TOPIC 2

Decision Trees: Their Size

(Related sections: 10.4, 14.4)

Summary: If a boolean function f *and* its negation $\neg f$ have a DNF of size N, then f can be computed a decision tree of size $n^{O(\log^2 N)}$. This is almost optimal: there are *explicit* boolean functions requiring decision trees of size $2^{O(\log^2 N)}$. The proof employs spectral arguments.

The *size* of a decision tree[1] is the number of all its leaves. Let $\mathrm{dt}(f)$ denote the minimum size of a decision tree computing f.

We already know (see Theorem 10.22 in the book) that that P=NP∩co-NP for decision trees if we consider their *depth*. In this lecture we will show that the situation changes drastically if we consider the *size* instead of the depth. It turns out that then there are explicit boolean functions f such that *both* f and its negation $\overline{f}$ have *nondeterministic* decision trees of small size, whereas the size of any *deterministic* decision tree for f is super-polynomial.[2]

Let $D(f)$ denote the minimal number of monomials in a DNF of f. That is, $D(f)$ is the minimal number t such that f can be written as an Or of t monomials.

Ex. 2.1. Let D_1 be a DNF of a boolean function f, and D_2 be a DNF of its negation $\overline{f}$. Show the following *cross-intersection property*: if K is a monomial in D_1 then every monomial in D_2 contains at least one literal which is contradictory to at least one literal in K.

Let f be a Boolean function, and suppose we know that $D(f)$ is small. Is then the decision tree also small? The following examples show that it may be *not* the case.

Ex. 2.2. Consider the following function $f(X)$ on $n = m^2$ boolean variables $X = \{x_{ij} : 1 \leq i, j \leq m\}$:

$$f = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{m} x_{ij}.$$

Show that $\mathrm{dt}(f) \geq 2^{\sqrt{D(f)}}$. *Hint*: Observe that all the minterms and maxterms of f have length m. Show that every such function requires a decision tree of size at least 2^m.

[1] See Sect. 10.4 of the book for the definition.

[2] We have not defined what a "nondeterministic decision tree" actually is, because we do not what to introduce a formal notion of "nondeterministic computation" (we don't need it later). Informally, a tree is nondeterministic if it has additional "guessing" nodes from which the computation can proceed in both directions. Thus, on one input we may have many computation paths from the root to a leaf. A tree accepts the inputs if at least one of these paths reach a leaf labeled by 1.

The function from this example has very small DNF (of size m) but the DNF of its negation

$$\overline{f} = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{m} \overline{x}_{ij}$$

is huge—it has 2^m monomials. It is therefore natural to ask what happens if *both* the function f *and* its negation $\overline{f}$ have small DNFs? Put otherwise, does P=NP∩co-NP for decision trees if we consider the *size* as their complexity measure? This question was open for a long time, and was recently answered *negatively* in [**18**]. Below we describe this solution.

2.1. The upper bound

The sum $N(f) := D(f) + D(\overline{f})$ will be called the *weight* of f. It is clear that $N(f) \leq \mathrm{dt}(f)$ (just because every decision tree represents *both* the function and its negation). It was long unknown if $\mathrm{dt}(f)$ is polynomial in $N(f)$, i.e. if $\mathrm{dt}(f) \leq N(f)^c$ for some absolute constant c.

The following result is due to Ehrenfeucht and Haussler (1989) and says that the decision tree size of any Boolean function is quasi-polynomial in its weight. In the next section we will show that this upper bound is almost optimal: there are explicit functions f for which $\mathrm{dt}(f) = 2^{\Omega(\log^2 N)}$. Here and throughout, $\log x$ stands for $\log_2 x$.

Theorem 2.3 (Ehrenfeucht–Haussler 1989)**.** *Let f be a Boolean function in n variables and $N = D(f) + D(\overline{f})$ be its weight. Then*

$$\mathrm{dt}(f) \leq \left(\tfrac{n}{\log^2 N}\right)^{O(\log^2 N)} \leq n^{O(\log^2 N)}.$$

We give a simpler and more intuitive proof of this result.

Proof. The idea is to employ the following simple "greedy" strategy: given DNFs for f and $\overline{f}$, let the decision tree *always* test the "most popular" literal first.

Assume, we have DNFs for both f and $\overline{f}$, and let N be the total number of monomials in these two DNFs. Since the disjunction of these two DNFs is a tautology (i.e., outputs 1 on all inputs), there must a monomial of length at most $\log N$.

> **Ex. 2.4.** Show this. *Hint*: Every monomial of length k accepts only 2^{n-k} of the inputs.

Select one of such monomials and denote its length by k. The selected monomial belongs to one of the two DNFs. By the cross-intersection property of monomials (see Exercise 2.1), every monomial in the other DNF contains at least one literal which is contradictory to at least one literal in the selected monomial. Hence, there is a literal in the selected monomial, which is contradictory to at least a $1/k$-portion of the monomials in the other DNF. Thus, if we evaluate this literal to 1, then all these monomials will get the value 0 and so will disappear from the DNF.

Test this variable first and apply this strategy recursively to both restrictions which arise. By the observation we just made, for each node v, at least one of its two successors is such that at least one of the two DNFs in it decreases by a factor of $(1 - 1/k)$. Let us call the corresponding outgoing edge(s) *decreasing*. Now, if v is a node (not a leaf) such that the path from the source to v contains s decreasing edges, at least one of the two initial DNFs was decreased at least $s/2$ times, and each time it was decreased by a factor of $1 - 1/k \geq 1 - 1/\log N$. If s would be at least $2\log^2 N$ then at least one of the DNFs at v would have only

$$\begin{aligned} N\left(1 - \frac{1}{\log N}\right)^{s/2} &< N \cdot e^{-s/(2\log N)} \\ &\leq N \cdot e^{-\log N} \\ &= N^{1-\log e} < 1 \end{aligned}$$

monomials, which is impossible (because v is not a leaf). Thus, *every* path to a leaf has at most n edges, and among them at most $s := 2\log^2 N$ can be decreasing. Recall that for *every* node at least one of the out-going edges was decreasing. Assume w.l.o.g. that every node has *exactly* one decreasing edge (if there were two, we simply ignore one of them). Mark decreasing edges by 1 and the remaining edges by 0. Then every leaf corresponds to a 0-1 string of length at most n with at most s ones. The number of such strings (and hence, the total number of leaves) does not exceed $L(n, s/2)$, where $L(n,t)$ denotes the maximal possible number of leaves in a decision tree of depth n such that *every* path from the root to a leaf has at most t 1-edges.

It remains to estimate $L(n,t)$ for $t = s/2$. Clearly, we have the following recurrence:

$$L(n,t) \leq L(n-1,t) + L(n-1,t-1),$$

with $L(0,t) = L(n,0) = 1$. Using the identity $\binom{n-1}{k-1} = \binom{n}{k} - \binom{n-1}{k}$, this recurrence implies

$$L(n,t) \leq \sum_{i=0}^{t} \binom{n}{i} \leq \left(\tfrac{ne}{t}\right)^t.$$

Thus,

$$\mathrm{dt}(f) \leq L(n, s/2) = L(n, \log^2 N) \leq \left(\tfrac{n}{\log^2 N}\right)^{O(\log^2 N)},$$

as desired. □

Ex. 2.5. (Research project). Is it possible to improve the upper bound $\mathrm{dt}(f) \leq n^{O(\log^2 N)}$ in Ehrenfeucht-Haussler theorem to $\mathrm{dt}(f) \leq 2^{O(\log^2 N)}$?

We have proved that every function has a decision tree of size quasi-polynomial in its weight. Can this upper bound be improved? The negative answer was recently given in [**18**]: there are explicit Boolean functions f for which

$$\mathrm{dt}(f) \geq 2^{\Omega(\log^2 N)} = N^{\Omega(\log N)}.$$

That is, there exist boolean functions f whose DNF and CNF *both* have size at most N, but any decision tree computing f requires size super-polynomial in N. This implies that, unlike for the depth of decision trees, for their size we have an inequality P≠NP∩co-NP.

2.2. Lower bounds

We already know how to prove that a Boolean function cannot be computed by a decision tree of small size: it is enough that its minterms and maxterms are long enough (see Exercise 2.2). However, this trivial argument cannot help us to establish trade-off between the weight of a Boolean function and its decision tree size: we want to show that the decision tree size is super-polynomial in the total number of minterms *and* maxterms, and this number is exponentially large in the length of minterms/maxterms.

Example: Let $\mathcal{A} = \{A_0, A_1\}$ be disjoint k-element subsets and let $\mathcal{B}$ be the family of all k-element subsets B of $\{1, \dots, n\}$ such that $B \cap A_0 \neq \emptyset$ and $B \cap A_1 \neq \emptyset$. Let f be a (monotone) Boolean function whose minterms are all the monomials $\bigwedge_{i\in B} x_i$ over all $B \in \mathcal{B}$. This function has only two maxterms $\bigvee_{i\in A_0} x_i$ and $\bigvee_{i\in A_1} x_i$. Thus, the weight $N(f)$ of f is $|\mathcal{B}| + 2 \geq \binom{n-2}{k-2} \geq n^{\Omega(k)}$. Since minterms and maxterms have length k, the (trivial) argument, mentioned above, yields a lower bound 2^k on the size of any decision tree computing f. But this bound is much smaller than $N(f)$.

So, to disprove the P=NP∩co-NP conjecture for the size of decision trees we have to argue in some more subtle way. For this purpose we will use an argument which has many applications in engineering. The argument is based on harmonic analysis of Boolean functions, and is known as the *spectral argument.*'

2.2.1. Fourier transforms for boolean functions

Roughly speaking, the main idea of what is known as "spectral argument" is to estimate how far is a given Boolean function apart from the parity function. For this it will be convenient to switch to $(-1, +1)$-notation, i.e., to consider Boolean functions as mappings from $\{-1, +1\}^n$ to $\{-1, +1\}$, where the correspondence $1 \to -1$ and $0 \to +1$ is assumed. Explicitely, this correspondence is given by the transformation $x' = 1 - 2x$ which transforms the value $x \in \{0, 1\}$ into the value $x' \in \{+1, -1\}$.

> **Ex. 2.6.** Show that in $(-1, +1)$ notation the $\mathrm{OR}(x, y)$ turns to the function $\mathrm{OR}(x, y) = (x + y - x \cdot y + 1)/2$. What about the $\mathrm{AND}(x, y)$? What about the parity function $x \oplus y$?

We treat the functions $f : \{-1, +1\}^n \to \{-1, +1\}$ as elements of 2^n-dimensional vector space with an inner product defined by

$$\langle f, g\rangle := 2^{-n} \sum_{x\in\{-1,+1\}^n} f(x)g(x).$$

The set of all monomials

$$X_S := \prod_{i\in S} x_i\,, \qquad S \subseteq [n] := \{1, \dots, n\}$$

forms an orthonormal basis for this space. That is

- Every function $f : \{-1, +1\}^n \to \{-1, +1\}$ can be uniquely expressed as $\sum_S \hat{f}(S)X_S$, where $\hat{f}(S)$ are real constants;
- If $S \neq T$ then $\langle X_S, X_T\rangle = 0$, and for every S, $\langle X_S, X_S\rangle = 1$.

The *Fourier transform* of f is just the expansion of f as a linear combination of the X_S's. Since the X_S's form an orthonormal basis, the Sth Fourier coefficient $\hat{f}(S)$ of f is found via:

$$\hat{f}(S) := \langle f, X_S\rangle = 2^{-n} \sum_x f(x)X_S(x)$$

> **Ex. 2.7.** Check the last statement. *Hint*: Suppose that $f = \sum_S \hat{f}(S) \cdot X_S$. To find $\hat{f}(T)$ take the scalar product of f with X_T.

Ex. 2.8. Let $f = x_i$ be a single variable. Show that then $\hat{f}(\{i\}) = 1$.

Ex. 2.9. Suppose that the value of f does not depend on the ith variable, i.e. that

$$f(x_1, \dots, x_{i-1}, +1, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, -1, x_{i+1}, \dots, x_n)$$

Show that then $\hat{f}(S) = 0$ for every S with $i \in S$.

If $f(X)$ is a boolean function in n variables $X = \{x_1, \dots, x_n\}$, its *leading Fourier coefficient* (or LFC, for short) is $\hat{f}(S)$ for $S = \{1, \dots, n\}$.

Ex. 2.10. What is the LFC of the parity function $x_1 \oplus x_2 \oplus \cdots \oplus x_n$?

If $f(X)$ is an arithmetic combination of some functions with disjoint sets of variables, then the LFC of f can be easily commuted form the LFCs of these functions.

Ex. 2.11. Let $X = X_1 \cup X_2$ be a partition of X into two nonempty blocks, $X_1 \cap X_2 = \emptyset$. Let $f(x) = f_1(X_1) + f_2(X_2) + f_1(X_1) \cdot f_2(X_2)$ where $f_i(X_i)$ is an arbitrary boolean function depending only on the variables in X_i, $i = 1, 2$. Prove that the LFC of f is just the product of the LFCs of f_1 and f_2. *Hint*: Observe that the summand $f_1(X_1) + f_2(X_2)$ does not contribute to the LFC of f (cf. Exercise 2.9), and use the disjointness of X_1 and X_2 to compute the LCF of $f_1(X_1) \cdot f_2(X_2)$.

2.3. The spectral lower bound

The following lemma estimates the size of a decision tree for f in terms of Fourier coefficients of f.

Lemma 2.12. *For every Boolean function f in n variables and every subset of indices $S \subseteq \{1, \dots, n\}$ we have the bound*

$$\mathrm{dt}(f) \geq 2^{|S|} \cdot \sum_{T \supseteq S} |\hat{f}(T)|. \tag{2.13}$$

Proof. Take a decision tree for f of size $\mathrm{dt}(f)$. For a leaf ℓ, let $\mathrm{val}(\ell) \in \{-1, +1\}$ be its label (recall that we are in $(-1, +1)$-notation), and let I_ℓ be the set of indices of those variables, which are tested on the path to ℓ. Let $B_\ell \subseteq \{-1, +1\}^n$ be the set of all the inputs that reach leaf ℓ; hence, $|B_\ell| = 2^{n-|I_\ell|}$.

Since each input reaches a *unique* leaf, the sets B_ℓ are mutually disjoint. Hence,

$$\hat{f}(T) = 2^{-n} \sum_x f(x) \cdot X_T(x) = 2^{-n} \sum_\ell \sum_{x \in B_\ell} f(x) \cdot X_T(x) = \sum_\ell \mathrm{val}(\ell) \cdot \Delta(T, \ell)$$

for every $T \subseteq [n] := \{1, \dots, n\}$, where

$$\Delta(T, \ell) := 2^{-n} \sum_{x \in B_\ell} X_T(x).$$

Now, if $T \not\subseteq I_\ell$ (i.e., if some variable x_i with $i \in T$ is not tested along the path from the root to the leaf ℓ) then $X_T(x) = +1$ for exactly half of the inputs $x \in B_\ell$, and hence, $\Delta(T, \ell) = 0$. If $T \subseteq I_\ell$ then the value of X_T is fixed on B_ℓ to either $+1$ or -1, and hence, $|\Delta(T, \ell)| = 2^{-n} \cdot |B_\ell| = 2^{-|I_\ell|}$. Thus, in both cases, $|\Delta(T, \ell)| \leq 2^{-|I_\ell|}$. Since for any $S \subseteq [n]$ there are only $2^{|I_\ell|-|S|}$ sets T satisfying $S \subseteq T \subseteq I_\ell$, we conclude that

$$\begin{aligned}\sum_{T:T\supseteq S}|\hat{f}(T)| &\le \sum_{T:T\supseteq S}\sum_{\ell}|\Delta(T,\ell)| = \sum_{\ell}\sum_{T:T\supseteq S}|\Delta(T,\ell)|\\ &\le \sum_{\ell}2^{-|S|} = 2^{-|S|}\cdot \mathrm{dt}(f),\end{aligned}$$

and the desired bound (2.13) follows. □

2.3.1. A super-polynomial trade-off

Recall that our goal is to exhibit a boolean function f which requires decision tree of size super-polynomial in its weight $N = D(f) + D(\overline{f})$. In this section we will prove a super-polynomial lower bound of the form $N^{\Omega(\sqrt{\log N})}$ for the Iterated Majority function. In the next section we will use a bit more subtle argument to prove the lower bound $N^{\Omega(\log N)}$ which (in view of Ehrenfeucht–Haussler's Theorem 2.3) is almost the best possible.

The majority of three Boolean variables is given by

$$MAJ_3(x_1,x_2,x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3.$$

Note that in the $(-1,+1)$-representation (i.e., when the correspondence $1 \to -1$ and $0 \to +1$ is assumed) we have that

$$MAJ_3(x_1,x_2,x_3) = (x_1 + x_2 + x_3 - x_1x_2x_3)/2.$$

Consider the monotone function F_h in $n = 3^h$ variables which is defined inductively as follows: $F_0 = x$, $F_1 = MAJ_3(x_1,x_2,x_3)$ and for $h \ge 2$

$$F_h = MAJ_3(F_{h-1}^{(1)}, F_{h-1}^{(2)}, F_{h-1}^{(3)})$$

where $F_{h-1}^{(\nu)}$ are three copies of F_{h-1} with disjoint(!) sets of variables.

> **Ex. 2.14.** Show that F_h has $n = 3^h$ variables and its weight is $2\cdot 3^{2^h-1}$.
> *Hint*: Observe that: (1) $D(F_0) = 1$ and $D(F_h) = 3\cdot D(F_{h-1})^2$, and (2) the minimal DNF of the negation $\overline{F}_h$ coincides with the DNF of F_h with all the variables negated.

Theorem 2.15. *Let F_h be the iterated majority function and $N = D(f) + D(\overline{f})$ be its weight. Then*

$$\mathrm{dt}(F_h) \ge N^{\Omega(\sqrt{\log N})}.$$

Proof. By the previous exercise, the function $F_h(x_1,x_2,\ldots,x_n)$ has $n = 3^h = 2^{c\cdot h}$ variables, where $c = \log 3 > 3/2$, and has the weight $N = 2\cdot 3^{2^h-1} = 2^{\Theta(2^h)} = 2^{\Theta(n^{2/3})}$. Since $2^{\Omega(n)} \ge 2^{\Omega(\log^{3/2}N)} = N^{\Omega(\sqrt{\log N})}$, it is enough to prove the lower bound

$$\mathrm{dt}(F_h) \ge 2^{\Omega(n)}.$$

To prove this we will apply Lemma 2.12 with $S = [n] = \{1,\ldots,n\}$. Letting a_h to denote the absolute value $\left|\hat{F}_h([n])\right|$ of the leading Fourier coefficient of F_h, this lemma yields

$$\mathrm{dt}(F_h) \ge a_h \cdot 2^n.$$

It remains therefore to prove an appropriate lower bound on a_h. We proceed by induction on h.

Clearly, $a_0 = 1$, since F_0 is a variable (cf. Exercise 2.9).

For the inductive step recall that in the $(-1,+1)$-representation,

$$MAJ_3(x_1, x_2, x_3) = \frac{1}{2}\left(\sum_{i=1}^{3} x_i - \prod_{i=1}^{3} x_i\right)$$

Thus,

$$F_h = \frac{1}{2}\left(\sum_{\nu=1}^{3} F_{h-1}^{(\nu)} - \prod_{\nu=1}^{3} F_{h-1}^{(\nu)}\right)$$

Using the property of leading Fourier coefficients, established in Exercise 2.11, we obtain that

$$a_h = \tfrac{1}{2} a_{h-1}^3.$$

Together with the condition $a_0 = 1$, this recursion resolves to

$$a_h \quad = \quad 2^{-3^0} \cdot a_{h-1}^3 = 2^{-3^0-3^1} \cdot a_{h-2}^3 = 2^{-3^0-3^1-3^2} \cdot a_{h-3}^3 = \ldots = 2^{-\Delta},$$

where

$$\Delta = 3^0 + 3^1 + 3^2 + \cdots + 3^{h-1} = \frac{3^h - 1}{3 - 1} = (3^h - 1)/2 = (n-1)/2.$$

Thus

$$\mathrm{dt}(F_h) \geq a_h \cdot 2^n \geq 2^{-(n-1)/2} \cdot 2^n = 2^{\Omega(n)},$$

as desired. □

2.3.2. An almost optimal trade-off (optional section)

In (0,1)-notation,

$$\textsc{nand}(x, y) = \overline{x} \vee \overline{y} = \begin{cases} 0 & \text{if } x = y = 1 \\ 1 & \text{otherwise} \end{cases}$$

In (+1,-1)-notation (via the correspondence $1 \to -1$ and $0 \to +1$),

$$\textsc{nand}(x, y) = \tfrac{1}{2}(xy - x - y - 1) = \begin{cases} +1 & \text{if } x = y = -1 \\ -1 & \text{otherwise} \end{cases}$$

Iterated NAND is a function G_h on $n = 2^h$ variables which is computed by a read-once formula of depth h with NAND-gates. Put otherwise, this function is defined by the following recursion: $G_0 = x$ and

$$G_h = \tfrac{1}{2}\left(G_{h-1}^{(1)} \cdot G_{h-1}^{(2)} - G_{h-1}^{(1)} - G_{h-1}^{(2)} - 1\right), \tag{2.16}$$

where $G_{h-1}^{(1)}$, $G_{h-1}^{(2)}$ are two copies of G_{h-1} with disjoint sets of variables. For this function we have that

$$N_h := N(G_h) \leq 2^{2^{(h/2)+1}} \approx 2^{\sqrt{n}}.$$

By Ehrenfeucht-Haussler theorem,

$$\mathrm{dt}(G_h) \leq \left(\tfrac{ne}{\log^2 N_h}\right)^{\log^2 N_h} \leq N_h^{O(\log N_h)}.$$

The next result shows that this bound is optimal.

Theorem 2.17. $\mathrm{dt}(G_h) \geq N_h^{\Omega(\log N_h)}$.

Proof. It is enough to prove that $\mathrm{dt}(G_h) \geq \exp(\Omega(n))$, because $n \approx (\log N_h)^2$.

Say that a Fourier coefficient $\hat{G}_h(T)$ is *dense* if for every subtree of height 2, T contains the index of at least one of the four variables in that subtree. Let

$$c_h := \left| \sum_{T \text{ dense}} \hat{G}_h(T) \right| .$$

By direct computation, $c_2 = 27/8 = 3.375$. Indeed, in order to compute c_2, we use the following transformation. Let $f_1 = G_1^{(1)} + 1/2$ and $f_2 = G_1^{(2)} + 1/2$. Then it follows from (2.16) that

$$G_2 = \tfrac{1}{2} f_1 f_2 - \tfrac{3}{4} f_1 - \tfrac{3}{4} f_2 + \tfrac{1}{8}.$$

Since each monomial in f_1 and f_2 contains at least one variable and the sets of variables of f_1 and f_2 are disjoint, there are no common monomials in the four terms in the above expression of G_2. Hence, it is easy to calculate the sum of the absolute values of the coefficients in the non-constant monomials, which is $c_2 = 1/2 \cdot r_1 \cdot r_2 + 3/4 \cdot (r_1 + r_2) = 27/8 = 3.375$, where $r_1 = r_2 = 3/2$ is the sum of the absolute values of the coefficients in f_1 and f_2.

For $h > 2$ we use the recursion (2.16) directly. Only the first term $G_{h-1}^{(1)} \cdot G_{h-1}^{(2)}$ can contribute to dense coefficients. Hence,

$$c_h = \tfrac{1}{2}\, c_{h-1}^2 .$$

This resolves to $c_h = 2(c_2/2)^{2^{h-2}}$ which is $\exp(\Omega(n))$ since $c_2 > 2$.

Applying Lemma with $S = \emptyset$, we obtain that

$$\mathrm{dt}(G_h) \geq 2^0 \cdot \sum_T |\hat{f}(T)| \geq c_h = \exp(\Omega(n)) ,$$

thus completing the proof of the theorem. $\square$