Chapter 6

Evaluating And/Or Trees

By Sariel Har-Peled, April 26, 2022[1]

That's all a prophet is good for - to admit somebody else is an ass or a whore.

The Violent Bear It Away, Flannery O'connor

6.1. Evaluating an And/Or Tree

Let T_{2k} denote a complete binary tree of height $2k$ – this tree has $n = 2^{2k}$ leaves. The inputs to the tree are boolean values stored in the leafs, where nodes are AND/OR nodes alternatingly. The task at hand is to evaluate the tree - where the value of a internal node, is the operation associated with the node, applied to the values returned from evaluating its two children.

Figure 6.1: The tree T_2, inputs in the leafs, and the output.

Defined recursively, T_2 is a tree with the root being an AND gate, and its children are OR gates. This tree has four inputs. More generally, T_{2k}, is T_2, with each leaf replaced by T_{2k-2}. Let $n = 2^{2k}$.

So the input here is T_{2k}, together with 2^{2k} values stored in each leaf of the tree. Consider here the *query model* – instead of read the values in the leafs, the algorithm has to explicitly perform a query to get the value stored in the leaf. The question thus is can we minimize the number of queries the algorithm needs to perform.

It is straightforward to evaluate such a tree using a recursive algorithm in $O(n)$ time. In particular, it following is not too difficult to show.

Exercise 6.1.1. Show that any deterministic algorithm, in the worst case, requires $\Omega(n)$ time to evaluate a tree T_{2k}.

The key observation is that AND (i.e., $\wedge$) gate evaluation can be shortcut – that is, if $x = 0$ then $x \wedge y = 0$ independently on what value y has. Similarly, an OR (i.e., $\vee$) gate evaluation can be shortcut – since if $x = 1$, then $x \vee y = 1$ independently of what y value is.

6.1.1. Randomized evaluation algorithm for T_{2k}

The algorithm is recursive. If the current node v is a leaf, the algorithm returns the value stored at the leaf. Otherwise, the algorithm randomly chooses (with equal probability) one of the children of v, and evaluate them recursively. If the returned value, is sufficient to evaluate the gate at v, then the

algorithm shortcut. Otherwise, the algorithm evaluates recursively the other child, computes the value of the gate and return it.

6.1.2. Analysis

Lemma 6.1.2. *The above algorithm when applied to T_{2k}, in expectation, reads the value of at most 3^k leaves, and this also bounds its running time.*

Proof: The proof is by induction. Let start with T_2. There are two possibilities:

(i) The tree evaluates to 0, then one of the children of the `AND` gate evaluates zero. But then, with probability half the algorithm would guess the right child, and evaluate it first. Thus, in this case, the algorithm would evaluate (in expectation) $\leq (1/2)2 + (1/2)4 = 3$ leafs.

(ii) If the output of the tree is 1, then both children of the root must evaluate to 1. Each one of them is an `OR` gate. Arguing as above, an `OR` gate evaluating to one, requires in expectation to read $(1/2)1 + (1/2)2 = 3/2$ leafs to be evaluated by the randomized algorithm. It follows, that in this case, the algorithm would read (in expectation) $2(3/2) = 3$. (Note, that this is an upper bound – if all the four inputs are 1, this algorithm would read only 2 leafs.)

For $k > 1$, consider the four grandchildren of the root c_1, c_2, c_3, c_4. By induction, in expectation, evaluating each of $c_1, \ldots, c_4$, takes 3^{k-1} leaf evaluations. Let $X_1, \ldots, X_4$ be indicator variables that are one if c_i is evaluated by the recursive algorithm. Let Y_i be the expected number of leafs read when evaluating c_i (i.e., $\mathbb{E}[Y_i] = 3^k$). By the above, we have that $\mathbb{E}[\sum_i X_i] = 3$. Observe that X_i and Y_i are independent. (Note, that the X_i are not independent of each other.) We thus have that the expected number of leafs to be evaluated by the randomized algorithm is

$$\mathbb{E}\left[\sum_i X_i Y_i\right] = \sum_i \mathbb{E}[X_i Y_i] = \sum_i \mathbb{E}[X_i]\,\mathbb{E}[Y_i] \leq 3\,\mathbb{E}[Y_i] = 3 \cdot 3^{k-1} = 3^k. \qquad \blacksquare$$

Corollary 6.1.3. *Given an `AND/OR` tree with n leafs, the above algorithm in expectation evaluates*

$$3^k = 2^{k \log_2 3} = \left(2^{2k(\log_2 3)/2}\right) = n^{(\log_2 3)/2} = n^{0.79248}$$

leafs.

6.2. Bibliographical notes

The `AND/OR` tree algorithm is from Marc Snir work [Sni85]. One can show a lower bound using Yao's min-max principle, which is implied by the minimax principle of zero sum games.

References

[Sni85] M. Snir. *Lower bounds on probabilistic linear decision trees*. *Theor. Comput. Sci.*, 38: 69–82, 1985.