeds a pre-set gain threshold for the greatest number of weightings is chosen to split on. Skewing is currently limited to nominal attributes. Moreover, the method can become harmful when the training set is small. While not presented and studied as an anytime algorithm, the skewing method can be viewed as a contract algorithm parameterized by the number of weightings. We intend to implement such version and compare its anytime behavior to LSID3.

Several researchers (Murphy & Pazzani, 1994; Webb, 1996) doubt the usefulness of Occam's Razor for decision tree induction. In our experiments, we never found a case where the decrease in the size of trees induced by LSID3 leads to a significant deterioration in accuracy. In many cases, when the tree was significantly smaller, the accuracy was also significantly higher. We believe that the different findings about the utility of Occam's Razor is due to the different type of concepts tested and the different algorithms used. We handled more difficult concepts, such as parity, and used more sophisticated lookahead algorithms that allowed us to find trees unobtainable by other methods.

5. Conclusions

In this work, we presented and empirically evaluated two lookahead-based algorithms for anytime induction of decision trees, namely ID3-k and LSID3. Existing greedy algorithms for learning decision trees require a fixed small amount of time. For example, for 16 out of the 17 datasets used in our experiments, ID3 finished in less than 0.1 second. We showed that for several real-world and synthetic datasets, both LSID3 and ID3-k can make use of higher budget of time. In most cases, when LSID3 was allocated few minutes, it produced trees of smaller size and of higher accuracy. The usage of more time is shown to be worthwhile when the concepts are hard and involve interdependencies between the attributes. In these cases, most of the existing greedy methods fail. Moreover, methods that attempt to simplify the trees by pruning do

not help and even hurt. An open question that we intend to tackle in future work is how our lookahead-based anytime algorithms perform when implemented on the top of C4.5. This expands the search space to trees that do not perfectly fit the training data, and thus allows handling noisy training sets as well.

When we examined the anytime behavior of LSID3 and ID3-k, we found that LSID3 has a better anytime behavior. The gap between the points of time at which ID3-k makes use of additional time grows rapidly with k, the depth of the lookahead. This limits the flexibility of ID3-k to trade between time and quality. In the case of LSID3, which is parameterized by the number of times SID3 is invoked for each candidate, the gap is almost constant. Another problem with ID3-k is that in some cases lookahead to depth k is insufficient to correctly predict the utility of an attribute.

The major contribution of this paper is the LSID3 algorithm which enables deep lookahead using stochastic search. The algorithm exhibits good anytime behavior and serves as a contract algorithm for producing better decision trees with additional resource allocation. This algorithm is only a first step in this direction. We are currently in the process of designing and implementing many variations of it, including algorithms that combine ID3-k and LSID3.

References

Baram, Y., El-Yaniv, R., & Luz, K. (2003). Online choice of active learning algorithms. *ICML'03* (pp. 19–26).

Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases.

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

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

Bouckaert, R. R. (2003). Choosing between two learning algorithms based on calibrated tests. *ICML'03* (pp. 51–58).

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees.* Monterey, CA: Wadsworth and Brooks.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Dong, M., & Kothari, R. (2001). Look-ahead based fuzzy decision tree induction. *IEEE-FS*, *9*, 461–468.

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

Kononenko, I., & Kovacic, M. (1992). Learning as optimization: Stochastic generation of multiple knowledge. *ICML'92* (pp. 257–262).

Minsky, M., & Papert, S. (1969). *Perceptrons: Introduction to computational geometry.* MIT Press, Cambridge, MA.

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

Murphy, P. M., & Pazzani, M. J. (1994). Exploring the decision forest: an empirical investigation of Occam's Razor in decision tree induction. *Journal of Artificial Intelligence Research*, *1*, 257–275.

Murthy, S. K., & Salzberg, S. (1995). Lookahead and pathology in decision tree induction. *IJCAI'95* (pp. 1025–1033).

Norton, S. W. (1989). Generating better decision trees. *IJCAI'89* (pp. 800–805).

Page, D., & Ray, S. (2003). Skewing: An efficient alternative to lookahead for decision tree induction. *IJCAI'03*.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* San Mateo, CA: Morgan Kaufmann.

Ragavan, H., & Rendell, L. (1993). Lookahead feature construction for learning hard concepts. *ICML'93* (pp. 252–259).

Russell, S. J., & Zilberstein, S. (1991). Composing real-time systems. *IJCAI'91* (pp. 212–217).

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

Schapire, R. (1999). A brief introduction to boosting. *IJCAI'99* (pp. 1401–1406).

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