Special Issue Paper

Applied Stochastic Models in Business and Industry

Received 15 July 2014, Revised 13 October 2014, Accepted 17 October 2014 Published online in Wiley Online Library

(wileyonlinelibrary.com) DOI: 10.1002/asmb.2086

Parallel construction of decision trees with consistently non-increasing expected number of tests

Irad Ben-Gal*† and Chavazelet Trister

In recent years, with the emergence of big data and online Internet applications, the ability to classify huge amounts of objects in a short time has become extremely important. Such a challenge can be achieved by constructing decision trees (DTs) with a low *expected number of tests* (ENT). We address this challenge by proposing the 'save favorable general optimal testing algorithm' (SF-GOTA) that guarantees, unlike conventional look-ahead DT algorithms, the construction of DTs with monotonic non-increasing ENT. The proposed algorithm has a lower complexity in comparison to conventional look-ahead algorithms. It can utilize parallel processing to reduce the execution time when needed. Several numerical studies exemplify how the proposed SF-GOTA generates efficient DTs faster than standard look-ahead algorithms, while converging to a DT with a minimum ENT. Copyright © 2014 John Wiley & Sons, Ltd.

Keywords: classification; big data; online applications; parallel computing; look-ahead algorithms; recommendation systems

1. Introduction

Decision tree (DT) models are commonly implemented in data mining and decision theory for purposes of classification. They can represent complex stochastic processes, involving various random attributes (features) in different areas. Examples for such areas include healthcare systems [1–4], geoscience [5, 6], linguistics [7], environmental studies [8, 9], manufacturing [10] and bioinformatics [11]. An example of a DT is seen in Figure 1, where nodes correspond to the features, arcs represent the different feature values and leaves correspond to the different classes (democrat or republican in this example).

Traditionally, the main performance measure of a DT was associated with its average classification accuracy. Indeed, in knowledge extraction and data mining applications, the classification accuracy is a leading quality measure, because the main purpose of these applications is to better understand the underlying stochastic process that is being studied. In recent years, however, with the emergence of big data and online Internet technologies, the ability to classify huge amounts of objects in a very short time (e.g., hundreds of thousands of users in a fraction of a second) has become extremely relevant and challenging. Some modern examples for big data applications that require fast classifications with a small *expected number of tests* (ENT) are credit scoring, customer churning, fraud detection, ad networks, online pricing and promotions (e.g., [12–15]).

This paper presents a new method for constructing efficient DTs in terms of their ENT. The ENT (a.k.a. the *average path length*) focuses on the time or the number of steps that is required for the classification task. Formally, the ENT is calculated by averaging the number of tests (arcs) an object has to follow from the tree root till it reaches a leaf [5]. Thus, if the path length to leaf i is denoted by l_i , and the probability of reaching leaf i is denoted by p_i, the ENT is calculated as $\sum_i p_i \cdot l_i$.

The effects of reaching a small ENT can be critical even in a more traditional example (let alone an online web example). Consider, for the purpose of illustration, a large hospital where DTs are used to classify new incoming patients to various medical tests and treatments [1, 3, 14, 16]. An efficient DT in this case is one that not only reaches a correct classification, but also does so with a low ENT. Now, suppose that the hospital accepts 5000 patients a month, and that the existing DT reaches an allocation with an average of 3.5 tests per patient, while each of these tests requires 4 hours on the average to be

Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel
**Correspondence to: Irad Ben-Gal, Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel.*
†E-mail: bengal@tau.a.il

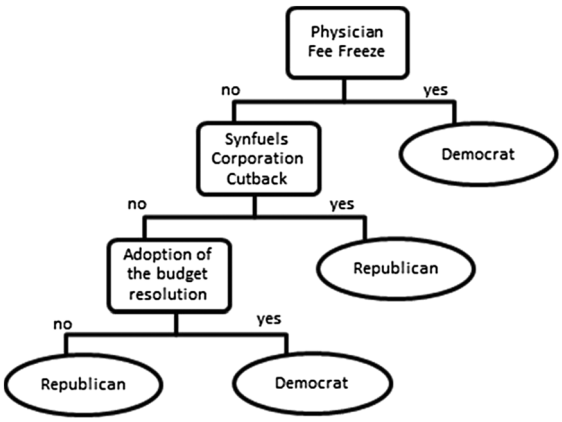

Figure 1. Decision tree based on the 'Congressional Vote' dataset, University of California, Irvine, repository.

executed. Improving the ENT from 3.5 tests per patient to 3.4 tests per patient would imply an overall reduction of 24,000 tests hours (or equivalently 1000 days) per year.

Decision trees are often constructed by a supervised learning process, based on training sets that contain records of feature values, tagged by their class. The construction of optimal DTs is known to be a nondeterministic polynomial time-complete (NP-complete) problem [17] and therefore is generally approximated by using an iterative greedy heuristic; whereby, in each node of the tree, the feature with the highest discrimination power (e.g., minimal entropy) is selected. Many of these greedy 'score-and-select' algorithms are similar and differ one from another by various selection and stopping criteria [18]. In general, this approach is not bad as it corresponds to efficient coding algorithms [10, 19–23]. However, in many cases, such a heuristic results in a greedy DT that could have been more efficient [19].

One approach that aims to reduce the greediness effects of the above heuristic is to use a 'look-ahead' scheme, that is, instead of choosing the feature (i.e., the test) for only one node at a time, it chooses the features for multiple levels of the tree simultaneously. One such original algorithm is the general optimal testing algorithm (GOTA) proposed by Hartmann *et al*., [24]. Selecting the features for a node and its offspring simultaneously would constitute a 'two-step look-ahead' procedure and so on. Clearly, as the degree of look-ahead increases, the complexity of the algorithm rises exponentially. Increasing the look-ahead to the maximum value, that is, to the number of available tests, would result in an exhaustive search leading to an optimal solution. This approach, however, is only practical for very small classification problems. Using a look-ahead that is any less than the number of available tests does not guarantee any improvement with respect to the ENT. Previous studies (e.g., [25, 26]), as well as examples in this work, show clearly that increasing the degree of look-ahead to any value that is less than the number of available tests does not guarantee a better solution than the one obtained with a lower look-ahead value. In other words, in many cases, by increasing the look-ahead steps, the algorithm spends exponentially more time looking for a good solution, but might result in a significantly worse solution. The expected tradeoff of complexity in return for a better solution does not apply in such cases.

The algorithm presented in this paper addresses the problem of obtaining worse solutions when increasing the construction complexity. The proposed algorithm, called the 'save favorable GOTA' (SF-GOTA), relies on the greedy 'one-step look-ahead' method and the known 'beam search' strategy to create multiple greedy trees, growing them out entirely before choosing the most efficient tree in terms of the ENT. The proposed method looks ahead up to a level of a *completed* DT before selecting a test; yet, it does not apply a look-ahead for all possible tests, but only for few selected tests that look promising. Such a strategy is appealing as it allows to use parallel processors to construct multiple trees, each of which separately and completely, before deciding whether the tree should be kept or discarded. The proposed approach is different from the *beam search* that can be considered as a branch and bound approach with an inadmissible pruning rule [27]. That is, the '*beam search*' saves only B solutions on each level of the tree and disregards the rest. This implies that a solution retained on a higher level of the tree could be discarded without being fully developed. This also implies that there is a dependency between the different branches of the tree, and therefore parallel processing is less evident in this case. Similarly, the proposed approach is different from the *random forest* approach, because the selection of features is based on their performance rather than a random selection and a majority-vote rule [28].

Experimental results show that the proposed algorithm produces solutions in a fraction of the time required by conventional look-ahead algorithms and provides a steady (monotonic) improvement in the ENT measure as the number of multiple trees is increased. Using public known datasets, the SF-GOTA reached the DTs with the lowest ENT in less than 1% of the time required by GOTA and its popular iterative dichotomiser 3 (ID3) equivalent, when using a look-ahead of the same degree. These phenomena are further discussed in the paper.

The rest of this paper is organized as follows. Section 2 provides a literature review while discussing some of the existing gaps. Section 3 presents the proposed SF-GOTA. Section 4 includes experimental results, a simulation study and a related discussion. Section 5 concludes the paper.

2. Literature review

The literature review begins with some generic and early methods for the construction of DTs. It continues by discussing some construction algorithms that are based on information theoretical measures, including the 'look-ahead' scheme, such as the one proposed here.

Hyafil and Rivest [17] showed that the construction of an optimal DT is an NP-complete problem, thereby often requires the development of heuristics aimed at minimizing the ENT. Rissanen [29] used the Kullback–Leibler divergence to maximize the information gain in the construction of efficient DTs. His approach has been adopted by many DT construction algorithms. Hartmann *et al*. [24] expanded this approach and developed GOTA for the construction of a DT. GOTA is less cited in comparison to some later DT algorithms, such as the ID3 and C4.5, although it was published earlier and in many ways proposed a similar DT construction, as well as a solid theoretical view. Hartmann *et al*. did not specify one certain method for scoring the different attributes, but suggested different scoring methods, depending on the user's objective. The general method consisted of measuring the information gain of a given attribute and normalizing it over some cost functions of the tree. An elaborated description of the algorithm is given in Section 3. Ohta and Kanaya [30] suggested a branch and bound approach to optimally construct a DT, yet the computational cost of their methods is high.

The rise of data mining promoted DTs as a practical prediction and classification tool. Quinlan's ID3 and C4.5 algorithms [21, 22, 31] utilized information gain principles and became extremely popular. The ID3 algorithm [21] selects attributes based on their information gain (reduction in entropy as a result of selecting the test in the node). The C4.5 algorithm [22] selects attributes based on a normalized information-gain ratio. The classification and regression tree (CART) algorithm of Breiman *et al*. [32, 33] uses a 'purity measure' that evaluates the uniformity of the data per different selections. These algorithms aim at minimizing the classification error, yet they use an information theoretical approach that might support an efficient construction of DTs also with respect to the ENT [20]. The simulation study in Section 4 shows that, in many cases, the obtained DTs are not ENT-efficient as expected.

As noted above, one conventional approach against the greediness effects of DTs is to incorporate some type of *look-ahead* algorithm: instead of constructing one level of the tree at each iteration, the algorithm constructs two or more levels of the tree simultaneously. This approach was implemented by Hartmann *et al*. [24] with GOTA and was suggested as an improvement feature for the ID3 by Esmeir and Markovitch [34]. The main drawback of such a solution is the exponential increase in complexity as the level *look-ahead* grows. The authors recognized this limit and suggested a time-restricted algorithm that returns the best tree within a given time frame.

Another direction was suggested by Yang *et al*. [35], who recommended reducing the number of available tests for each node by pre-selecting the most promising tests. Breiman [28] suggested the random forest method: an ensemble of unpruned classification (or regression) trees created by using bootstrap samples of the training data and random feature selection in tree induction. Prediction is made by aggregating (majority vote or averaging) the predictions of the ensemble. The random forest model aims at minimizing the classification error and, as an ensemble, is less applicable for minimizing the ENT, as considered here. Page and Ray [36] suggested an alternative method to the look-ahead, using skewing techniques to achieve the advantages of a look-ahead function with a constant increase in the complexity. Although this method seems very promising from a complexity standpoint, it is only applicable to datasets with no more than six to seven attributes.

Murthy [25] conducted a number of experiments using C4.5 and CART and tested the difference between the expected depth of the greedy DT and the expected depth of the optimal DT. He found that using a look-ahead method in the DT construction algorithm often resulted in a worse DT than when using a completely greedy algorithm. This result coincides with his earlier publication with Salzburg [26], whereby a look-ahead function was found to be suboptimal with respect to both the accuracy and the tree size.

Elomaa and Malinen [37] found that the $k-$step look-ahead approach (k denoting the number of levels set simultaneously) is not always beneficial from an accuracy standpoint. As an alternative approach, they suggested a splitting rule that combines the $k-$step look-ahead together with a voting process. Norton [38] suggested the IDX algorithm, which works with an overlapping look-ahead algorithm. The IDX controls the complexity of the algorithm by considering the classification costs; however, under various circumstances, additional look-ahead becomes less and less warranted. Ben-Gal *et al*. [19] proposed the dual information distance method for efficient construction of DTs that are computationally attractive, yet relatively robust to noise. The dual information distance heuristic selects features by considering both their immediate contribution to the classification task as well as their future potential effects in terms of orthogonality with respect to the selected features.

Lowerre (1976) proposed the 'beam search' algorithm for a speech recognition algorithm known as the hypernyms and alignment of relational paraphrases [2], which has been adapted later for many different uses. The 'beam search' approach is a form of branch and bound with inadmissible pruning [27]. Specifically, instead of pruning branches that are bounded by a worse solution, there is a beam of width B that defines the number of solutions that are investigated at each level. If the width B is constant, this prescribes the number of nodes to be saved (all others are eliminated) and implies a linear complexity in the number of tree levels. The implication is that a solution that may have seemed favorable on a previous level could be discarded because the offspring of another solution are found to be superior. This is called 'inadmissible pruning' because there is no assurance that the saved solutions are indeed better than the discarded ones.

Finally, let us note that many of the above-described algorithms are recursive in a way that prevents efficient distribution of the tree construction to parallel processors. Parallel processing schemes works best when each processor can be executed independently of all others [39], as satisfied by the SF-GOTA for the development of ENT-efficient DTs.

3. The 'save favorable general optimal testing algorithm'

This section describes the proposed SF-GOTA algorithm. For the purpose of completeness and because SF-GOTA relies on the GOTA algorithm [24], the latter is introduced as well.

3.1. The general optimal testing algorithm (GOTA) [24]

The GOTA algorithm aims at minimizing the ENT. Like other top–down DT construction algorithms (e.g., ID3, C4.5 and CART), the GOTA scores the available tests (we use the terms 'test', 'attribute' and 'feature' interchangeably) as candidates for each node. Every test is scored by the information gain it obtains in that node and weighted by the ENT that reaches that node. The information gain measures the reduction of entropy per test (in bit/test units) obtained by applying the test to further partition the records in the node. In particular, denote the distribution of records at a node before (or after) applying a test, or a series of tests, by the random variable X (or $X|T$ respectively). The information gain is then defined by $IG = H(X) - H(X|T)$ where $H(\cdot)$ is the well-known entropy function while the score is computed as $Score = \frac{H(X)-H(X|T)}{w(X,X|T)}$, where $w(X, X|T)$ is the weighted number of tests required to reach from distribution X to the conditional distribution $X|T$.

The above score expresses the 'learning rate' in bits/test units, measuring how much information can be gained on the average per test. At each stage of the construction of the DT, the algorithm chooses those attributes that provide the highest learning rate at each node. The basic form of the algorithm selects only one test at a time and, therefore, the denominator of the measure is simply the probability of a record to reach that node in the DT—a probability that is independent of the attribute chosen in the node and results in a very similar algorithm to that of the ID3 [31], with slight changes related to the trimming of the tree, as seen in the pseudocode in Figure 2. Hartmann *et al*. [24] suggested to add a look-ahead function, by which tests are allocated to several nodes simultaneously by considering more than one level in the DT at each construction stage.

For example, if one applies a two-step look-ahead (i.e., by selecting two test levels simultaneously), the denominator is the sum of the probability of reaching both the first unclassified node as well as the probabilities of reaching each of that node's unclassified offspring. Some attributes may fully classify some of the records and, therefore, the denominator may change for different attributes. In general, the number of test-allocation combinations in a k-step look-ahead GOTA is

$$\sum_{i=0}^{\left\lceil \frac{n}{k} \right\rceil} \left(2^{i \cdot k} \prod_{j=0}^{k-1} (n - (i \cdot k + j))^{(2^j)} \right), \tag{1}$$

where n is the number of available features and k is the number of the look-ahead levels. Note that the product terms in Equation (1) appear within the summation term, thus affecting significantly the computational complexity of the proposed algorithm.

1. Define look ahead value k, $d = k$
2. Calculate entropy at node
3. If $entropy > 0$ Calculate information gain of all available tests else end
   - 3.1. While $d > 1$ calculate the $remaining\ entropy$ in left and right sons for each test
     - 3.1.1. For each of {left son, right son}, if $remaining\ entropy > 0$
       - 3.1.1.1. Calculate Information gain of all remaining tests for son
       - 3.1.1.2. $d = d - 1$ and go to 3.1
     - 3.1.2. Update weighted information gain (wrt test probability) and sum of probabilities
   - 3.2. Calculate F measure: divide accumulated information gain by sum of probabilities
   - 3.3. Select solution with the highest F measure (refers to learning rate).
   - 3.4. Continue to level $current + k$
   - 3.5. For every node on level $current + k$
     - 3.5.1. Go to 2.

Figure 2. The general optimal testing algorithm (GOTA) pseudocode [24].

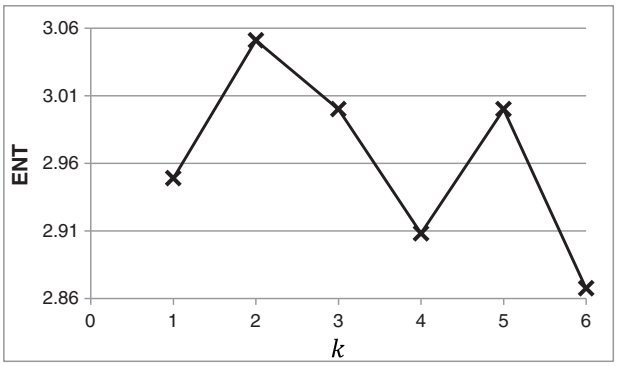

Figure 3. Expected number of tests (ENT) as a function of the look-ahead size k dataset for dataset 'Zoo' taken from the University of California, Irvine, repository.

In general, the GOTA performs well for small values of k, yet its complexity increases fast in k. Moreover, we show that increasing k does not guarantee that a better solution is obtained, and counterintuitively, often higher-ENT trees are obtained by larger k values. This phenomenon is seen in Figure 3 that is based on the 'Zoo'[‡] dataset of the University of California, Irvine (UCI) repository [40]. A DT is constructed for this dataset using the GOTA algorithm [24] with varying values of the look-ahead factor $k = 1, \ldots, 6$.

Although one might expect the ENT to steadily decrease as the look-ahead factor is increased, it is seen in Figure 3 that such a case does not apply here. The ENT starts at around 2.95 bits, and then rises to around 3.05 bits when a look-ahead of two levels is used. Increasing the look-ahead to three or four levels results in an improvement in the ENT, but when the look-ahead is further increased to five, once again there is a steep rise in the ENT. In this study, we found that such an anomalous behavior is not rare: around 80% of the datasets that were analyzed from the UCI repository exhibited such a non-monotonic behavior (without a noticed difference between large datasets and small datasets). Note again that such phenomenon becomes challenging because an increase in k leads to an exponential growth in the processing time. This fact might discourage practitioners from investing more time looking for a better DT solution by increasing the look-ahead factor k.

The non-monotonic behavior of the ENT in the look-ahead factor might seem counterintuitive at first glance, but actually it makes sense. Once the decision has been made to allocate certain features (tests) at certain nodes, the GOTA algorithm provides no options of backtracking and correcting the decision if the solution yields a high ENT. Thus, given the first allocations, all that is being left is to try and improve the ENT with the top of the tree being fixed already. Considering the set of all the tree constructions, one can think that any allocation of a test in the tree fixes the search to a subset of such constructions. Using a look-ahead function fixes the search to different subsets, and the search can no longer access previous 'promising' solutions from these subsets to other ones.

The SF-GOTA presented next aims to address some the above challenge.

3.2. SF-GOTA

The proposed algorithm, SF-GOTA, generates a different look-ahead scheme. The algorithm is similar to the ID3 or to the GOTA with $k = 1$, but instead of choosing only the best test to place at the specified node, it chooses the s best features/tests, where s is a user-defined parameter (called also the 'subsets parameter') that depends on the available computing power (e.g., number of parallel processors) or selected arbitrarily by the user, in a manner which is similar to the selection of the k parameter in GOTA. These s solutions are used to create s different trees, each of which begins with one of the s best tests. Proceeding to the next level of the tree, instead of choosing the best test to put in each of the offspring, the algorithm chooses the $s - 1$ best tests, and creates a tree for each such selection. Assuming a binary decision tree, at this point there are $s \cdot (s-1)^2$ possible trees. This process continues on for each level of the tree until $s = 1$, at which point only the best test is used to further grow the tree, while no more trees are generated. The tree is then grown out to its leaves with $s = 1$. Once all trees are constructed, the most efficient tree is selected as seen in the pseudocode in Figure 4.

The SF-GOTA is based upon a number of very simple insights. First, the root node of the tree is one of the most important nodes. A 'wrong' allocation of test/feature at the root node is probabilistically more costly than 'wrong' allocations lower down in the tree. The reason is that all the records must undergo the root node test, as opposed to lower tests in the tree

[‡] *One attribute that is not binary in 'Zoo' is the 'number of legs' attribute that was modified to the binary attribute 'four legs or more'.*

1. Choose s value according to available parallel processors, or arbitrarily
2. Calculate entropy of the node
3. Calculate information gain (IG) for all available tests.
4. Choose s most favorable tests
   4.1. For each test chosen
      4.1.1. Update $s = \max(s-1,1)$
      4.1.2. Partition data according to test chosen
      4.1.3. Calculate accumulated expected number of tests
      4.1.4. For each partition that is not fully classified, recourse to step 2
5. Choose the test with the shortest accumulated expected number of tests.

Figure 4. The save-favorable general optimal testing algorithm (SF-GOTA) pseudocode.

Table I. Running example.

Symp	U_1	U_2	U_3	U_4	U_5	U_6	Information gain
Prob.	0.2	0.1	0.05	0.2	0.3	0.15	
Class	A_1	A_1	A_2	A_3	A_4	A_3	
$\mathbf{T_1}$	1	0	0	0	1	0	0.7245
$\mathbf{T_2}$	1	0	0	1	0	1	0.7173
$\mathbf{T_3}$	0	0	0	1	0	1	0.9341
$\mathbf{T_4}$	1	1	1	0	1	0	0.9341
$\mathbf{T_5}$	0	1	1	0	0	1	0.2610
$\mathbf{T_6}$	0	1	0	1	1	0	0.3506

where only part of the records will undergo the test. Therefore, in the lower nodes of the tree the effect on the ENT value is more local. In addition, solutions obtained by different look-ahead procedures often contain those attributes with the highest information gain. These are the most promising attributes, and it would be prudent to pursue them further. A third important point regards the inefficiency of the existing look-ahead methods—a major problem with look-ahead algorithms is that they select the tests for k steps, set them and continue building the tree based on those k steps. There is no option of backtracking and correcting wrong test selection in these k steps. The SF-GOTA keeps the different solutions and enable to abandon inefficient solutions at later stages. Figure 4 presents the pseudocode of the SF-GOTA algorithm.

At first glance, the proposed algorithm might seem to be just as complex as the original GOTA algorithm with k-steps look-ahead. Yet, note that the number of construction combinations analyzed with SF-GOTA is

$$\prod_{j=0}^{s-2}(s-j)\cdot 2^j \cdot \sum_{i=1}^{n}(n-i+1)\cdot 2^{i-1}, \tag{2}$$

where n denotes the number of available features. The above expression contains two separate terms: $\prod_{j=0}^{s-2}(s-j)\cdot 2^j$ is the number of trees generated by the SF-GOTA, and $\sum_{i=1}^{n}(n-i+1)\cdot 2^{i-1}$ is the cost of developing one GOTA tree with $k=1$ (the simplest and cheapest possibility). Moreover, note that now the sum-term and the product-term are separated unlike Equation (1) . This is important because it implies that there are no dependencies between the different trees that are constructed by the algorithm. As a result, the algorithm can be easily programmed for parallel processing, simply by separating the construction of different trees to separate processors [39]. By using parallel processing, the execution time of the algorithm can be reduced further, compared to the conventional GOTA algorithm. In theory, with enough parallel processors, the number of combinations can decrease to $O(2^n)$. For example, if for a certain problem, 5 hours are required to run SF-GOTA with $s=6$, and one wishes to increase s to 7, one could divide the work among seven processors, and keep the same running time, while exploring 164,413,440 more DT combinations. If one aims to increase s to 8, he can divide the work between $7\cdot 8 = 56$ processors, and still maintain the same running time while exploring 84,556,431,360 more DT combinations than with $s=6$. The only limit is the number of processors available for use and, therefore, grid computing technology can fit well such purpose. More information on computation time is given in a later section.

3.3. An SF-GOTA running example

To further illustrate the proposed scheme, this subsection provides a running example. Let us start with the example in Table I.

The dataset includes six records: $U_1,\ldots,U_6$. Each of these records has some *a priori* probability of appearing. The probabilities can be uniformly distributed or otherwise depending on available *a priori* information. Each record can be

Applied Stochastic Models in Business and Industry

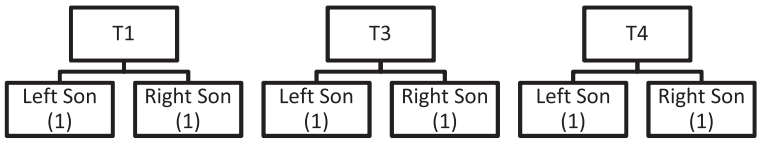

Figure 5. First level of decision tree.

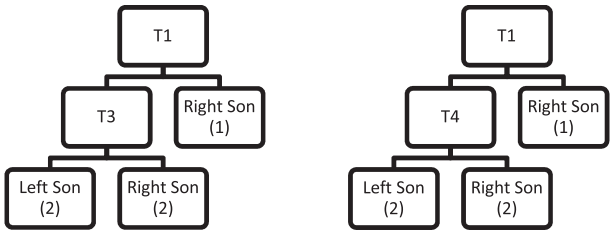

Figure 6. Second level of decision tree, left offspring.

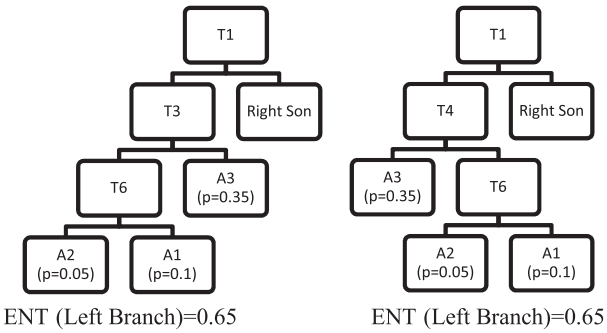

Figure 7. Left branch of root node fully grown. ENT, expected number of tests.

classified to one out of four classes: $A_1, \ldots, A_4$. The entropy of the entire problem is calculated according to the probabilities of each of the classes: $A_1, \ldots, A_4$. The last column shows the information gain of each test.

It should be noted that both tests $\mathbf{T_3}$ and $\mathbf{T_4}$ have the same information gain of 0.9341 bits. Simple examination of the attribute columns reveals that tests $\mathbf{T_3}$ and $\mathbf{T_4}$ are, in fact, identical (with opposite symbols) and produce the same exact partition: every entry where test $\mathbf{T_3}$ obtains a '0' sign, $\mathbf{T_4}$ obtains a '1', and vice versa. Thus, the selection between them should be arbitrary.

3.3.1. Stage 1 – choosing the test for the tree root. Running this problem with SF-GOTA with a subset parameter $s = \mathbf{3}$ implies that one would select the three best tests at this stage and create a separate tree for each, as depicted in Figure 5.

3.3.2. Stage 2 – choosing the test for left offspring on the second level. For each of the trees, one would continue on to the next level and then choose the $s - 1 = 3 - 1 = 2$ best attributes. For the tree beginning with $\mathbf{T_1}$, the information gain is calculated for each of the tests $\mathbf{T_2}, \ldots, \mathbf{T_6}$ for the left offspring. The best tests are found to be $\mathbf{T_3}$ and $\mathbf{T_4}$. Accordingly, two trees are constructed, one with t_3 and the other with t_4 as left offsprings of the t_1 node, as shown in Figure 6.

3.3.3. Stage 3 – building the rest of the tree. Continuing on to the next level of the tree, only the best attributes must be kept so no more trees are created, while the existing trees are grown up to their leaves using the one-step look-ahead method.

Figure 7 shows the two full grown left side treves that begin with $\mathbf{T_1}$ in the root node.

3.3.4. Stage 4 – choose the best sub-tree. Both trees result in the same ENT. Therefore, either one can be selected arbitrarily.

3.3.5. Stage 5 – choosing the test for right offspring in the second level. Figure 8 shows the same process carried out on the right offspring of the root node.

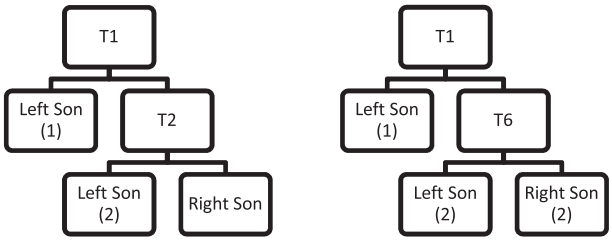

Figure 8. Second level of decision tree, right offspring.

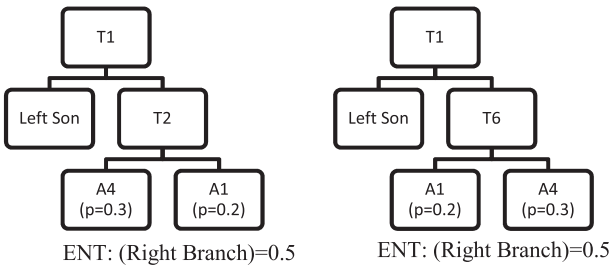

Figure 9. Right branch of root node fully grown. ENT: expected number of tests.

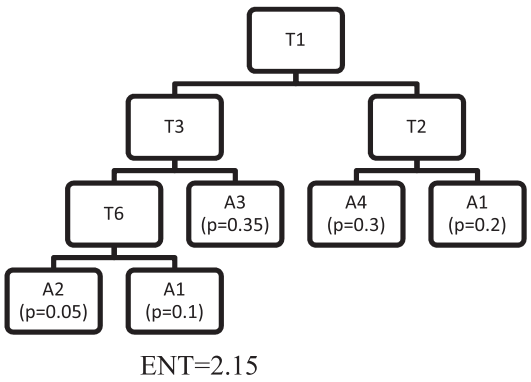

Figure 10. Full decision tree for root node T1. ENT: expected number of tests.

The two best attributes for the right offspring are either $\mathbf{T_2}$ or $\mathbf{T_6}$. A tree is created for each option.

3.3.6. Stage 6 – building the rest of the tree. Continuing on to the next level, only the best attribute needs to be kept, so at this point, the tree is grown up to its leaves, as seen in Figure 9.

As there is no difference between the two possibilities, either one can be selected.

Note that there is no dependency between the left offspring and the right offspring of the root node, and that each is solved with no consideration given to any other branch in the tree.

Figure 10 shows the final DT as the combination of the best left branch and best right branch that were found so far.

The same process is repeated for the case where $\mathbf{T_3}$ and $\mathbf{T_4}$ are placed at the root node.

3.3.7. Stage 7 – choosing the best complete tree. Returning to the first level of the tree, there are now three possible trees from which one has to select the best one, as seen in Figure 11.

Of these three possible trees, the tree with the lowest ENT is chosen (in this case with 2.15 bits), as the one with $\mathbf{T_1}$ in the root node. It is important to note that algorithms such as ID3, C4.5 and the GOTA with no look-ahead would result in either of the two right-hand trees (since $\mathbf{T_3}$ and $\mathbf{T_4}$ have higher information gain, as seen in Table I). Using a look-ahead procedure, they would all reach the optimal left-hand solution with a higher ENT. This toy example is presented only to describe the process of SF-GOTA and is not one where SF-GOTA is necessarily the fastest algorithm.

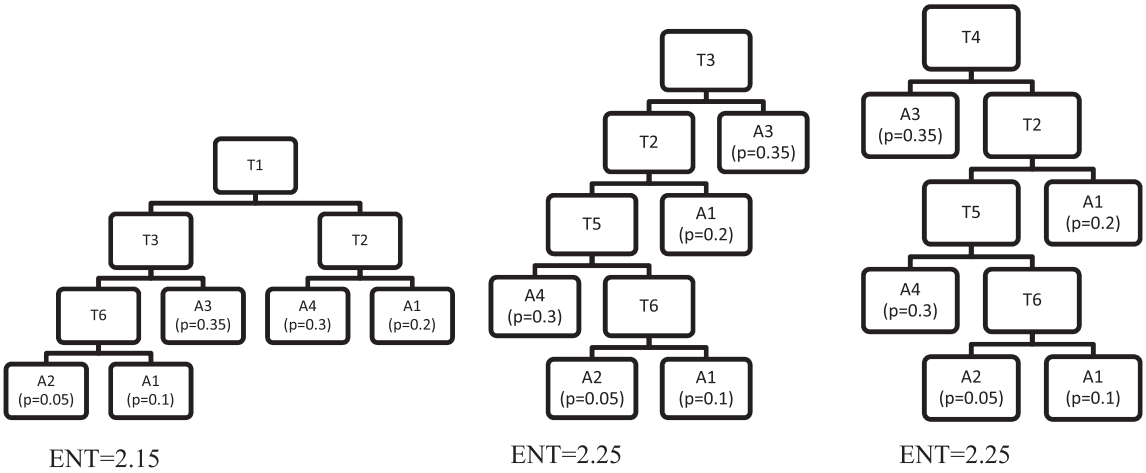

Figure 11. Chosen decision trees for each possible tree-root possibility. ENT, expected number of tests.

It should be further noted that SF-GOTA with $s = 1$ is identical to GOTA with $k = 1$, as both algorithms choose only the best test at each node and therefore produce identical decision trees. Also, both SF-GOTA and GOTA with $s = k = n$ (n- number of tests available in the dataset) will produce an optimal decision tree (i.e., by following an exhaustive search).

An extension to the proposed algorithm for cases where no particular test provides any positive information gain, but only a combination of tests provides non-zero information gain, is given in the appendix (termed SF-GOTA+).

4. Experimental results

The SF-GOTA was implemented on all binary datasets available from the UCI repository, as well as some categorical datasets that could be easily transformed into binary datasets. Two more artificial datasets were created in order to test the performance of the SF-GOTA on large datasets. The obtained results were compared to GOTA with varying degrees of look-ahead size (k).

SF-GOTA's strength is in its lower complexity, while guaranteeing to obtain a solution that is *at least as good* (monotonically non-increasing ENT) as a previous solution with a lower value of the subsets parameter s. This is because all previous solutions are retained, whereas the set of solutions obtained with a larger s value necessarily includes the set of previous solutions. Accordingly, it would seem relevant to measure SF-GOTA by its average improvement rate. The improvement rate is calculated by the number of bits saved divided by the additional time/steps that are required to reach that solution. Such a measurement of improvement rate was calculated in two separate ways:

(i) Improvement in the ENT measure relative to the best solution obtained with k or s parameters that are one less than the current value.
(ii) Improvement in the ENT measure relative to the best solution obtained with k or s equal to 1, as in the GOTA algorithm (or other greedy solutions like the ID3).

Each of these measures has its merits. GOTA's problem of producing inferior solutions while increasing k would have a greater effect on the average improvement rate if calculated according to the first measure. This would result in obtaining more negative values than if compared to the initial solution, as proposed in the second measure. On the other hand, GOTA's exponential complexity implies that the denominator in the improvement rate would be considerably larger according to the second measure than by the first measure, resulting in a smaller improvement. This is, of course, the consideration also with respect to the SF-GOTA, as its complexity is also exponential. Yet, the GOTA's rise in complexity is much steeper than SF-GOTA's (see Equations (1) and (2)), and therefore would be more affected by the fast rising denominator.

For each dataset, a time limit was specified, within which the average improvement rate was measured for both algorithms. This is necessary because as the allotted time is increased, it is more likely for both algorithms to reach the optimal solution. Once the optimal solution is reached, there can be no further improvement, and therefore any additional testing would eventually bring the average improvement rate down to zero. The time limit was different for the various datasets so as to reflect the different amount of time necessary to reach a solution for different-sized datasets.

Table II presents the average improvement rate of the 10 datasets for GOTA and SF-GOTA. Bold-faced entries indicate the superior algorithm for each dataset with regards to the improvement rate. The 2nd and 3rd columns calculate the

Table II. Average improvement rate.

Average improvement Rate $\left[\frac{\Delta ENT}{\Delta time}\left[\frac{bits}{second}\right]\right]$	Compared with previous measurement		Compared with initial measurement		Dataset specs (after preprocessing)
	GOTA	SF-GOTA/SF-GOTA+	GOTA	SF-GOTA/SF-GOTA+	[records, attributes, classes]
Zoo	−7.46E-02	**5.25E − 02**	−7.94E-02	**6.23E − 02**	[54, 14, 7]
MOFN 3-7-10	−1.84E-04	**2.14E − 04**	−2.18E-04	**2.24E − 04**	[1024, 10, 2]
Vote	9.74E-04	**7.30E − 04**	1.03E-03	**7.48E − 04**	[283, 16, 2]
Chess[2]	2.20E-05	**1.30E − 04**	2.20E-05	**1.23E − 04**	[3050, 35, 2]
Nursery[3]	**3.80E − 06**	6.01E-08	**4.18E − 06**	6.01E-08	[7508, 15, 2]
Car	**5.06E − 03**	6.86E-03	**5.64E − 03**	6.68E-03	[1728, 12, 2]
Flare	**1.41E − 02**	2.60E-03	**1.52E − 02**	2.79E-03	[169, 13, 2]
Lymphography	9.40E-03	**1.45E − 02**	9.86E-03	**1.52E − 02**	[148, 29, 2]
Test1	3.83E-03	**1.98E − 02**	4.19E-03	**2.40E − 02**	[1500, 20, 4]
Test2	8.13E-03	**1.05E − 01**	8.74E-03	**1.31E − 01**	[1500, 25, 7]

[2] Solved with SF-GOTA+ see Appendix
[3] Solved with SF-GOTA+ see Appendix

improvement rate according to the first measure, that is, the average is calculated by comparing the solutions obtained with two sequential values of either k or s. The 4th and 5th columns calculate the improvement rate according to the second measure, that is, the average is calculated by comparing every solution to the initial solution obtained by using GOTA or SF-GOTA with a parameter (the look-ahead parameter or the subset parameter, respectively) equal to 1. Note that for datasets 'Chess' and 'Nursery', the second measure calculates the improvement rate for GOTA by comparing it to the solution obtained with $k = 2$. This is because both these datasets could not be solved with GOTA where $k = 1$ (see Appendix for more details).

Table II shows that although the improvement rates are different between the two calculation measures, the qualitative analysis is similar. In all cases where the SF-GOTA had the higher average improvement rate by the first measure, it also obtained the higher average improvement rate by the second measure and vice versa. In seven out of 10 of the datasets that were analyzed, SF-GOTA outperformed GOTA with respect to the average improvement rate using both calculation measures.

As seen, the rates are understandably very small. The improvement between two algorithm executions with different parameters is generally not large, with the largest differences being around 1 bit. The additional time (in seconds) required to achieve that improvement is exponential, and, except for some special cases, greater than the improvement in the ENT by a few orders of magnitude. Note also that these are the rates and not the absolute improvements. These rates, although seemingly marginal, can be translated into considerable improvements in the ENT, which in turn translates into hundreds or even thousands of hours of saved time when utilizing the constructed DT on bid dataset, as shown in the hospital example in the introduction, not to mention big data applications where the saving potential is even higher. It is interesting to observe the improvement rates for GOTA in certain cases. The average improvement rates for the datasets 'Zoo' and 'MOFN 3-7-10' are negative. This is another indication of the fact that increasing the look-ahead parameter k does not assure obtaining a superior solution.

The following graphs (Figures 12–14) depict the ENT for three different datasets when solved with GOTA and with SF-GOTA for different values of s and k. The horizontal axis of the graphs show the time necessary to obtain the solution in seconds for a conventional 4 cores computer (on a logarithmic scale). All three of these figures show the unintuitive behavior described in the introduction, whereby increasing the look-ahead parameter increased the ENT for GOTA. Such a behavior does not exist in SF-GOTA, which monotonically improves (or stays constant) in time. In addition, the time between consecutive solutions given by SF-GOTA is much shorter than the time between consecutive solutions given by GOTA. In addition, although this is not generally guaranteed, in all three of these cases, SF-GOTA reached the lowest (not necessarily optimal) ENT in less time than GOTA.

Note that there are also cases where SF-GOTA is not as obviously superior. Figures 15 and 16 show two such datasets, 'Flare' and 'Vote', where GOTA obtains better results than SF-GOTA for quite a long time period. On the other hand, in both figures, GOTA has at least one point where the ENT suddenly rises when increasing the look-ahead. In the 'Vote' dataset, this creates a situation where at the point of the greatest look-ahead for GOTA, the solution is inferior to the current SF-GOTA solution. This does not happen in the 'Flare' dataset, and although there is a rise in the ENT with GOTA, it is still better than the current SF-GOTA solution at that stage.

Applied Stochastic Models in Business and Industry

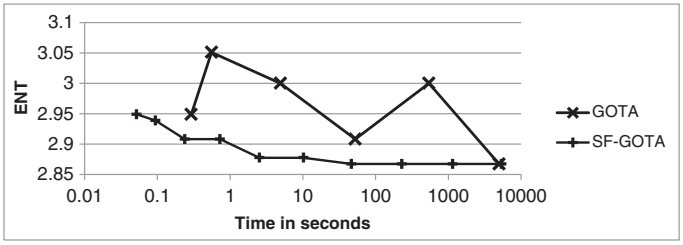

Figure 12. Dataset 'Zoo' expected number of tests (ENT) over time. GOTA, general optimal testing algorithm; SF-GOTA, save favorable GOTA.

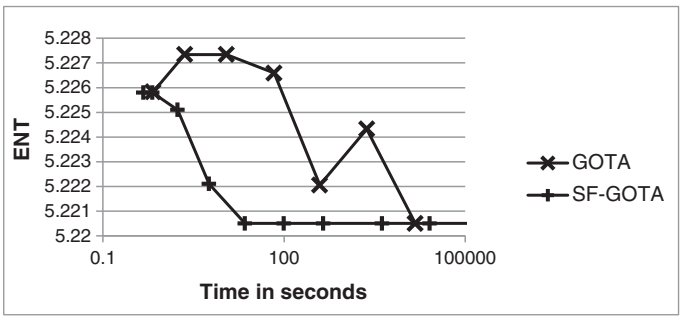

Figure 13. Dataset 'MOFN 3-7-10' expected number of tests (ENT) over time. GOTA, general optimal testing algorithm; SF-GOTA, save favorable GOTA.

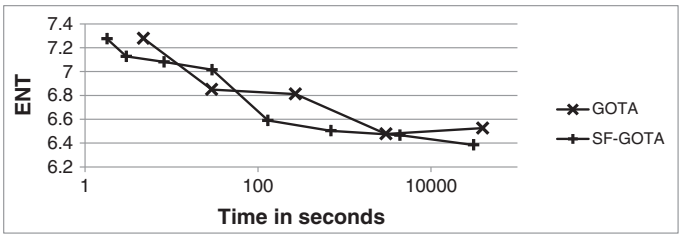

Figure 14. Dataset 'Test1' expected number of tests (ENT) over time. GOTA, general optimal testing algorithm; SF-GOTA, save favorable GOTA.

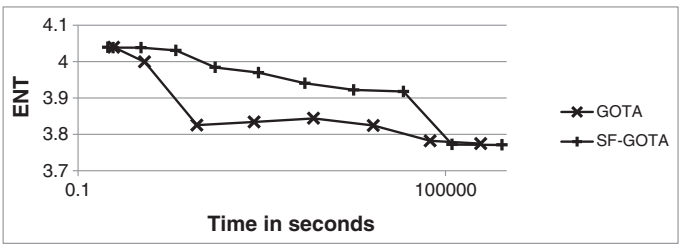

Figure 15. Dataset 'Flare' expected number of tests (ENT) over time. GOTA, general optimal testing algorithm; SF-GOTA, save favorable GOTA.

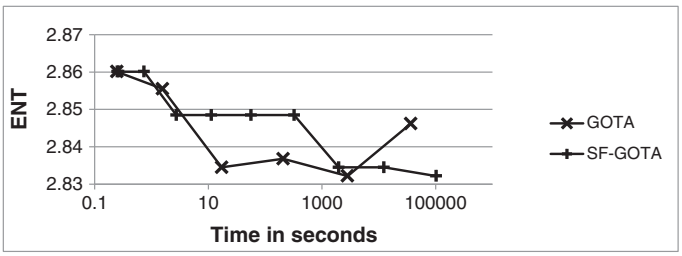

Figure 16. Dataset 'Vote' expected number of tests (ENT) over time. GOTA, general optimal testing algorithm; SF-GOTA, save favorable GOTA.

Table III. Simulation study, SF-GOTA ENT improvement when increasing s.

	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=6$
Average ENT	2.18776	2.11786	2.06953	2.04563	2.03104	2.02139
Average improvement (in bits) wrt previous s value		0.0699	0.04833	0.0239	0.01459	0.00965
Average improvement (percent) wrt previous s value		42.01%	29.05%	14.37%	8.76%	5.8%

SF-GOTA, save favorable general optimal testing algorithm; ENT, expected number of tests.

One conclusion in this case is that GOTA has great potential for big improvements, and can make considerable jumps in the ENT. The problem is that these jumps can be in either direction. A risk-seeking user may be willing to risk an inferior solution for the chance of obtaining a greatly superior solution. A risk-averse user would be safer with SF-GOTA, by which investing a greater amount of time (with higher values of s) cannot result in inferior solutions.

4.1. Simulation study

A reasonable question at this stage should refer to the optimal value of s when implementing SF-GOTA. Unlike standard look-ahead algorithms that are used for DT construction, the tradeoff between complexity and optimality clearly exists in this case. Increasing the value of s produces a better or an equivalent solution (but definitely not a worse one), yet the price, time-wise, may be formidable. Therefore, one must choose a high enough value of s to improve the solution, but not so high as to make it impractical. There is no single answer that can be applied for all datasets. Some datasets require a very high value of s in order to derive a benefit, whereas for others, a relatively low value of s is sufficient. In order to study this phenomenon, a simulated study was executed, in which a large number of random datasets were generated and solved by the SF-GOTA with different values of s. The results indicate what value of s provides the highest improvement on the average for the simulated datasets.

The simulated study includes 100,000 randomly generated datasets. Each dataset contained 10 symptoms and 6 attributes. The probabilities of each symptom were also randomly generated from a uniform distribution. Each dataset was solved by SF-GOTA with s values ranging from 1 (greedy solution, identical to GOTA with $k=1$) up to 6 (obtaining the optimal solution for these cases by an exhaustive search). Each solution was obtained under a second (including the optimal solutions with $s=6$).

The first interesting observation is the number of cases where the greedy solution was not optimal. The simulation shows that 56.79% of the datasets did not achieve the optimal DT when using a greedy algorithm. Of those 56.79%, the average difference between the greedy solution and the optimal solution was 0.293 bits, which constitutes an improvement of 11.77%. This is a rather large difference, considering that the average optimal ENT of all 100,000 datasets was around 2.02 bits. For all 100,000 datasets, those that were solved to optimality with the greedy algorithm and those that were not, the average improvement was of 0.16637 bits (7.6% improvement).

The next point of interest was the average improvement achieved when increasing s from 1 to 2, then from 2 to 3 and so on.

Table III shows the average ENT (over all 100,000 datasets) and the average improvement in the average ENT when increasing the value of s. The largest improvement, as can be seen, is when increasing s from 1 to 2. This supports the intuitive insight mentioned in Section 3 that the most influential allocations in the tree are at the root, and that it is worth allowing for more maneuverability higher up in the tree.

Another interesting point regards the time necessary to construct an optimal DT. The number of combinations tested to find the optimal DT when solved with GOTA was $\prod_{j=0}^{n-1}(n-j)^{2^j}$. For a dataset with six attributes, this requires testing $1.65 \cdot 10^{13}$ different DTs. If one were to assume that a processor can carry out 10^9 calculations per second, this should take approximately 20 min; yet, in the simulation study, the optimal solution was calculated in under a second by using the SF-GOTA. Again, this is an expected performance time in many big data applications today.

5. Conclusions and future research

Recent big data applications require the classification of a huge amount of objects in an extremely short time. For example, in advertising networks, credit scoring and fraud detection applications, the classification of hundreds of thousands of entities (e.g., clients) is expected sometimes in less than a second. The ENT is thus a critical measure in such cases.

This research aims to provide guidance and practical considerations for the construction of DTs where the objective function is to minimize the ENT. It is well known that the construction of a DT is an NP-complete problem, and accordingly, there are complexity limits for the construction of an optimal DT. The existing heuristics for the constructing DTs vary; some use a greedy algorithm that continuously searches for the best feature to be placed at each node, with less consideration

as to how this may affect the DT in future construction steps (with exceptions such as in [19] or the work done by Michael Franklin and his colleagues in the Berkeley AMP lab). Others try to improve the ENT by adopting a look-ahead procedure and set features to multiple tree levels at once. The challenge with these methods is in the sharp exponential increase of the computational complexity in the look-ahead level. Other DT construction methods aim to lower the complexity by selecting only subsets of features at each step of the tree construction. In this way, the algorithm can look further ahead without bearing the full cost of a standard look-ahead function. In all these approaches, however, the underlying assumption is that the solution improves as the number of steps that are looked ahead is increased. This assumption is shown to be false in several experiments carried out in this study, as well as in previous studies [26]. Following this observation, the incentive to use a look-ahead function is greatly reduced. If one pays considerably more in processing time and may end up with a worse solution, a risk-averse user would not try to construct a DT with conventional look-ahead schemes.

The proposed SF-GOTA algorithm has a number of appealing properties. First, when the subsets parameter is set to $s = 1$, it produces an identical solution to the *ID*3 or the GOTA, and thus practically generalizes these algorithms. Second, as the subsets parameter s is increased, SF-GOTA adds additional trees to test, although it never removes 'promising' trees from the pool of trees that were already constructed. In this manner, SF-GOTA ensures that, at all times, a larger 'subsets parameter' (denoted here by s) cannot produce worse results than those obtained by a smaller subsets parameter. A third important point is the algorithm complexity. The complexity increase of SF-GOTA in the parameter s is considerably more moderate compared to the complexity increase of GOTA, ID3 or C4.5 in the parameter k (reaching a polynomial complexity with enough processors). This allows for a larger look-ahead investigation than the one available by the GOTA: the GOTA evaluates the interactions among various features at the first k levels of the tree, whereas SF-GOTA does not evaluate all these interactions on the first s levels of the tree, but instead it investigates only those interactions that look the most promising. Nonetheless, as a heuristic, it is of course possible that SF-GOTA will reach a local optima, missing the optimal combination of features, as seen in several examples in this study. The fourth and possibly one of the most important advantages of the SF-GOTA is its natural fit to be processed on parallel processors: because the SF-GOTA generates multiple separated DTs, each tree can be constructed on a different processor. This allows one to increase the subsets parameter s without experiencing the full rise in complexity if additional processors are available. Last but not least, SF-GOTA is a very simple and straightforward algorithm and does not require any complex pre-processing to implement. In fact, although we focused in this study on the ENT measure and binary trees, the same approach can be easily extended and implemented to other measures such as classification accuracy on non-binary trees.

The numerical solutions obtained with SF-GOTA were found promising, indicating that the algorithm is competitive with respect to other DT construction heuristics. The low complexity of the SF-GOTA leads to fast identification of potential solutions. The consistent improvement nature of the algorithm implies that a practitioner could essentially choose a value of s that guarantees obtaining a solution within a given time frame and be certain that this solution is the best one over all SF-GOTA implementations with a lower value of s.

There are several future research directions that are of immediate interest. Finding the right algorithm parameters, such as the number of sub-trees s and the look-ahead k is a straightforward extension for this work. Using a 'branch and bound' scheme to refine the SF-GOTA search is another interesting direction. Supporting non-binary datasets, as well as developing a multi-objective optimization scheme for minimizing both the ENT and the classification error, could result in an improved algorithm, because these criteria do not necessarily lead to the same optimal solution. Moreover, the SF-GOTA does not account for missing or noisy data, which is non-realistic and should be addressed to obtain a higher algorithmic robustness. Finally, the proposed SF-GOTA framework could be associated with other approaches that aim for proper selection of variables, such as the '*InfoQ's dimension of chronology*' suggested by Kenett and Smueli [41].

Appendix

Decision tree construction algorithms that are based on information gain choose the test to be placed in a given node according to the test that gives the highest (normalized) information gain that is greater than zero. An interesting situation occurred in some of the datasets used in this study, whereby it was not possible to choose a test for a node because all tests gave zero bits of information.

There are two possible reasons why a test generates zero bits of information. The first reason can be seen in Table A.1. Namely, none of the available tests is capable of separating the symptoms of the dataset. In this case, it is pointless to place a test in the given node of the tree.

The second reason that a test can generate zero bits of information can be seen in Table A.2. Calculating the information gain of each of the available tests results in zero bits of information gain, even though the tests do partition the data. In this case, combining two tests together, such as $\mathbf{T_1}$ and $\mathbf{T_2}$ would result in an information gain of 1 bit (full classification).

Table A.1. Zero information gain (1).

Symp	U_3	U_7	U_{43}	U_{59}	U_{60}
Problem	0.2	0.15	0.25	0.15	0.25
Class	A_1	A_2	A_1	A_5	A_3
$\mathbf{T_2}$	0	0	0	0	0
$\mathbf{T_4}$	1	1	1	1	1
$\mathbf{T_7}$	0	0	0	0	0

Table A.2. Zero information gain (2).

Symp	U_1	U_2	U_3	U_4
Problem	0.25	0.25	0.25	0.25
Class	A_1	A_1	A_2	A_2
T_1	1	0	1	0
T_2	1	0	0	1
T_3	0	1	1	0
T_4	0	1	0	1

1. Choose s value according to available parallel processors, or arbitrarily
2. Calculate entropy of the node
3. Calculate information gain (IG) for all available tests.
4. If IG function for all tests=0
   4.1. Calculate IG for every combination of two tests
5. Choose s most favorable tests
   5.1. For each test chosen
      5.1.1. Update $s = \max(s-1,1)$
      5.1.2. Partition data according to test [combination] chosen
      5.1.3. Calculate accumulated expected number of tests
      5.1.4. For each partition that is not fully classified, recourse to step 2
6. Choose the test [combination] with the shortest accumulated expected number of tests.

Figure A.1. SF-GOTA+ pseudocode.

Whereas the first case of zero information gain should result in no test being allocated to the node, the second case requires such an allocation, allowing the algorithm to continue building the decision tree. In order to achieve this goal, the SF-GOTA algorithm was amended to identify cases such as these by checking if a combination of two tests would result in a positive information gain, when all individual tests result in zero bits of information gain. If there is a combination that results in a positive information gain, the algorithm will place two tests in one node, which would result in up to four offsprings (in a binary decision tree) and continue constructing the decision tree by the same algorithm from that point onwards.

According to the above principle, Figure A.1 presents the pseudocode for the amended algorithm, termed as SF-GOTA+.

Note: A simulated study was conducted in order to test for the frequency of cases where the amended algorithm was required, as well as indicating when a combination of two tests was insufficient. The results of the simulation study suggest that the first case is a very rare occurrence (happening only 328 times out of 55,860,000 datasets), that may occur when the symptoms are uniformly distributed. No cases where a combination of two tests were insufficient to avoid a zero information gain were detected in this simulated study. This does not imply of course that such a case cannot happen, but if it does, extending the SF-GOTA+ for a combination of tree cases and higher is straightforward.

References

1. Lee VJ, Lye DC, Sun Y, Leo YS. Decision tree algorithm in deciding hospitalization for adult patients with dengue haemorrhagic fever in Singapore. *Tropical Medicine & International Health* 2009; **14**(9):1154–1159.
2. Lowerre BT. The HARPY speech recognition system. *Unpublished Doctoral Dissertation*, Carnegie-Mellon University, Pittsburgh, 1976.
3. Olanow CW, Watts RL, Koller WC. An algorithm (decision tree) for the management of Parkinson's disease (2001): treatment guidelines. *Neurology* 2001; **56**(suppl 5):S1–S88.
4. Vlahou A, Schorge JO, Gregory BW, Coleman RL. Diagnosis of ovarian cancer using decision tree classification of mass spectral data. *BioMed Research International* 2003; **5**:308–314.
5. Balk B, Elder K. Combining binary decision tree and geo-statistical methods to estimate snow distribution in a mountain watershed. *Water Resources Research* 2000; **36**(1):13.

6. Simard M, Saatchi SS, De Grandi G. The use of decision tree and multiscale texture for classification of JERS-1 SAR data over tropical forest. *IEEE Transactions on Geoscience and Remote Sensing* 2000; **38**(5):2310–2321.
7. Pederson T. A decision tree of bigrams is an accurate predictor of word sense. *NAACL '01 Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies*, Pittsburgh, PA, 2001, 1–8.
8. Pal M, Mather PM. An assessment of the effectiveness of decision tree methods for land cover classification. *Remote Sensing Of Environment* 2003; **86**(4):554–565.
9. Tooke TR, Coops NC, Goodwin NR, Voogt JA. Extracting urban vegetation characteristics using spectral mixture analysis and decision tree classifications. *Remote Sensing of Environment* 2009; **113**(2):398–407.
10. Ben-Gal I, Herer Y, Raz T. Self-correcting inspection procedure under inspection errors. *IIE Transactions on Quality and Reliability* 2003; **34**(6):529–540.
11. Schietgat L, Vens C, Struyf J, Blockeel H, Kocev D, Džeroski S. Predicting gene function using hierarchical multi-label decision tree ensembles. *BMC Bioinformatics* 2010; **11**(1):2.
12. Baesens B, Van Gestel T, Viaene., Stepanova M, Suykens J, Vanthienen J. Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society* 2003; **54**:627–635.
13. Hand DJ, Henley WE. Statistical classification methods in consumer credit scoring: a review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 1997; **160.3**:523–541.
14. Metwally A, Agrawal D, Abbadi AE. Using association rules for fraud detection in Web advertising networks. *In Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment*, Trondheim, Norway, 2005, 169–180.
15. Phua C, Alahakoon D, Lee V. Minority report in fraud detection: classification of skewed data. *ACM SIGKDD Explorations Newsletter* 2004; **6**(1):50–59.
16. Tarter RE. Evaluation and treatment of adolescent substance abuse: a decision tree method. *The American Journal of Drug and Alcohol Abuse* 1990; **16**(1-2):1–46.
17. Hyafil L, Rivest RL. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters* 1976; **5**(1):15–17.
18. Perner P (ed.) *Data Mining on Multimedia Data*, vol. 2558. Springer: Verlag, Berlin Heidelberg, New York, 2002.
19. Ben-Gal I, Dana A, Shkolnik N, Singer G. Efficient construction of decision trees by the dual information distance method. *Quality Technology & Quantitative Management (QTQM)* 2014; **11**(1):133–147.
20. Goodman Rodney M, Smyth P. Decision tree design from a communication theory standpoint. *IEEE Transactions on Information Theory* 1988; **34**(5):979–994.
21. Quinlan JR. Induction of decision trees. *Machine Learning* 1986; **1**:81–106.
22. Quinlan JR. *C4.5: Programs for Machine Learning*. Morgan Kaufman Publishers: San Francisco, California, 1993.
23. Ben-Gal I. An upper bound of the weight-balanced testing procedure with multiple testers. *IIE Transactions on Quality and Reliability* 2004; **36**(5):481–491.
24. Hartmann CR, Varshney PK, Mehrotra KG, Gerberich C. Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory* 1982; **28**(4):565–577.
25. Murthy SK. On Growing Better Decision Trees from Data. *Dissertation Paper*, John Hopkins University, 1997.
26. Murthy SK, Salzburg S. Lookahead and pathology in decision tree induction. *IJCAI-95 Proceedings*, Montreal, Que., 1995, 1025–1031.
27. Zhou R, Hansen E. Beam-stack search: integrating backtracking with beam search. *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, Monterey, CA, 2005, 90–98.
28. Breiman L. Forests random. *Machine Learning* 2001; **45**:5–32.
29. Rissanen J. Modelling by shortest data description. *Automatica* 1978; **14**:465–471.
30. Ohta S, Kanaya F. Optimal decision tree design based on information theoretical cost bound. *IEICE Transactions* 1991; **E74**:2523–2530.
31. Quinlan JR, Rivest RL. Inferring decision trees using the minimum description length principle. *Information and Computation* 1989; **80**(3): 227–248.
32. Timofeev R. Classification and regression trees (CART) theory and applications. *Master Thesis*, Humboldt University, Center of Applied Statistics and Economics, Berlin, 2004.
33. Breiman L, Friedman JH, Olshen RA, Stone CJ. *Classification and Regression Trees*: Wadsworth, Belmont, CA, 1984. Since 1993 this book has been published by Chapman & Hall, New York.
34. Esmeir S, Markovitch S. Lookahead-based algorithms for anytime induction of decision trees. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML '04)*. ACM: New York, 2004; 33–39.
35. Yang C, Tian S, Long B. Selection of optimum test points set for analog faults dictionary techniques. *The 7th IEEE World Congress on Intelligent Control and Automation (WCICA 2008)*, Chongqing, China, 2008, 4970–4975.
36. Page D, Ray S. Skewing: an efficient alternative to lookahead for decision tree induction. *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI-2003*, Acapulco, Mexico, 2003, 601–607.
37. Elomaa T, Malinen T. *On Lookahead Heuristics in Decision Tree Learning*, Lecture Notes in Computer Science, vol. 2871, 2003. Foundations of Intelligent Systems: San Francisco, California, 2004, 445–453.
38. Norton SW. Generating better decision trees. *In Proceedings of the* 11^{th} *International Joint Conference on Artificial Intelligence, IJCAI-89*, vol. 1, Detroit, MI, 1989, 800–805.
39. Evans DJ. *Parallel Processing Systems: An Advanced Course*. Cambridge University Press: London, England, 1982.
40. Frank A, Asuncion A. *UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]*. University of California, School of Information and Computer Science: Irvine, CA, 2010.
41. Kenett SR, Shmueli G. On information quality. *Journal of the Royal Statistical Society A* 2014; **177**:3–38. Part 1.