Interruptible Anytime Algorithms for Iterative Improvement of Decision Trees

Saher Esmeir
Computer Science Department
Technion—IIT
Haifa 32000, Israel
esaher@cs.technion.ac.il

Shaul Markovitch
Computer Science Department
Technion—IIT
Haifa 32000, Israel
shaulm@cs.technion.ac.il

ABSTRACT

Finding a minimal decision tree consistent with the examples is an NP-complete problem. Therefore, most of the existing algorithms for decision tree induction use a greedy approach based on local heuristics. These algorithms usually require a fixed small amount of time and result in trees that are not globally optimal. Recently, the LSID3 contract anytime algorithm was introduced to allow using extra resources for building better decision trees. A contract anytime algorithm needs to get its resource allocation a priori. In many cases, however, the time allocation is not known in advance, disallowing the use of contract algorithms. To overcome this problem, in this work we present two interruptible anytime algorithms for inducing decision trees. Interruptible anytime algorithms do not require their resource allocation in advance and thus must be ready to be interrupted and return a valid solution at any moment. The first interruptible algorithm we propose is based on a general technique for converting a contract algorithm to an interruptible one by sequencing. The second is an iterative improvement algorithm that repeatedly selects a subtree whose reconstruction is estimated to yield the highest marginal utility and rebuilds it with higher resource allocation. Empirical evaluation shows a good anytime behavior for both algorithms. The iterative improvement algorithm shows smoother performance profiles which allow more refined control.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept-learning, Induction*; H.2.8 [**Database Management**]: Applications—*Data Mining*

General Terms

Algorithms, Experimentation

Keywords

Decision trees, Anytime algorithms, Anytime learning, Cost-quality tradeoff, Hard concepts

UBDM '05, August 21, 2005, Chicago, Illinois, USA.

1. INTRODUCTION

Despite the recent progress in developing advanced induction algorithms, such as Support Vector Machines [7], *decision trees* [6, 22] are still considered attractive for many real-life applications mostly due to their interpretability [13]. Craven and Shavlik [8] listed several reasons for the importance of the comprehensibility of learned classifiers. These reasons include, among others, the possibility to validate the induced model by human and to generate human-readable explanations for the classifier predictions.

Another model evaluation criterion that is mentioned in [8] is the flexibility of the model representation. In this manner, decision trees have a great advantage: they can be easily converted into logical rules. When classification cost is an important factor, decision trees are favored since they test only values of the features on the path from the root to the relevant decision leaf. In terms of accuracy, decision trees were shown to be competitive with other classifiers for several learning tasks [12, 28, 15, 32].

Based on the Occam's Razor principle [2], small decision trees that are consistent with the training examples have better predictive power than their larger counterparts. Finding the smallest consistent tree, however, was shown to be NP-complete [16, 19]. For this reason, 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 for generating a better decision tree.

Algorithms that are able to trade resources for the quality of the output are called *anytime algorithms* [25, 3]. Recently, the LSID3 algorithm for anytime induction of decision trees has been introduced [9]. The algorithm performs repeated lookahead probes in order to evaluate candidate splitting attributes. The number of repetitions is determined in advance according to the allocated time.

LSID3 requires that the allocated time will be known ahead and does not guarantee any solution if the allocation is not honored, and hence is called a *contract anytime algorithm*. As such, LSID3 has two shortcomings. First, in many real-

life applications the time allocated for the learning phase is not known a priori. One example for such a setup is when the user is willing to allow the induction algorithm to run until the classifier is needed, and the time for this event is not known in advance. Another example is an application where the user wants the induction algorithm to run until it reaches some expected accuracy on a set-aside validation set. A second problem is that LSID3 assumes a mapping from the time allocation to the contract parameter, i.e., to the number of lookahead probes the algorithm can afford. In many cases, however, such a mapping is not possible.

To overcome the above problems, we need to come up with an *interruptible anytime algorithm*, which does not require the allocated time in advance and can therefore be interrupted anytime. In this work we present two interruptible anytime algorithms for decision tree induction, that can trade off the learning cost for the quality of the produced hypothesis and allow queries for solution at any moment. We start with a method that converts LSID3 to an interruptible algorithm using the general sequencing method described in [26]. 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 new repair-based algorithm, *IIDT*, that repeatedly replaces subtrees of the current hypothesis by subtrees generated with higher resource allocation and are therefore expected to be better. The two methods are empirically tested on datasets representing difficult concepts. Note that in this paper we assume a batch setup where all the training examples are given at the beginning of the learning process, unlike the incremental setup in which the induced tree is restructured when a new training instance becomes available [30].

2. CONTRACT INDUCTION OF DECISION TREES

The interruptible algorithms presented in the next sections both use a contract algorithm as a component. Specifically, in this paper we use the LSID3 algorithm. This section gives a short overview of this algorithm.

LSID3 adopts the top-down induction of decision trees (TDIDT) scheme. Under this framework, an attribute is chosen to partition the entire dataset into subsets, each of which is used to recursively build a subtree.

In ID3 each candidate split is evaluated by the information gain it yields and the attribute that maximizes this measure is selected. In LSID3 we measure the usefulness of a candidate split by the expected size of the subtree it results in. One can estimate this size by calling ID3 itself. Nevertheless, this results in a fixed time algorithm rather than in an anytime one. Moreover, ID3 might be insufficient to correctly predict the size. Therefor, in order to produce a better estimation of the tree size, instead of calling ID3 once, LSID3 samples the space of "good" trees by repeatedly invoking a stochastic version of ID3 (SID3). In SID3, 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. Since SID3 is not a deterministic algorithm, different runs of it might return different trees of different sizes. The size of each tree is an upper bound on the optimal tree size and hence we consider the minimal one as the estimator.

```
Procedure LSID3-Choose-Attribute(E, A, r)
  If r = 0
    Return ID3-Choose-Attribute(E, A)
  Foreach a ∈ A
    Foreach v_i ∈ domain(a)
      E_i ← {e ∈ E | a(e) = v_i}
      min_i ← ∞
      Repeat r times
        T ← SID3(E_i, A − {a})
        min_i ← min(min_i, Size(T))
    total_a ← Σ_{i=1}^{|domain(a)|} min_i
  Return a for which total_a is minimal
```

Figure 1: Attribute selection in LSID3.

```
Procedure Sequenced-LSID3(E, A)
    T ← ID3(E, A)
    i ← 0
    While not-interrupted
        r ← 2^i
        T ← LSID3(E, A, r)
        i ← i + 1
    Return T
```

Figure 2: Conversion of LSID3 to an interruptible algorithm by sequenced invocations.

Given an attribute a, LSID3 partitions the set of examples according to the different values a can take and calls SID3 several times for each subset. a is evaluated by summing up the estimated size of each subtree. For each subtree there are several estimations obtained from several calls to SID3. The algorithm considers the minimal one.

LSID3 is a contract algorithm parameterized by r, the number of times SID3 is called for each candidate. LSID3 with $r = 0$ is defined to be ID3. Figure 1 formalizes the choice of splitting attributes as made by LSID3. The runtime of LSID3 grows linearly with r. Let m be the number of examples and $n = |A|$ be the number of attributes. The worst-case time complexity of ID3 is $O(mn^2)$ [30]. It is easy to see that SID3 has the same worst-case complexity. LSID3(r) invokes SID3 r times for each candidate split. Recall the analysis in [30] for the time complexity of ID3, we can write the runtime of LSID3(r) as $\sum_{i=1}^{n} r \cdot i \cdot O(mi^2)$. An empirical based average-case analysis for ID3 showed that the complexity of ID3 is actually linear in n rather than quadratic i.e., $O(nm)$ [29]. Hence, we derive that the average case complexity of LSID3 is

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

3. SEQUENCING CONTRACT ANYTIME ALGORITHMS

By definition, every interruptible algorithm can serve as a contract one. Russell and Zilberstein [26] 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 $\tau, 2\tau, \ldots, 2^i\tau, \ldots$ where τ is a free parameter that affects the granularity of the composed algorithm. Smaller values of τ will yield more frequent improvements, but there is no benefit in setting it to a too low value for which the improvement in quality is insignificant. It can be shown that the above sequence of runtimes is optimal when the different runs are scheduled on a single processor [26].

This general approach can be used to convert LSID3 into an interruptible algorithm. LSID3 gets its contract time in terms of r, the number of samplings per node. When $r = 0$, LSID3 is defined to be identical to ID3 which requires much less time than LSID3 with $r = 1$. Therefore, we slightly modify the sequencing method by first calling LSID3 with $r = 0$ and then continue according to the original method with exponentially increasing values of r, starting from $r = 1$. Figure 2 formalizes the resulting algorithm.

One problem with the sequencing approach is the exponential growth of the gaps between the points of time at which an improved result can be obtained. This is due to the generality of the algorithm that views the contract algorithm as a black-box. Thus, in the case of LSID3 at each iteration the whole decision tree is rebuilt. In Section 4 we present an interruptible anytime algorithm that instead of trying to rebuild the whole tree, iteratively improves subtrees.

4. INTERRUPTIBLE INDUCTION BY ITERATIVE IMPROVEMENT

In this section we present IIDT, an interruptible algorithm for decision-tree learning. As in LSID3, IIDT exploits additional resources in attempt to produce better trees. The key 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, on the contrary, does not get its resource allocation in advance and might be queried for a solution at any moment.

IIDT first performs a quick induction of an initial tree by calling ID3. It then iteratively attempts to improve the current tree by choosing a node, computing its next resource allocation and rebuilding the subtree below it. If the newly induced subtree is better than the existing one, a replacement takes place. We formalize IIDT in Figure 3.

Figure 4 illustrate the way IIDT works. The target concept is $a_1(x) \oplus a_2(x)$ with additional two 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 that allows using different approaches for choosing the node to improve, for allocating resources for an improvement iteration, for rebuilding a subtree and for deciding whether an alternative subtree is better or not. In

```
Procedure IIDT(E, A)
    T ← ID3(E, A)
    While not-interrupted
        node ← CHOOSE-NODE(T, E, A)
        t ← subtree of T rooted at node
        A_node ← {a ∈ A | a ∉ ancestor of node}
        E_node ← {e ∈ E | e reaches node}
        r ← NEXT-R(node)
        t' ← REBUILD-TREE(E_node, A_node, r)
        If EVALUATE(t) > EVALUATE(t')
            replace t with t'
    Return T
```

Figure 3: Interruptible learning of decision trees.

the remainder of this section we focus on the components of IIDT and suggest a possible implementation that is based on LSID3.

4.1 Reconstructing a Subtree

After deciding upon the amount of resources allocated for the reconstruction process, the problem becomes a task for a contract algorithm. A good candidate for such an algorithm is LSID3 which exhibited good anytime performance in the empirical study reported in [9]. We expect that calling LSID3 with higher resource allocation will result in a better subtree.

4.2 Choosing a Subtree to Improve

Intuitively, the next node we would like to improve is the one with the highest expected marginal utility, i.e., the one with the highest ratio of expected benefit and expected cost [14, 24]. Estimating the expected cost and expected gain of rebuilding a subtree is a difficult problem. There is no apparent way for estimating the expected improvement either in terms of tree size or generalization accuracy. In addition, precise prediction of the resources to be consumed by LSID3 is not an easy task. In the remainder of this subsection we show how to approximate these values, and how to incorporate these approximations into the node selection algorithm.

4.2.1 Resource Allocation

The LSID3 algorithm receives its resource allocation in terms of r, the number of samplings devoted for each attribute. We adopt here the above mentioned strategy that doubles the amount of resource allocation at each iteration. Thus, if the resources allocated for the last improvement attempt of *node* were $r = \text{LAST-R}(node)$, the next allocation will be $2r$.[1]

4.2.2 Expected Cost

The expected cost can be approximated using the average time complexity of LSID3 as expressed by Equation 1. For each *node*, we estimate the expected runtime of LSID3(r) to rebuild the subtree below it by $\text{COST}(node) = \text{NEXT-R}(node) \cdot m \cdot n^3$.

[1]Note that LAST-R can be inherited from an ancestor of *node*, in case the subtree rooted at *node* was reproduced as a part of a containing tree.

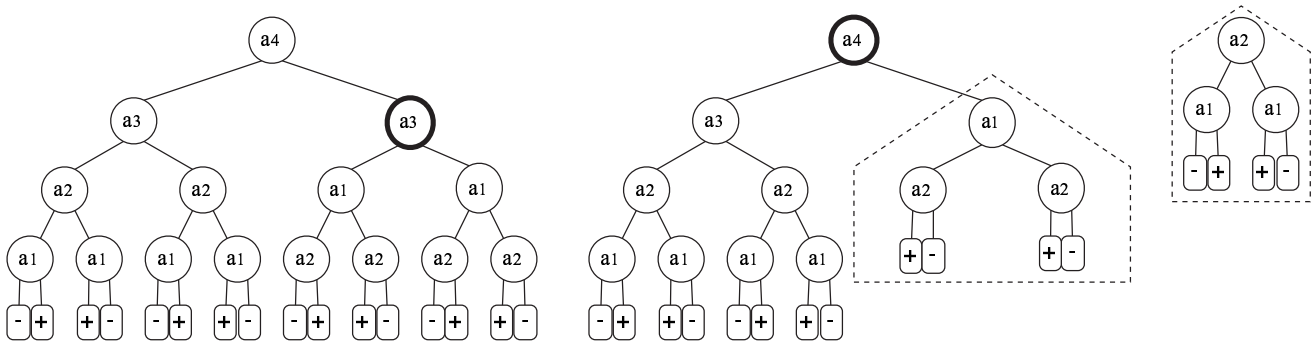

Figure 4: Iterative improvement of the decision tree produced for the 2-XOR concept $a_1(x) \oplus a_2(x)$ with additional two irrelevant attributes a_3 and a_4.

We observe that in terms of expected cost, subtrees rooted in deeper levels are preferred since they have less examples and attributes to consider and thus have shorter expected runtime. We also observe that since for each node the next time allocation doubles the previous one, nodes that were previously selected for improvement a large number of times, will have higher associated costs and thus are less likely to be chosen again.

4.2.3 *Expected benefit*

The whole framework of decision trees induction rests on the assumption that smaller consistent trees are better than large ones. Therefore the size of a subtree can serve as a measure for its quality. We cannot, however, know or estimate the size of the reconstructed subtree before actually building it. Therefore, we use instead an upper limit on the reduction in size that can be achieved.

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 and by LSID3. Similarly if a subtree was replaceable by another subtree of depth 1, i.e., consists of a single split, 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 (measured by the number of leaves) is b^2. Hence, an upper bound on the benefit from reconstructing a tree t that was previously induced is $\textsc{Size}(t) - b^2$.

Ignoring the expected costs, and relying solely on the expected benefit results in always giving the highest score to the root node. This makes sense: assuming we have infinite resources, we would attempt to improve the whole decision tree rather than parts of it.

4.2.4 *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 due to their low associated costs. In the short term, such a behavior would indeed be beneficial but in the long term it can be harmful since when the algorithm later improves subtrees in upper levels, the resources spent on deeper nodes are wasted. On the other hand, if the algorithm would have first selected the upper level trees, this waste would be avoided, but the time gaps between potential improvements would have been increased.

To allow control of the tradeoff between the efficiency of resource usage and the flexibility of anytime control we add a granularity parameter $0 \leq g \leq 1$ that serves as a threshold for the minimal time allocation to 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 by this definition, it is possible to have nodes with cost which is higher than the cost of the root node, and therefore with relative cost higher than one. Such nodes, however, can never be selected for improvement since 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 the sequencing algorithm described in Section 3.

Figure 5 formalizes the procedure for choosing a node for reconstruction.

4.3 Evaluating a Subtree

Although LSID3 was shown to produce better trees when allocated more resources, an improved result is not guaranteed. Thus, to avoid a degradation in the quality of the induced tree, we replace an existing subtree 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 of a smaller size, would it replace an existing subtree. This guarantees that the size of the complete decision tree is monotonically decreasing.

Another possible measure is the accuracy of the decision

Procedure CHOOSE-NODE(T, E, A)
  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($node$)
    $cost_{node} \leftarrow r_{node} \cdot |E_{node}| \cdot |A_{node}|^3$
    $max\text{-}cost \leftarrow$ NEXT-R($root$) $\cdot |E| \cdot |A|^3$
    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 5: Choosing a node for reconstruction.

tree on a set-aside validation set of examples. Only if the accuracy on the validation set increases, the modification is 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 and thus makes the learning process harder. The second is the bias towards overfitting the validation set, that might reduce the generalization abilities of the tree.

5. EXPERIMENTAL EVALUATION

A variety of experiments were conducted to test the performance and anytime behavior of IIDT. We compare two versions of IIDT to the fixed time algorithms ID3 and C4.5. Both versions of IIDT use the components described in Section 4. IIDT(1) is parameterized with a granularity factor 1 and thus behaves exactly as the sequencing method described in Section 3, while in IIDT(0.1) the granularity factor is set to 0.1. In addition, we tested a version of IIDT that evaluates subtrees by their accuracy on a validation set rather than their size. However, this modification did not help and in some cases a degradation in the performance was observed. Thus, in what follows we describe the results for the size-based estimation.

The behavior of anytime learners on easy concepts is not interesting since the greedy algorithms are able to produce good trees with small allocation of resources. Therefore, we present here the results for more complex concepts that can benefit from larger resource allocation: the *Glass* and the *Tic-Tac-Toe* UCI datasets [1], the *20-Multiplexer* dataset [23] and the 10-XOR dataset, generated with additional 10 irrelevant attributes. Table 1 summarizes the basic characteristics of the used datasets.

The performance of the different algorithms is compared both in terms of generalization accuracy and size of the induced trees, measured by the number of leaves. Following the recommendations of Bouckaert [4], 10 runs of 10-fold cross-validation experiment were conducted for each dataset.

Dataset	Instances	Attributes		Classes
		Nominal	Numeric	
Glass	214	10	9	7
Tic-Tac-Toe	958	9	0	2
Multiplexer-20	500	20	0	2
XOR-10	10000	20	0	2

Table 1: Characteristics of the datasets used.

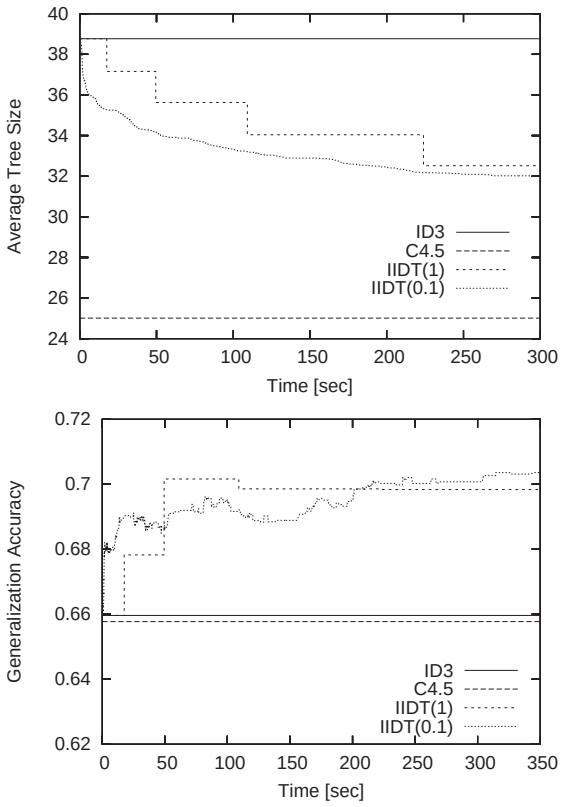

Figure 6: Anytime behavior on the Glass dataset.

Figures 6, 7, 8 and 9 show the anytime graphs for both tree size and accuracy for the 4 datasets. Each graph represents an average of 100 runs (for the 10×10 cross validation). In all cases the both anytime versions indeed exploit the additional resources and produce better trees, both in terms of size and accuracy.[2] 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 10-XOR problem. There, the tree size decreases from 4000 leaves to almost the optimal size 2^{10}, and the accuracy is increased from 50% (which is the accuracy achieved by ID3 and C4.5) to almost 100%. The shape of the graphs is typical to anytime algorithms with diminishing returns. The difference between the performance of the two anytime algorithms is interesting. IIDT(0.1) with the lower granu-

[2]Note that in some cases C4.5 produces smaller (yet less accurate) trees since it allows inconsistency with the training data by post-pruning the tree.

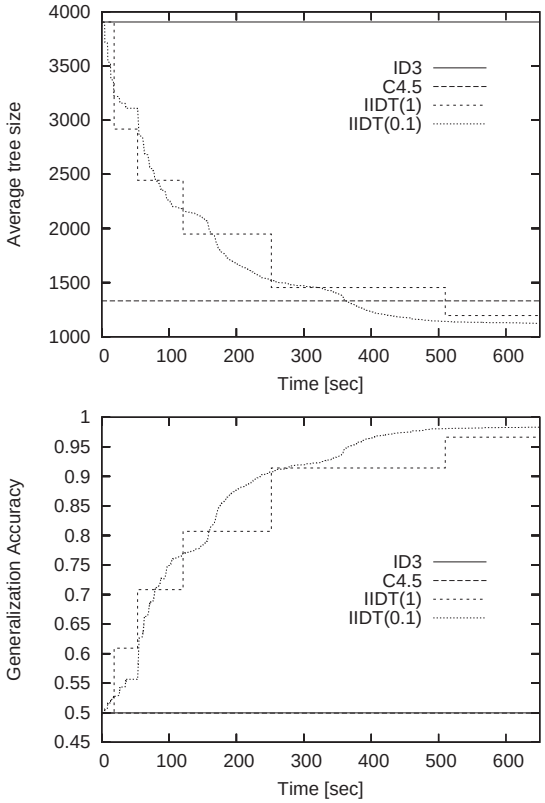

Figure 7: Anytime behavior on the 10-XOR dataset.

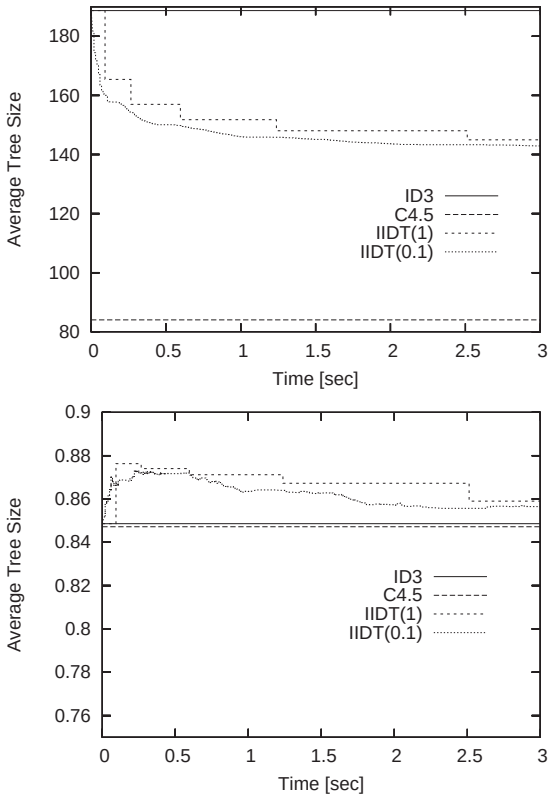

Figure 8: Anytime behavior on the Tic-Tac-Toe dataset.

larity 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 dominates the one for IIDT(1) due to its more sophisticated node selection.

The smoothness of the IIDT(0.1) graph is somehow misleading since it represents an average of 100 step graphs with steps occurring in different time points (vs. the graph for IIDT(1) where the steps are roughly at the same time points). Figure 10 shows one anytime graph (out of the 100). We can see that although the IIDT(0.1) graph is less smooth than the average, it is still much smoother than the corresponding IIDT(1) graph.

6. RELATED WORK

While, to our knowledge, no other work tried specifically to design an anytime interruptible algorithm for decision tree induction, there are several related works that need to be discussed here. Opitz introduced an anytime approach for theory refinement [20]. This approach starts by generating a knowledge-base neural network from a set of rules, and then it uses the training data and the additional time resources in an attempt to improve the resulted hypothesis.

Lizotte et al. [18] presented a model for a budgeted learning task. In their work the term budgeted learning refers to the problem of collecting a data sample under budget constraints for the total cost of the tests that can be taken. Although this notation is equivalent to anytime contract algorithms the problem dealt by Lizotte et al. is different than this faced by our anytime approach: while the first attempts to find the best way to spend a budget for collecting a sample of data, we assume that the dataset has already been obtained and address the question of how to exploit our budget to learning a better hypothesis from this data.

Pruning techniques also attempt to obtain smaller decision trees, but their goals and their search space are different. The main goal of pruning is to avoid overfitting the data. Pruning techniques are orthogonal to our approach and tackle different problems. We intend to integrate pruning phases in IIDT and thus allow handling overfitting problems.

Ensemble-based methods can also be viewed as anytime algorithms. The boosting method [27] iteratively refines the constructed ensemble by increasing the weight of misclassified instances and adding a new hypothesis learned based on the updated weights. This process can continue as long as the time allocation allows. In bagging [5] a committee of trees is formed by making bootstarp replicates of the training set and using each such replication to learn a decision tree. Additional resources can be exploited to generate larger committees. Unlike the problem we face in this work, the classifiers constructed by the boosting and bagging algorithms consist of ensembles of decision trees rather than a single tree. A major problem with ensemble-based meth-

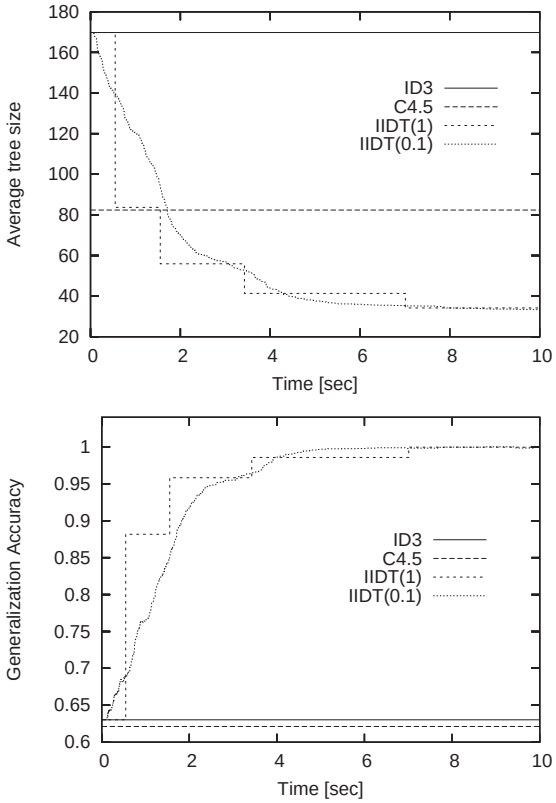

Figure 9: Anytime behavior on the 20-Multiplexer dataset.

ods is that in many cases the induced ensemble is large, complex and difficult to interpret [11]. Another problem is that when the concept to learn is hard, greedy trees are unable to discover any knowledge about the target concept and hence their combination cannot improve the performance. To experimentally test this, we examined the performance of Bagging on the XOR10 dataset. Our results indicate that the committee failed to learn the concept and performed no better than a random guesser, even for a large number of tree-members (up to 1000). In the future, we intend to empirically compare the anytime behavior of of other ensemble methods such as Boosting and Random Decision Tree [10] to IIDT, as well as examining committees of trees produced by more expensive algorithms such as LSID3.

Papagelis and Kalles [21] presented GATree, an algorithm that uses genetic algorithms to evolve decision trees. When tested on several UCI datasets, GATree was reported to produce trees as accurate as C4.5 but of significantly smaller size. GATree can be viewed as an anytime interruptible algorithm that uses additional time to produce more and more generations. We conducted several experiments with GATree, with its default parameters as reported in [21]. For this purpose we used the free GATree version available in `http://www.GATree.com`. The results indicate that the anytime behavior of GATree is problimatic. Although improvements were observed, they were not consistent and the results suffered from considerable fluctuations. In addition,

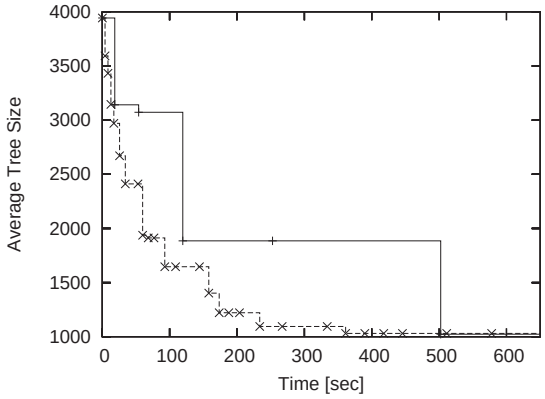

Figure 10: Time steps for an arbitrary run on 10-XOR.

we tested our IIDT on the parity concepts used to evaluate GATree. Although we could not use exactly the same datasets, we followed the same method to create them and the results show that IIDT achieved better results than those reported for GATree. For example, for the 4 attributes parity problem with 6 additional irrelevant attributes, IIDT was able to reach 99% accuracy while GATree was reported to have 85% average accuracy.

Utgoff [31] presented DMTI, an induction algorithm that uses a direct measure of tree quality instead of greedy heuristic to evaluate the possible splits. Several possible tree measure were examined and the MDL (Minimum Description Length) measure had the best performance. DMTI can use a fixed amount of additional resources and hence cannot serve as interruptible anytime algorithm. Further, DMTI uses the greedy approach to produce the lookahead trees and that might be insufficient to well-estimate the usefulness of a split.

Last et al. [17] introduced an interruptible anytime algorithm for feature selection. Their proposed method selects features by constructing an information-theoretic connectionist network, which represents interactions between the input attributes and the target class.

7. CONCLUSIONS

In this work explored the problem of how to produce better decision trees when more time resources are available. Unlike the contract setup that was addressed in a previous study, this work does not assume a priori knowledge of the amount of the resources available and allows the user to interrupt the learning phase at any moment.

The major contribution of this paper is the IIDT framework that can be adjusted to use any contract algorithm for reproducing a decision tree and any measure for choosing the subtree to rebuild. We studied an instantiation of this framework that bases the decision of what subtree to rebuild next on the expected cost and expected benefit and uses LSID3 for rebuilding subtrees.

The reported experimental study shows that IIDT exhibits

a good anytime behavior allowing a tradeoff between the cost of the learning process and the quality of the induced hypothesis. The smoothness of the performance profiles was shown to be flexibly controlled by the granularity parameter.

In the future we, intend to apply monitoring techniques for optimal scheduling of IIDT. In addition, we plan to integrate pruning phases in the IIDT framework as well as examining several different strategies for choosing nodes and improving subtrees.

8. REFERENCES

[1] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

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

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

[4] R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *ICML'03*, pages 51–58, Washington, DC, USA, 2003.

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

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

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

[8] M. W. Craven and J. W. Shavlik. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Computer Science department, University of Wisconsin, Madison, 1996.

[9] S. Esmeir and S. Markovitch. Lookahead-based algorithms for anytime induction of decision trees. In *ICML'04*, pages 257–264, 2004.

[10] W. Fan, H. Wang, P. S. Yu, and S. Ma. Is random model better? on its accuracy and efficiency. In *ICDM'03*, pages 51–58, 2003.

[11] 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.

[12] E. Gabrilovich and S. Markovitch. Text categorization with many redundant features: Using aggressive feature selection to make svms competitive with c4.5. In *ICML'04*, pages 321–328, 2004.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag, 2001.

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

[15] J. Huang, J. Lu, and C. Ling. Comparing naive bayes, decision trees, and svm using accuracy and auc. In *ICDM'03*, Melbourne, FL, 2003.

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

[17] M. Last, A. Kandel, O. Maimon, and E. Eberbach. Anytime algorithm for feature selection. In *RSCTC'01*, pages 532–539. Springer-Verlag, 2001.

[18] D. J. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive bayes classifiers. In *UAI'03*, Acapulco, Mexico, 2003.

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

[20] D. W. 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.

[21] A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. In *ICML'01*, pages 393–400, San Francisco, CA, USA, 2001.

[22] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[24] S. J. Russell and E. Wefald. Principles of metareasoning. In *KR'89*, pages 400–411, San Mateo, California, 1989.

[25] S. J. Russell and S. Zilberstein. Composing real-time systems. In *IJCAI'91*, pages 212–217, Sydney, Australia, 1991.

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

[27] R. Schapire. A brief introduction to boosting. In *IJCAI'99*, pages 1401–1406, Stockholm, Sweden, 1999.

[28] Y. Shan, E. Milios, A. Roger, C. Blouin, and E. Susko. Automatic recognition of regions of intrinsically poor multiple alignment using machine learning. In *CSB'03*, 2003.

[29] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

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

[31] 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.

[32] X. Zhu, S. Sun, S. E. Cheng, and M. Bern. Classification of protein crystallization imagery. In *EMBS'04*, San Francisco, CA, 2004.