

On the reliable computation of certain Boolean functions by noisy decision trees

A. Pedrotti

Dipartimento di Matematica, Università di Trento

Via Sommarive 14, 38050 Povo (Trento) - Italy

apedrott@science.unitn.it

Several computation models have been introduced in order to describe algorithms that act on unreliable inputs. Among them, noisy decision trees [FRPU94] describe the computation of Boolean functions of n inputs x_1, \dots, x_n , which are known only probabilistically. More precisely, a noisy Boolean decision tree is a computation tree where each node corresponds to querying one input variable x_i . The answer to a query is wrong with probability q , where q is a given constant $< 1/2$. It will not be relevant for our purposes to detail whether the error probability ranges within $[0, q]$ or is precisely equal to q (see [Pip85] and [FRPU94] for this distinction). Given a Boolean function $f(x_1, \dots, x_n)$ and a constant $\delta < 1/2$, we want to study the depth $T_f(n, \delta)$ of noisy decision trees that compute $f(x_1, \dots, x_n)$ with error probability bounded by δ ; for brevity, we say that they perform a δ -estimation of $f(x_1, \dots, x_n)$. Note that, by Chernov bounds [MR95], one can prove that each input variable x_i can be β -estimated by first querying it $O(\log(\frac{1}{\beta}))$ times, and next performing a majority voting among the answers. As a consequence, every Boolean function can be δ -estimated by $O(n \log(\frac{n}{\delta}))$ queries by $\frac{\delta}{n}$ -estimating each of the n inputs.

Much of the literature about the noisy decision tree model is concerned with functions that are difficult to compute in this model. One example is the parity function, whose δ -estimation needs $\theta(n \log(\frac{n}{\delta}))$ queries [PST91, RS91]. On the other hand, we will focus on the OR function, for which a tight $\theta(n \log(\frac{1}{\delta}))$ bound is proved in [FRPU94]. Our discussion can be easily paraphrased for the AND function. Note that, by the result of [FRPU94], a sequential estimation of the inputs, as described above, is not optimal. On the other hand, it turns out that the optimal querying algorithm described in [FRPU94] employs the same number of queries on every input, whereas the sequential evaluation can stop as soon

as the estimation of some input x_k returns one, thus needing only $O(k \log(\frac{n}{\delta}))$ queries. Our aim is to combine the advantages of the two methods, in order to design an $O(k \log(\frac{1}{\delta}))$ algorithm.

The algorithm of [FRPU94] consists of a tournament with $\lceil \log_2 n \rceil$ rounds, followed by an appropriate estimation of the winning input. In the first round of the tournament, the input variables pair up and play $\frac{\delta}{2}$ -accurate matches: each such match involves the $\frac{\delta}{4}$ -estimation of both parties. In the second round, the winners pair up and play $\frac{\delta}{4}$ -accurate matches. The error probability of a match is halved at each round, until the $\frac{\delta}{2^{\lceil \log_2 n \rceil}}$ -accurate final match. The algorithm ends up with a $\frac{\delta}{2^{\lceil \log_2 n \rceil}}$ -estimation of the winning variable.

We agree that a tie is broken in favor of the variable with minimum index. Let us suppose that not all inputs are zero, and let $k = \min\{j \mid x_j = 1\}$. It is easy to see that, due to the tie-breaking rule, x_k is the legitimate winner of the tournament; we want to estimate the probability that x_k be the actual winner. Let the *path* of an input variable x_j be the path connecting the leaf associated to x_j with the root. Note that x_k wins the tournament if all the matches lying on its path behave correctly; note also that there is no need to care about the correctness of the matches outside that path. As a consequence, x_k wins with probability out of $\frac{\delta}{2} + \frac{\delta}{4} + \dots + \frac{\delta}{2^{\lceil \log_2 n \rceil}} = \delta - \frac{\delta}{2^{\lceil \log_2 n \rceil}}$. Taking also the final estimation of the winner into account, we see that with probability out of δ the return value of the function is correct.

If all inputs are zero, no matter which input variable wins the tournament, the final estimation will correctly return zero with probability out of $\frac{\delta}{2^{\lceil \log_2 n \rceil}}$.

To improve on the above algorithm, we propose the following modifications:

- a) a match belonging to round r does no longer consist

of two $\frac{\delta}{2^{r+1}}$ -accurate variable estimations. Namely, if the opponents of a certain match are x_i and x_j , with $i < j$, we $\frac{\delta}{2^r}$ -estimate x_i , and we do not estimate x_j at all. If the estimation of x_i returns one, then x_i wins; otherwise, x_j wins;

- b) instead of playing the tournament round by round, we schedule the matches according to the following policy. Let us consider the tree associated to the tournament, whose leaves correspond to input variables, and whose internal nodes correspond to matches (in particular, the root corresponds to the final match). We perform a symmetric visit of that tree. Let us consider an internal node N , and the left and right subtrees T_L and T_R rooted at its sons. When we visit N , we already know the winner of the partial tournament represented by T_L ; let it be x_L . The visit of N consist of an estimation of x_L ; if the return value is one, we do not need to visit T_R at all.

Let us suppose that not inputs are zero, and let us note that the legitimate winner x_k of the modified tournament is the same as in the formulation of [FRPU94]. Due to a) and b) we see that, if x_k wins, no node lying on the right side of the path of x_k is visited. An exploration of the whole tree can take place only if something goes wrong on the path of x_k , and hence with probability at most δ . As a consequence, the algorithm needs an expected number $\mathbf{E}[T(n, \delta)] \in O((k + \log_2 n + \delta n) \log(\frac{1}{\delta}))$ of queries.

To make a further step towards an $O(k \log(\frac{1}{\delta}))$ complexity, we return to sequential evaluation. Let us show that the approach of δ -estimating each input, stopping as soon as some estimation returns one, works whenever there is some non-zero input. Let as usual $k = \min\{j \mid x_j = 1\}$. If the computation stops before evaluating x_k , then the correct result one has been returned "by the wrong reason." Otherwise, the evaluation of x_k returns the value one with probability out of δ , as desired.

The difficulties arise when all inputs are zero. Here, each variable estimation is a potential source of trouble; we would need each input to be estimated with accuracy $\frac{\delta}{n}$, but we cannot afford for this. A way out is to evaluate each input x_j as follows. We first query x_j as many times as is required for a $\frac{(n-1)\delta}{n}$ -accurate evaluation, that is, $O(\log(\frac{1}{\delta}))$ times; if the answer is zero, we trust it. On the other hand, if the answer is one, we look for a confirmation, performing a $\frac{\delta}{n}$ -accurate checking evaluation. Obviously, we choose to trust the result of the check rather than the original one. The key observation is that the expected number of checks that are encountered throughout the algorithm is low. Namely,

it is roughly δn if all inputs are zero, and $1 + \delta n$ otherwise. It can be shown that in the former case we have $\mathbf{E}[T(n, \delta)] \in O(n \log(\frac{1}{\delta}) + n\delta \log n)$; in the latter case we have $\mathbf{E}[T(n, \delta)] \in O((k + \delta n) \log(\frac{1}{\delta}) + (1 + \delta n) \log n)$. If $\delta n \in o(1)$, the latter complexity becomes $O(k \log(\frac{1}{\delta}) + \log n)$.

The undesired second term can be eliminated by tuning the accuracy of the evaluations and of the checks more finely. Let us require each evaluation to be $\frac{\delta}{2}$ -accurate, and let us require the i -th check to be $\frac{\delta}{2^i}$ -accurate. It can be shown that these choices lead to an expected number $\mathbf{E}[T(n, \delta)] \in O((k + n\delta) \log(\frac{1}{\delta}) + n^2 \delta^2)$ of queries. For $\delta n \in o(1)$, this is the desired result.

We hope that the above techniques can be extended to face other problems in the field of reliable computation. We are especially interested in the study of how a noisy computation can be interspersed with appropriate checks in order to enhance its reliability. This approach has already been adapted in [P98] to other problems and models than those considered above. We feel that its application to the noisy decision tree model should be further investigated.

References

- [FRPU94] U. Feige, P. Raghavan, D. Peleg, and E. Upfal, Computing with noisy information, *SIAM Journal on Computing*, **23** (1994), 1001-1018.
- [MR95] R. Motwani and P. Raghavan, "Randomized Algorithms," Cambridge University Press, Cambridge, NY, 1995.
- [Pip85] N. Pippenger, On networks of noisy gates, *Proc. IEEE Symposium on the Foundations of Computer Science* (1985), 30-38.
- [P98] A. Pedrotti, Reliable RAM computation in the presence of noise, *PhD thesis, Scuola Normale Superiore, Pisa, Italy, 1998*.
- [PST91] N. Pippenger, G. D. Stamoulis, and J. N. Tsitsiklis, On a lower bound for the redundancy of reliable networks with noisy gates, *IEEE Transactions on Information Theory*, **37** (1991), 639-643.
- [RS91] R. Reischuk and B. Schmelz, Reliable computation with noisy circuits and decision trees - a general $n \log n$ lower bound, *Proc. IEEE Symposium on the Foundations of Computer Science* (1991), 602-611.

Fast Abstract ISSRE Copyright 1999.