

2.1 Decision trees

A decision tree is a binary tree where each node is labeled with a literal and each of the edges correspond to one of the values which the literal can take. The leaves of the tree output the label of the path that leads to that leaf, and the output is either 0 or 1. The size of the decision tree is equal to the number of nodes in the tree. An example decision tree is illustrated in :

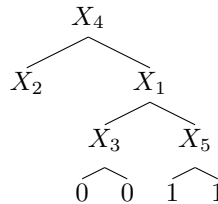


Figure 1

As for example, if we consider the parity function, we have a function that maps from a stream of bits (0 or 1) to a single bit, i.e. either 0 or 1. Expressed mathematically, we can represent the function as $\text{PAR}(X): f\{X\}^n \rightarrow \{0, 1\}$, where $X \in \{0, 1\}^n$ and the output is 1 if the number of 1's in X is odd, and 0 otherwise.

We know that we need a $O(n)$ decision list in order to compute the parity of a stream of bits of length n . Therefore, in order to compute the parity of the first $\log n$ bits of an input of length n , we need a $O(\log n)$ decision list. However, to compute the same parity function using a decision tree, we'll need a decision tree of size n to compute the parity of the first $\log n$ bits. We generalize this result as we move on to prove the next fact.

Fact 1 *Every polynomial-sized decision tree can be computed by a $\log n$ decision list.*

Equivalently, we might say that there exists a $n^{O(\log n)}$ time Mistake Bound algorithm for computing poly-sized decision trees.

In order to prove the fact, we need to define the *rank* of a decision tree.



Figure 2

Definition 1 Rank is a function mapping decision trees to integers;

$$\text{Rank} : DT \rightarrow \Psi^+.$$

The rank of a null tree or a single leaf node tree is 0. The rank of all other trees that can be represented as in FIG in the form of a root node and two subtrees rooted at that node can be computed as follows:

- If $\text{Rank}(T_1) = \text{Rank}(T_2)$ then $\text{Rank}(T) = 1 + \text{Rank}(T_1)$
- otherwise $\text{Rank}(T) = \text{Max}(\text{Rank}(T_1), \text{Rank}(T_2))$

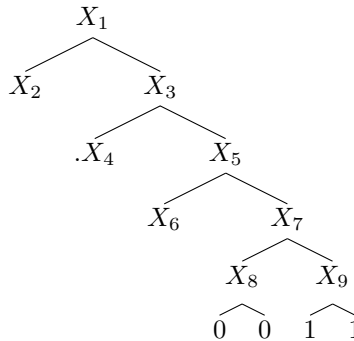


Figure 3

As for example, the rank of the tree in Figure 2 is computed to be 3, and that of the one in Figure 3 is 4.

Now, in order to prove Fact 1, we first prove the following claim:

Claim 1 If a decision tree has size s , then rank of the tree is less than or equal to s

Proof: We prove this by induction on the structure of the decision tree. The base case is trivial, that is, of a tree with just one node. The rank of the tree is (by definition) 0 and we know that $\log_2 1 = 0$ [since the number of nodes in the tree is 1].

Now, lets assume that the tree shown in Figure 2 has a size s , and that the hypothesis holds for the subtrees rooted at T_1 and T_2 . Here, two cases arise:

Case 1: $\text{Rank}(T_1) \neq \text{Rank}(T_2)$ In this case, $\text{Rank}(T) = \text{Rank}(T_2) \leq \log s$ by assumption.

Case 2: $\text{Rank}(T_1) = \text{Rank}(T_2)$ In this case, we know that $\text{Rank}(T) = 1 + \text{Rank}(T_1)$ by definition. The subtree rooted at T_1 should have at most $s/2$ nodes, and therefore $\text{Rank}(T_1) \leq \log(s/2)$.

Therefore,

$$\begin{aligned} \text{Rank}(T) &\leq 1 + \log_2 \frac{s}{2} \\ &\leq 1 + \log_2 s - \log_2 2 \\ &\leq \log s. \end{aligned}$$

Hence proved.



Now, we build the final proof of the fact originally stated.

Claim 2 *A Rank(t) decision tree has (or is computed by) a $t + 1$ decision list*

Proof: We prove this by induction again. The base case is trivial, since for one node a decision tree and a decision list are basically the same structure. We also assume, without loss of generality that the decision list never falls off the end, i.e., the last node in the list always computes and fires if the evaluation reaches till that node.

Let $C_1 \dots C_m$ be a set of conjunctions at the nodes of a decision list that represents the subtree rooted at T_1 and $D_1 \dots D_n$ be the set of conjunctions at nodes of the decision list that is equivalent to the subtree rooted at T_2 . Please note here that the total number of literals in each of the C_i and D_j conjunctions is at most t .

Again, we deal with two cases here.

Case 1: $\text{Rank}(T_1) = \text{Rank}(T_2) = t - 1$

From the node T , we reach the subtree rooted at T_1 only when the value of the literal being evaluated at the node (X_3 , say) is 0. Otherwise, when $X_3 = 1$, we reach the subtree rooted at T_2 . Since the decision lists C and D evaluate these subtrees already, if we add the literal \bar{X}_3 to each conjunction in C and X_3 to each conjunction in D , then we have two lists that evaluate the paths to each subtree including node T . Now, combining these two lists by concatenating D to C yields a decision list with at most $t + 1$ literals at each node. Thus the case is proved.

Case 2: $\text{Rank}(T_1) \neq \text{Rank}(T_2)$

Let $\text{Rank}(T_1) = t-1$ and $\text{Rank}(T_2) = t$. The subtree rooted at T_1 along with the node T is converted into a decision list by adding \bar{X}_3 to each conjunction in that list. We can then concatenate this list with the decision list that represents T_2 .

However, it is not immediately obvious how the latter subtree rooted at T_2 may yield a t -decision list, since it itself has rank t . Now, since the hypothesis is inductively assumed to be true, we can consider the subtrees rooted at T_2 . If one of the children here has a subtree of rank t again, we recursively find such children until we come across a subtree of rank $t-1$. Once found, the case becomes symmetric to the beginning of Case 2.

In either case, the right ends of the decision lists are not reachable since one of the nodes of each decision tree fires.

Thus the proof.

