

# Anytime Induction of Decision Trees: an Iterative Improvement Approach

Saher Esmeir and Shaul Markovitch

Computer Science Department  
Technion—Israel Institute of Technology, Haifa 32000, Israel  
{esaher , shaulm}@cs.technion.ac.il

## Abstract

Most existing decision tree inducers are very fast due to their greedy approach. In many real-life applications, however, we are willing to allocate more time to get better decision trees. Our recently introduced LSID3 contract anytime algorithm allows computation speed to be traded for better tree quality. As a contract algorithm, LSID3 must be allocated its resources a priori, which is not always possible. In this work, we present IIDT, a general framework for interruptible induction of decision trees that need not be allocated resources a priori. The core of our proposed framework is an iterative improvement algorithm that repeatedly selects a subtree whose reconstruction is expected to yield the highest marginal utility. The algorithm then rebuilds the subtree with a higher allocation of resources. IIDT can also be configured to receive training examples as they become available, and is thus appropriate for incremental learning tasks. Empirical evaluation with several hard concepts shows that IIDT exhibits good anytime behavior and significantly outperforms greedy inducers when more time is available. A comparison of IIDT to several modern decision tree learners showed it to be superior.

## Introduction

Despite the recent progress in developing advanced induction algorithms, such as SVM (Vapnik 1995), *decision trees* are still considered attractive for many real-life applications. This is due in large part to their interpretability (Hastie, Tibshirani, & Friedman 2001). Craven (1996) listed several reasons why the comprehensibility of a classifier is so important. For one thing, comprehensibility makes it possible for humans to validate the induced model and to generate human-readable explanations for the classifier predictions. When classification cost is high, 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 were shown to be competitive with other classifiers for several learning tasks.

Smaller consistent decision trees are likely to have better predictive power than their larger counterparts. This is in accordance with Occam's Razor (Blumer *et al.* 1987;

Domingos 1999). The problem of finding the smallest consistent tree, however, is known to be NP-complete (Hyafil & Rivest 1976; Murphy & McCraw 1991). That is why most existing decision tree induction algorithms take a greedy approach and use local heuristics for choosing the best splitting attribute. The greedy approach indeed performs quite well for many learning problems and is able to generate decision trees very fast. In some cases, however, when the concept to learn is hard and the user is willing to allocate more time, the existing greedy algorithms are not able to exploit the additional resources to generate a better decision tree.<sup>1</sup>

Recently, we have introduced the LSID3 algorithm that can induce better decision trees given larger allocation of time (Esmeir & Markovitch 2004). The algorithm evaluates a candidate splitting attribute by estimating the size of the smallest consistent tree under it. The estimation is based on sampling the space of consistent trees, where the size of the sample is determined in advance according to the allocated time. LSID3 requires advanced knowledge of the allocated time and does not guarantee any solution if the contract is not honored. Such algorithms are called *contract* anytime algorithm (Russell & Zilberstein 1996).

In many real-life applications, however, the time allocated for the learning phase is not known a priori. In this work we present IIDT<sup>2</sup>, a general framework for *interruptible* induction of decision trees, which does not require that time be allocated in advance and can therefore be interrupted whenever necessary. At the center of our proposed framework is an iterative improvement algorithm that repeatedly selects a subtree whose reconstruction is expected to yield the highest marginal utility. The algorithm then rebuilds this subtree with a higher resource allocation. The algorithm can be configured to prune the tree in order to avoid overfitting. It can also be easily modified to handle incremental learning tasks.

## LSID3: a Contract TDIDT Algorithm

In this section we give a brief overview of the LSID3 algorithm, which we use as a component of the interruptible framework that will be presented in the next section.

<sup>1</sup>There are applications where the allocated time is not sufficient for building a tree even by a greedy algorithm. In this work we do not deal with such problems.

<sup>2</sup>Interruptible Induction of Decision Trees

```

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

```

\* When  $a$  is numeric the loop is over all possible splitting tests and  $a$  is not filtered out when calling SID3.

Figure 1: Attribute selection in LSID3

LSID3 adopts the general top-down induction of decision trees (TDIDT) scheme, and invests more resources for making better split decisions. For every possible candidate split, LSID3 attempts to estimate the size of the resulting subtree were the split to take place, and favors the one with the smallest expected size. The estimation is based on a biased sample of the space of trees rooted at the evaluated attribute. The sample is obtained using a stochastic version of ID3, called SID3. In SID3, the splitting attribute is chosen semi-randomly with a probability proportional to its information gain. LSID3 is a contract algorithm parameterized by  $r$ , the sample size. When  $r$  is larger, we expect the resulting estimation to be more accurate and LSID3 to improve. Figure 1 lists the attribute selection components in LSID3.

The runtime of LSID3 grows linearly with  $r$ . Let  $m$  be the number of examples and  $n = |A|$  be the number of attributes. Shavlik, Mooney, & Towell (1991) reported for ID3 an empirically based average-case complexity of  $O(mn)$ . 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, and thus the runtime of LSID3( $r$ ) can be written as

$$\sum_{i=1}^n r \cdot i \cdot O(mi) = \sum_{i=1}^n O(rmi^2) = O(rmn^3). \quad (1)$$

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  $r$  is difficult. To handle such cases, we need to come up with an interruptible anytime algorithm.

### Interruptible Induction of Decision Trees

By definition, every interruptible algorithm can serve as a contract algorithm. Russell & Zilberstein (1996) showed that 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  $r = 0$ ,

```

Procedure IIDT( $E, A$ )
 $T \leftarrow \text{GREEDY-BUILD-TREE}(E, A)$ 
While not-interrupted
   $y \leftarrow \text{CHOOSE-NODE}(T, E, A)$ 
   $t \leftarrow$  subtree of  $T$  rooted at  $node$ 
   $r \leftarrow \text{NEXT-R}(node)$ 
   $t' \leftarrow \text{REBUILD-TREE}(\text{EX}(y), \text{ATT}(y), r)$ 
  If BETTER( $t', t$ ) replace  $t$  with  $t'$ 
Return  $T$ 

Procedure EX( $node$ )
Return  $\{e \in E \mid e \text{ reaches } node\}$ 

Procedure ATT( $node$ )
Return  $\{a \in A \mid a \notin \text{ancestor of } node\} \cup$ 
 $\{a \in A \mid a \text{ is numeric}\}$ 

```

Figure 2: The IIDT framework

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  $r = 1$ .

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 entire decision tree is rebuilt. In addition, the minimal value that can be used for  $\tau$  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. We start by describing the IIDT framework and its components. Then, we show how IIDT can be configured to prune the resulting trees and how it can be modified to handle incremental learning tasks.

### The IIDT Algorithm

Iterative improvement approaches, which start with an initial solution and repeatedly modify it, have been successfully applied to many AI domains. The key idea behind these 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 repairs 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 repeatedly modify split decisions.

IIDT first performs a quick construction of an initial de-

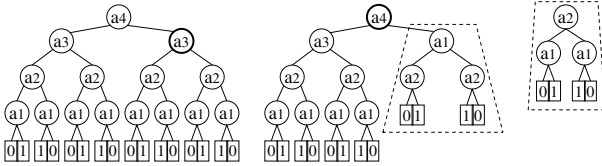


Figure 3: Improving a decision tree by IIDT

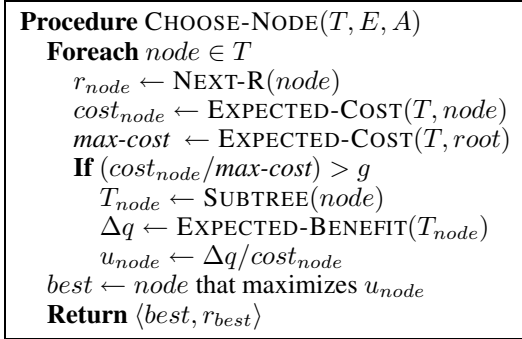


Figure 4: Choosing a subtree for reconstruction

cision 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, the existing subtree is replaced. IIDT is formalized in Figure 2. Figure 3 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 entire tree is replaced by a tree that perfectly describes the concept.

As a general framework, IIDT can use different approaches for choosing which node to improve, for allocating resources, for rebuilding a subtree, and for deciding whether an alternative subtree is better.

Once the subtree to rebuild is chosen and the resource allocation is determined, 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. LSID3 receives its resource allocation in terms of  $r$ , the sample size. 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 repair process. Following Russell & Zilberstein (1996), the optimal strategy 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  $\text{NEXT-R}(y) = 2 \cdot \text{LAST-R}(y)$ . Note that when a node  $y$  is chosen for the first time  $\text{LAST-R}(y) = 0$ . In this case the next allocation will be  $\text{NEXT-R}(y) = 1$ .

Intuitively, the next node we would like to improve is the one with the highest expected marginal utility (Hovitz 1990;

Russell & Wefald 1989), i.e., the one with the highest ratio between the expected benefit and the expected cost. Predicting the expected benefit and expected cost of rebuilding a subtree is a difficult problem. In what follows we show how to approximate these values, and how to incorporate these approximations into the node selection procedure.

*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 the expected runtime of  $\text{LSID3}(r)$  for rebuilding the subtree below a node  $y$  by  $\text{NEXT-R}(y) \cdot m \cdot n^3$ , 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 expected to be less costly because they have fewer examples and attributes to consider. Thus, their expected runtime is shorter. Further, 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.

*Granularity:* Considering the cost approximation described above, the selection procedure may repeatedly attempt to improve subtrees rooted at deep nodes due to their low associated costs. This behavior would indeed be beneficial in the short term, 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 nodes are 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 any-time performance flexibility, we add a *granularity parameter*  $0 \leq g \leq 1$ . This parameter serves as a threshold for the 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.

*Expected benefit:* Following Occam's Razor, the size of a subtree can serve as a measure for its quality. It is difficult, however, to know 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 size reduction of an existing subtree is to a depth of 2. Assuming that the maximal number of values per attribute is  $b$ , the maximal size of such a tree (measured by the number of leaves) is  $b^2$ . Hence, an upper bound on the benefit from reconstructing an existing tree  $t$  is  $\text{SIZE}(t) - b^2$ . If the expected costs are ignored and only the expected benefits considered, the highest score would always be given 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. Note that there may be nodes whose cost 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,

```

Procedure TS-GREEDY-BUILD-TREE( $E, A$ )
  Return ID3( $E, A$ )

Procedure TS-REBUILD-TREE( $E, A, r$ )
  Return LSID3( $E, A, r$ )

Procedure TS-EXPECTED-COST( $T, node$ )
  Return NEXT-R( $node$ ) · |EX( $node$ )| · |ATT( $node$ )|3

Procedure TS-EXPECTED-BENEFIT( $T$ )
   $l-bound \leftarrow (\min_{a \in A_{node}} |DOMAIN(a)|)^2$ 
  Return TREE-SIZE( $T$ ) –  $l-bound$ 

Procedure TS-BETTER( $T_1, T_2$ )
  Return TREE-SIZE( $T_1$ ) < TREE-SIZE( $T_2$ )

```

Figure 5: IIDT-TS

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 method.

*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 a tree of lower quality, IIDT replaces an existing subtree with a newly induced alternative only if the latter is expected to improve the quality of the complete tree. As in the expected benefit approximation, we measure the usefulness of a tree 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.

Figure 4 formalizes the node selection component in IIDT. We refer to the above-described instantiation of IIDT that uses the tree size as a quality metric by *IIDT-TS*. Figure 5 formalizes IIDT-TS.

### Pruning the IIDT Trees

Pruning tackles the problem of how to avoid overfitting the data, mainly in the presence of classification noise. Pruning, however, is not able to overcome wrong decisions made when learning hard concepts. Therefore, our anytime approach and pruning address different problems: whereas pruning attempts to simplify the induced trees to avoid overfitting, IIDT attempts to exploit additional time for better learning of hard concepts. We view pruning as orthogonal to lookahead and suggest considering it to avoid overfitting.

Pruning can be applied to the trees produced by IIDT in two different ways. The first is to post-prune a tree when it is returned to the user. In this case IIDT maintains consistent trees and only when interrupted, it applies a post-pruning phase. When resumed, IIDT continues to improve the unpruned tree. This solution requires the user take into account the delay imposed by pruning. In the anytime setup, where much time is invested for growing the tree, this delay is usually negligible with respect to the growing time.

```

Procedure EE-GREEDY-BUILD-TREE( $E, A$ )
  Return C4.5( $E, A$ )

Procedure EE-REBUILD-TREE( $E, A, r$ )
  Return LSID3-P( $E, A, r$ )

Procedure EE-EXPECTED-COST( $T, node$ )
  Return NEXT-R( $node$ ) · |EX( $node$ )| · |ATT( $node$ )|3

Procedure EE-EXPECTED-BENEFIT( $T$ )
   $l-bound \leftarrow 0$ 
  ForEach  $c \in C$ 
     $E_c \leftarrow \{e \in E_T, CLASS(e) = c\}$ 
     $l-bound \leftarrow l-bound + EXPECTED-ERROR(E_c)$ 
  Return EXPECTED-ERROR( $E_T$ ) –  $l-bound$ 

Procedure EE-BETTER( $T_1, T_2$ )
  Return  $\sum_{l \in T_1} EXPECTED-ERROR(E_l) <$ 
          $\sum_{l \in T_2} EXPECTED-ERROR(E_l)$ 

```

Figure 6: IIDT-EE

The second alternative we propose maintains (possibly) inconsistent trees by building a pruned initial tree (using C4.5 (Quinlan 1993)), and by post-pruning each of the replacement trees (using LSID3-p). In this case the tree can be returned immediately, without any post-pruning delay.

Using the size of a tree as a measure for its quality is not applicable to inconsistent trees: one can easily create a tree of size 1 when consistency is not forced. One possible solution is to evaluate the quality of a subtree by its *expected error*. The expected error of a tree is measured by summing up the expected errors of its leaves, exactly as in C4.5's error-based pruning. We denote this instantiation of IIDT by *IIDT-EE*. In IIDT-EE, the expected benefit of rebuilding a subtree is measured by the maximal possible reduction in the expected error. A trivial lower bound for the expected error is the error if all examples were immediately split into subsets according to their label. Figure 6 lists IIDT-EE.

### Incremental IIDT

As an anytime algorithm, IIDT continuously invests resources in attempt to improve the induced tree. It is not designed, however, to exploit additional data that becomes available with time. One way to handle examples that are presented incrementally is to rerun the inducer when a new example arrives. The restart approach is very reasonable for greedy learners, where the time “wasted” by starting the induction process from scratch is minimal. This straightforward conversion, however, is much more problematic in the case of anytime learners: it is not reasonable to discard the current hypothesis, which was produced using significant resources, just because a new example arrives. Greedy restructurings, as suggested in ID5R (Utgoff 1989) and its successive ITI (Utgoff, Berkman, & Clouse 1997), allow to quickly obtain the same greedy tree as in the batch setup. Applying such corrections to IIDT, however, will nullify its







