COMPUTING WITH NOISY INFORMATION [*]

Felix Schmetz
Advanced Algorithms and Data Structures
ETH Zürich

November 1, 2018

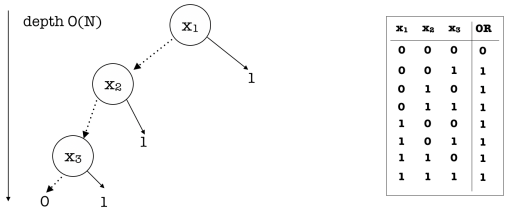
I. Introduction

Algorithms and trees are closely related in the field of computer science. Any algorithm can be represented by a decision (or comparison) tree. The time-complexity of an algorithm is closely related to the depth of the corresponding decision tree. This paper studies fault tolerant algorithms i.e algorithms where the decisions can be faulty and provides bounds for the depth (or time-complexity) of the resulting decision (or comparison) trees.

i. Model

We consider two types of trees.

Boolean Decision Trees
The first type of tree is the *Boolean Decision Tree* which is used to represent boolean functions of N variables.



x_1	x_2	x_3	OR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Figure 1: Boolean decision tree (OR)

Each leaf is labeled 0 or 1, and corresponds to an evaluation of the function. As we see in *Figure 1*, each node in the tree which is not a leaf, is one of the given input values. Edges represent a decision the algorithm makes: We go along the left edge if the value of the variable represented by the node we are querying is 0 and the right edge if it is 1.

Comparison Trees
The second type of tree we are considering are *Comparison Trees*, which are used in problems that require comparisons. In such a tree each node represents a comparison between two elements of the input (in search algorithms one of these element will always be the query). These elements are represented by their indices.

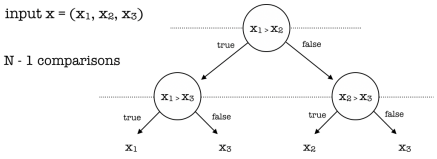


Figure 2: Comparison tree: max element in input

Noise
When we talk about *noise*, we are simply talking about mistakes made by a computer in the decision nodes. There are several ways to deal with faults, e.g. reconfigure (fix) the hardware to repeat a computation if an error is detected, or create a *robust algorithm* that tolerates faults.
In our *noisy* model for the first type of tree, we receive the wrong value for a given input variable with a certain probability $1 - p$, $p \in \left(\frac{1}{2}, 1\right)$. For the second type of tree each comparison returns a wrong result with probability $1 - p$, $p \in \left(\frac{1}{2}, 1\right)$.

As we have discussed above, this paper models algorithms as randomized decision or comparison trees that result in a correct computation with probability at least $1 - Q$ in every instance, where Q is a tolerance parameter bounded in $\left(0, \frac{1}{2}\right)$

Example 1.1: Majority voting
We consider a *clean* comparison tree, i.e. no noise is present, to find the maximum of N elements. Such a tree would have depth $N - 1$. In our noisy model we will most likely have a faulty result if each node is susceptible to failure, thus we have to come up with a better algorithm. A simple solution to that problem is using a majority vote for each comparison. We can repeat every comparison in the tree $O\left(\log\left(\frac{N}{Q}\right)\right)$ times and determine the correct result by a majority vote. The probability of each comparison being wrong is at most $\frac{Q}{N}$, and the depth of this tree is then $O(N \log(\frac{N}{Q}))$ because each of the $(N-1)$ nodes of the *clean* tree is repeated $O(\log(\frac{N}{Q}))$ times.

Similarly, we can apply the same method $O\left(\log\left(\frac{N}{Q}\right)\right)$ repetitions and majority voting) on any decision tree of depth and create a noisy algorithms of depth $O\left(d \log\left(\frac{d}{Q}\right)\right)$.

Conclusions::
This paper shows that while the logarithmic factor described above is unavoidable for some problems, it is unnecessary for others.

Another conclusion is that there is a distinction between the *static adversary* case where the correctness of every node is fixed at p and the *dynamic adversary* case where the correctness of each node in the tree can be set by an adversary optimally in the range $p \in [p, 1)$.

ii. Related papers and problems

Other papers have studied networks of noisy gates, in which every gate could give the wrong answer with some probability, Boolean decision trees in which an adversary is allowed to corrupt at most k nodes (read operations) along any root-leaf path or comparison trees for binary search for N elements, where k comparisons are incorrect.

The paper we are discussing is different to the related papers, since we have no control over the 'faultiness' of nodes in the tree in advance and treat each of them independently, which basically means that each node in our tree can be independently faulty with some probability. By working in this very specific setting we are able to deal with a fairly large amount of errors and nonetheless compute the correct solution.

iii. Notation

Let's define a few key notations:

- $D^{\text{Prob}}_{N,Q}(\Pi) = \Omega(f)$ is the minimum depth of a noisy probabilistic tree for input of size N and problem Π with tolerance Q.
- $D^{\text{Det}}_{N,Q}(\Pi) = O(f)$ is the same for a deterministic tree.
- $D_{N,Q}(\Pi) = \Theta(f)$ is the notation without the distinction between probabilistic and deterministic.

f is the complexity of the algorithm to solve the problem as a function of N and Q.

II. Boolean decision trees

The goal of this section is to find a lower bound on the depth for a noisy boolean decision tree computing the K-of-N threshold function.

We define the threshold function as

$$TH_K^N = \begin{cases} 1, & w(\bar{X}) \geq K, \\ 0, & \text{otherwise.} \end{cases},$$

where $\bar{X} = (X_1, ..., X_N)$ and $w(\bar{X}) = \sum_{i=1}^{N} X_i$.

K = 1: In this most basic case, the threshold function is essentially just the OR of all inputs.

THEOREM 2.1: $D_{N,Q}^{\text{Prob}}(OR) = \Omega\left(\frac{N \log \frac{1-Q}{Q}}{\log \frac{p}{1-p}}\right)$

PROOF: Let $\bar{X} = (X_1, ..., X_N)$ be the input vector of length N, $\bar{0} = (0, ..., 0)$ and $\bar{1}_j$ denote an input vector $X_j = 1$ and all remaining entries 0. We show that distinguishing between $\bar{0}$ and $\bar{1}_j$ requires the stated depth. Let $Pr\{l|\bar{X}\}$ be the probability of reaching a leaf l in a boolean decision tree of depth d for the input $\bar{X}$ (In a probabilistic decision tree it combines the the probabilities of random choices of the algorithm with the probabilities of the random answers to the queries).

Assume that X_j appears $r(j,l)$ times on the path from root to l. Then

$$Pr\{l|\bar{1}_j\} \geq \left(\frac{1-p}{p}\right)^{r(j,l)} Pr\{l|\bar{0}\}$$

.

Distinguishing between the case of the 0 vector and the vector with $X_j = 1$ at a specific leaf depends on the ratio of probabilities of random answers and the number of times X_j is present on the path.

For any leaf l, $\sum_{j=1}^{N} r(j,l) = d$, since it is the sum of all encounters of X_j, for $j = 1...N$ along a certain path from root to leaf l. Therefore $\sum_{j=1}^{N} \left(\frac{1-p}{p}\right)^{r(j,l)}$ attains its minimum over all choices of $r(j,l)$ at $N\left(\frac{1-p}{p}\right)^{\frac{d}{N}}$. This can be shown using *Jensen's Inequality*.

For a set of leaves L, we define

$$Pr\{L|\bar{X}\} = \sum_{l \in L} Pr\{l|\bar{X}\}$$

Thus, letting S denote the set of leaves labeled 0, we get

$$\begin{aligned} \sum_{j=1}^{N} Pr\{S|\bar{1}_j\} &= \sum_{j=1}^{N} \sum_{l \in S} Pr\{l|\bar{1}_j\} \\ &\geq \sum_{l \in S} \sum_{j=1}^{N} \left(\frac{1-p}{p}\right)^{r(j,l)} Pr\{l|\bar{0}\} \\ &\geq Pr\{S|\bar{0}\} N \left(\frac{1-p}{p}\right)^{\frac{d}{N}} \end{aligned} \tag{1}$$

It holds that $Pr\{S|\bar{0}\} \geq (1-Q)$ and for every j, $Pr\{S|\bar{1}_j\} \leq Q$, thus we have

$$QN \geq (1-Q)N \left(\frac{1-p}{p}\right)^{\frac{d}{N}}.$$

We can rewrite our result to obtain the bound on d:

$$QN \geq (1-Q)N \left(\frac{1-p}{p}\right)^{\frac{d}{N}}.$$

$$Q \geq (1-Q) \left(\frac{1-p}{p}\right)^{\frac{d}{N}}.$$

$$\log Q \geq \log(1-Q) + (\log(1-p) - \log p)\frac{d}{N}.$$

$$N \log Q \geq N \log(1-Q) + (\log(1-p) - \log p)d.$$

$$d(\log p - \log(1-p)) \geq N \log(1-Q) - N \log Q.$$

$$d \geq \frac{N \log\left(\frac{1-Q}{Q}\right)}{\log\left(\frac{p}{1-p}\right)}.$$

q.e.d.

The result from the theorem we have just proven is of great help to proving the following lower bound.

THEOREM 2.2: For every $K \leq \frac{N}{2}$ (symmetry) $D^{\text{Prob}}_{N,Q}(TH^N_K) = \Omega\left(\theta N \log K + \frac{N \log \frac{1-Q}{Q}}{\log \frac{p}{1-p}}\right)$, for $\theta = \frac{(1-\frac{Q}{1-Q})}{\log \frac{1}{1-p}}$.

We will not go into detail for this proof but rather explain how the proof of Theorem 2.1 can be extended to prove 2.2. It is important to keep in mind that for our lower bounds it suffices to use a static adversary.

We need to split the proof in two parts. In the case that $K \leq \max\{\frac{(1-Q)}{Q}, C\}$, for some constant C, then the adversary can announce the values of input variables $X_1, X_2, ...X_k$ in advance to be 1. This means that computing TH^N_K is then reduced to the problem of computing the OR function of the remaining $N - K + 1$ bits.

On the other hand, if for a sufficiently large constant C, $K > max\{(1-Q)/Q, C\}$, the proof is a bit more complicated, such that we will not go into any details here.

We present the following upper bound without proof.

THEOREM 2.3: $D^{\text{Det}}_{N,Q}(TH^N_K) = O\left(N \log \frac{m}{Q}\right)$, where $m = \min\{K, N-K\}$. In particular, $D^{\text{Det}}_{N,Q}(OR)$ and $D^{\text{Det}}_{N,Q}(AND)$ are both $O\left(N \log \frac{1}{Q}\right)$

III. Comparison trees

This section concerns noisy comparison trees. We will discuss their bounds for several algorithms and present the ideas behind some proofs to understand how these trees behave under noise.

We present an algorithm for noisy binary search by thinking of is as a *random walk* on the 'exact' (non-noisy) binary search tree. By presenting this algorithm we will also show bounds for algorithms requiring binary search, such as sorting, merging and K-SEL (Selecting the k-th largest element).

Each node of the tree represents a subinterval of $(-\infty, \infty]$, and is labeled by a pair representing the end points of this interval. Each leaf of the search tree represents an interval between two consecutive input values. There are $N+1$ leaves, with the i-th representing $(x_{i-1}, x_i]$, where $x_0 = -\infty$ and $x_{N+1} = \infty$. For an internal node u of the tree, let T_u, denote the subtree rooted at u, then u is the interval obtained by merging the intervals of the leaves. The tree has depth $\lceil \log N \rceil$. To search with unreliable comparisons we extend the tree: Each leaf x_l is a parent of a chain of length $m' = O(\log(N/Q))$. The nodes of the chain are labeled with the same interval as the leaf.

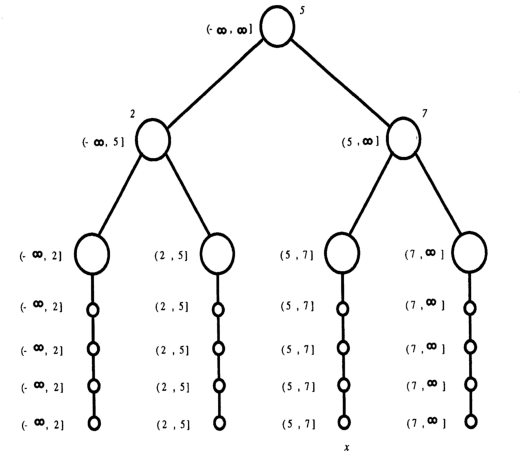
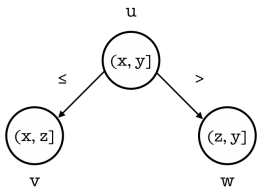


Figure 4: Extended tree for (2, 5, 7).[*]

Let X be the key being searched for in the tree. The search begins at the root of the tree, and advances or backtracks according to the results of the comparison. Whenever reaching a node u, the algorithm first checks that X really belongs to the interval $(x_l, x_h]$ associated with u, by comparing it to the endpoints of the interval. Such a test can fail due to a noisy comparison, in which case the algorithm interprets a failure as revealing an inconsistency due to an earlier mistake, and consequently, the computation backtracks to the parent of u in the tree. If the test succeeds, on the other hand, then the computation proceeds to the appropriate child of u. To select the appropriate child of u we compare the key with the central element z of u's interval.

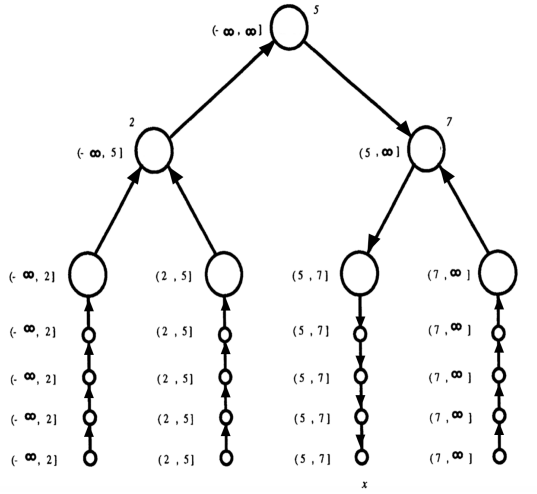


The search is continued for $m = O(log(N/Q))$ steps, $m < m'$ (thus never reaches the end of any chain). The outcome of the algorithm is the left endpoint of the interval labeling the node at which the search ends. In out figure, for X = 6, the algorithm should end at the leaf marked x.

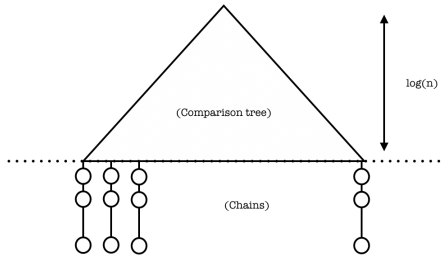
The paper goes on to prove that the appropriate chain is reached with probability at least $1 - Q$ with the help of Markov Chains.

This means we can conduct BINARY-SEARCH in $O(log(N/Q))$.

PROOF: Let w be a leaf of the extended tree and the correct insertion location of an element X. We direct all edged of the tree towards w, thus our tree looks like the following



We define a forward transition to be a step along an edge in the direction it is pointing to. A backward transition is the inverse.
Let m_f be a random variable counting the number of forward transitions and let m_b denote the number of backward transitions. It can be shown using *Chernoff's* bound for $m = m_f + m_b = c \log(n/Q)$, for a suitable constant c that $m_f - m_b > \log N$ with probability at least $1 - Q$, implying that the appropriate chain is reached.



This means that if we find ourself in a chain we are right with probability at least $1 - Q$, since more than $\log N$ transitions were forward transitions.
The same goes for sorting: Using N insertions of our searching algorithm we achieve sorting in $O(N \log(N/Q))$. It can be shown that the lower bounds for binary search and sorting are the same as their upper bounds.

IV. Conclusion

We briefly summarize our results and give a quick outlook on how these can be extended.

i. Final importance

The problem with this paper is, it does not really come to a specific result by itself. The main takeaway here are a series of proofs for properties and characteristics of fault-tolerant algorithms and their decision (or comparison) tree depth when susceptible to failure. This nevertheless does not reduce the importance of the paper, after all it has been cited around 67 times (source ACM).

ii. Extensions and Open Problems

There exists a series of problems tightly coupled to the ones we have discussed here, such that the reductions from the bounds the paper found can be extended to find bounds for the noisy tree depth of problems, such as

1. **UNARY-TO-BINARY:** Binary Representation of input
2. **COMPARISON:** Compare two numbers
3. **ADDITION:** Addition of input
4. **MATCHING:** Does the adjacency matrix have a matching with $\geq k$ edges?

Definitions from [A.K. Chandra et al. 1984]

Results of this paper can also be extended in such a way so that we can show for any function with a constant-depth formula of size N, there exists a noisy decision tree of depth $O\left(log\frac{1}{Q}\right)$.

References

[*] U. Feige, P. Raghavan, D. Peleg, E. Upfal. Computing with Noisy Information

[A.K. Chandra et al. 1984] A.K. CHANDRA, L. STOCKMEYER AND U. VISHKIN, Constant depth reducibility, SIAM J. Comput., 13 (1984), pp. 423-439.