Decision Trees

Katrina LaCurts

September 13, 2007

1 Introduction

A decision tree is a model used for determining the output of a boolean function. A decision tree typically reads one bit of the input and then makes a decision based on whether that bit is 0 or 1. It continues to read bits in this manner until it has enough information to evaluate the function. This model naturally lends itself to a visual representation as a binary tree.

Complexity classes on decision trees are not the same as the standard complexity class on a Turing Machine model. For example, it is fairly easy to show that the decision tree P is not equal to the decision tree NP. This unfortunately does not lead us to the conclusion that P≠NP on Turing Machines. In this paper, when we speak of classes like P and NP, we really mean decision tree P and decision tree NP.

We are measuring decision tree complexity in terms of the number of leaves. Another way to measure this complexity is depth of the tree, which is equivalent to the number of variables read. While this has some "practical" application [1], it doesn't generalize well to defining complexity classes. For one, we clearly couldn't define P as "functions with a polynomial number of variables read." Assuming we have infinite memory (and hence only need to read a variable once), all functions would then be in P. We could define P as poly-log depth, but there is not a lot that one can do in poly-log depth.

2 Preliminaries

Definition. A boolean function is in P if it can be solved by a decision tree with a polynomial number of leaves.

Definition. A boolean function is in NP if it can be solved by a forest of polynomial-leaved trees.

We can take two trees T_1 and T_2 and combine them. For every leaf of T_1 that evaluates to 0, attach tree T_2 to it. We will refer to this as the combined tree of T_1 and T_2.

Definition. The combined tree of two decision trees T_1 and T_2 is the tree gotten by adjoining tree T_2 to every leaf of T_1 that evaluates to 0.

Product Theorem. *Given two decision trees T_1 and T_2, the number of leaves in their combined tree is $\leq L_1 * L_2$, where L_1 is the number of leaves in T_1 and L_2 is the number of leaves in T_2.*

[1] Saks, Wigderson, "Probabilistic Boolean Decision Trees and the Complexity of Evaluating Game Trees"

Proof. T_1 cannot have more than L_1 leaves that evaluate to 0. For each of these leaves, we gain L_2 new leaves by attaching T_2. Thus, the total number of leaves is (number of 0-leaves in T_1) $*$ $(L_2) \leq L_1 * L_2$. □

Corollary. *If we have a problem that can be by a forest consisting of a constant number of polynomial-leaved decision trees, it is in P.*

Proof. Let $T_1, T_2, \ldots, T_k$ be the trees which solve a boolean function f, and let $L_1, L_2, \ldots, L_k$ be their respective numbers of leaves. By the Product Theorem and by induction, the total number of leaves in the combined tree is $\leq L_1 * L_2 \ldots * L_k$. Since each L_i is a polynomial, the product is a polynomial. Hence the combined tree is in P and thus f is. □

3 NP = Everything

In the Turing Machine model of complexity, there are quite a few classes "bigger" than NP. This is not the case when considering decision trees. In fact, we have the following theorem.

Theorem. *NP contains every boolean function.*

Proof. Consider a boolean function f on n variables and its corresponding truth table. We can convert its truth table into a DNF formula, with each clause being a line in the truth table. Since there are n variables, we will get 2^n clauses, each containing all of the n variables (either as-is or negated). We can compute this by making a decision tree for each clause. This decision tree has $O(n)$ leaves. There are 2^n such trees, and hence we have a forest of polynomial-leaved trees. $f \in NP$. □

4 P ≠ NP

Is NP = P in terms of decision trees? We present two examples to show that it's not.

4.1 Example 1

Consider $(x_1 \vee x_2 \vee \cdots \vee x_{\sqrt{n}}) \wedge (x_{\sqrt{n}+1} \vee \cdots \vee x_{2\sqrt{n}}) \wedge \cdots \wedge (x_{(\sqrt{n}-1)\sqrt{n}+1} \vee \cdots \vee x_n)$. That is, a CNF formula on n variables broken up into $\sqrt{n}$ blocks of size $\sqrt{n}$. We claim that this is not in P. In particular, we claim that each path down the tree must be at least $\sqrt{n}$ long.

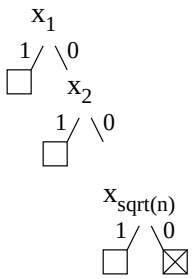
Proof. Suppose there exists a path with $< \sqrt{n}$ variables read. There are two cases.

1. The decision tree claims that this path evaluates to 0. For the function to evaluate to 0, we must have all of the variables in one block be 0. Since there are $\sqrt{n}$ variables in each block, the decision tree could not have checked them all, and thus be sure that the path evaluates to 0.
2. The decision tree claims that this path evaluates to 1. For the function to evaluate to 1, we must have at least one variable from each block be 1. Since there are $\sqrt{n}$ blocks, the decision tree could not have checked them all, and thus can't be sure that the path evaluates to 1.

Each case leads to a contradiction and so each path in the tree must be at least $\sqrt{n}$ long. □

By forcing each path to be at least $\sqrt{n}$ long means that any decision tree for this problem will have $\Omega(2^{\sqrt{n}})$ leaves. $2^{\sqrt{n}}$ is not a polynomial. This problem is in NP, however, as we showed in the previous section. Thus, P ≠ NP.

Notice that our decision tree may be redundant in the sense that we may read the same variable more than once in the tree. For example, we can imagine the following tree for our function.



The tree simply reads each variable in the first block until it finds a variable that evalutes to 1. It then moves onto the second block, and so forth. Each box in the figure represents the tree moving onto the next block in the CNF formula. The first step in each block, then, will be to read $x_{\sqrt{n}+1}$. This presents no problem for us, but it does mean that we might want to look into this problem in a branching program context.

4.2 Example 2

Consider $(x_1 \wedge x_2 \wedge \cdots \wedge x_{\sqrt{n}}) \vee (x_{\sqrt{n}+1} \wedge \cdots \wedge x_{2\sqrt{n}}) \vee \cdots \vee (x_{(\sqrt{n}-1)\sqrt{n}+1} \wedge \cdots \wedge x_n)$. This is the DNF analog of example 1. As in the previous example, our strategy still forces each branch in the tree to be $O(\sqrt{n})$. Thus, there are at least $\Omega(2^{\sqrt{n}})$ leaves. This example is not in P.

These two problems provide a good example of why we're not measuring complexity in terms of number of variables read. Consider $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ vs. example 1. Clearly this is in P. In both cases, however, we need to read all of the variables. If we were to measure complexity in terms of number of variables read, these two problems would be in the same class despite example 1 being an intuitively "harder" problem.

4.3 Generalization

Both of these examples generalize nicely. We'll consider only the first one (the CNF formula), but the second example is analogous.

Consider a CNF formula with n variables divided into b blocks of size s. First of all, we can immediately see that $n = b * s$ and thus $b = n/s$.

Theorem. *Any decision tree for this problem will have* $\Omega(2^{min(b,s)}) = \Omega(2^{min(s,n/s)})$ *leaves.*

Proof. As we discussed in the proof in example 1, in order for a tree to evaluate to 0, we must have all of the variables in one block be 0 and thus must read at least s variables. In order for a tree to evaluate to 1, we must have at least one variable in each block be 1, and thus must read at least b variables. Each branch in any decision tree must then have $\Omega 2^{min(s,b)}$ leaves. □

Corollary. *We can generate a function that can be solved by an* $\Omega(2^{n^\epsilon})$ *leaves for any* $0 < \epsilon \leq \frac{1}{2}$.

Proof. In our previous example, let $s = n^\epsilon$. Then $b = n/s = n^{1-\epsilon}$. Our tree then has $\Omega(2^{min(n^\epsilon, n^{1-epsilon})})$. If $\epsilon \leq \frac{1}{2}$, our tree has $\Omega(2^{n^\epsilon})$ leaves. □

5 NP ∩ co-NP ⊈ P

In terms of decision trees, NP ∩ co-NP ⊈ P. We present a more in-depth analysis of the proof given by Jukna et. al[2]. It is known that a boolean function in n variables has a decision tree of size $2^{O(\log n \log^2 N)}$ where N is the total number of clauses (monomials) in the minimal DNFs for f and $\neg f$ (size still being the number of leaves in the tree). What we want to show is that this bound is tight up to the $\log n$ factor. In particular, we'll show that MAJ_3 requires a decision tree of size $2^{\Omega(\log^{1.58} N)}$.

Definition. MAJ_3 is the iterated function in which each gate has three inputs and outputs the majority bit of the three inputs.

Definition. Let F_h be the monotone function in $n = 3^h$ variables computed by a balanced read-once formula of height h in which every gate is MAJ_3.

Definition. Let $\text{DNF}(f)$ denote the minimum number of monomials in a DNF for f.

Definition. $\|F_h\|$ =$\text{DNF}(F_h)$ + $\text{DNF}(\neg F_h)$.

Theorem. *$dt(F_h) \geq 2^{\log^\gamma N_h}$ where $N_h = \|F_h\|$ and $\gamma = \log_2 3 \approx 1.58$.*

Clearly this theorem leads us to our desired conclusion, that MAJ_3 requires a $2^{\Omega(\log^{1.58} N)}$ sized decision tree.

Notice that $\text{DNF}(F_0) = 1$, since F_0 is just one variable. Also notice that $\text{DNF}(F_h) = 3{*}\text{DNF}(F_{h-1})^2$.

> $\text{DNF}(F_h) = \text{DNF}(F_{h-1}) \wedge \text{DNF}(F_{h-1}) \wedge \text{DNF}(F_{h-1})$, so this explains the $3{*}\text{DNF}(F_{h-1})$. Each $\text{DNF}(F_{h-1})$ is a collection of ands of ors, so we use the distributive law to transform them into a collection of ors. This is where we get $\text{DNF}(F_{h-1})^2$.

Furthermore, F_h is self-dual; that is, $\text{DNF}(F_h)$=$\text{DNF}(\neg F_h)$.

Using this fact, we have $N_h = 2 * 3^{2^h - 1}$ (This is easily verifiable) and then $n = 3^h = \Theta(\log^\gamma N_h)$ (Also easily verifiable). Now that we can describe n in terms of N_h, we turn to the following lemma.

Lemma. *$dt(f) \geq 2^{|S|} * \sum_{T \supseteq S} |\hat{f}_h(T)|$ where $S \subseteq [n]$, and $\hat{f}_h(T)$ is the T^{th} Fourier coefficient.*

We're going to take this lemma on faith and apply it to $S = [n]$. We then have that $dt(f) \geq 2^n |\hat{f}_h([n])|$. Now we would like to get a bound on $|\hat{f}_h([n])|$.

Let $a_h = \hat{F}_h([n])$. $a_0 = 1$ since F_0 is just one variable. Now we're going to switch from {1,0} notation to {-1,+1} notation, where -1 indicates a true value. Notice that

$$\text{MAJ}_3(x_1, x_2, x_3) = (x_1 + x_2 + x_3 - x_1x_2x_3)/2.$$

F_h =$\text{MAJ}_3(F_{h-1}^{(1)}, F_{h-1}^{(2)}, F_{h-1}^{(3)})$ where each $F_{h-1}^{(i)}$ is on a disjoint set of variables.

Then $F_h = \frac{1}{2}(\sum_{v=1}^{3} F_{h-1}^{(v)} - \prod_{v=1}^{3} F_{h-1}^{(v)})$. This is just utilizing the previous formula. The summation term does not dominate since F_{h-1} depends on fewer than $n = 3^h$ variables. So now we have $a_n = -\frac{1}{2} * a_{h-1}^3$. Solving the recurrence we get $a_h = (-1)^h * (\frac{1}{2})^{\frac{n-1}{2}}$. Then $dt(F_h) \geq 2^n * (\frac{1}{2})^{\frac{n-1}{2}} \in \Omega(2^n)$, which is what we wanted.

[2]Jukna, Razborov, Savicky, Wegener "On P versus NP∩co-NP for Decision Trees and Read-once Branching Programs"

6 Open Questions

- Are there problems with $\log n$ decision trees in their forest that are not in P?
- Do more leaves help? That is, are we better off with more trees for a problem, or with more leaves in a tree. Give the Product Theorem, does this even matter?
- Is RandP = P? If we measure complexity in terms of the number of variables read, the two classes are equal[3].
- Does the decision tree P correspond to a known class in terms of Turing Machines?

[3]Nisan, "CREW PRAMs and Decision Trees"