Decision tree model

From Wikipedia, the free encyclopedia

In computational complexity and communication complexity theories the **decision tree model** is the model of computation or communication in which an algorithm or communication process is considered to be basically a decision tree, i.e., a sequence of branching operations based on comparisons of some quantities, the comparisons being assigned the unit computational cost.

The branching operations are called "tests" or "queries". In this setting the algorithm in question may be viewed as a computation of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ where the input is a series of queries and the output is the final decision. Every query is dependent on previous queries.

Several variants of decision tree models have been introduced, depending on the complexity of the operations allowed in the computation of a single comparison and the way of branching.

Decision trees models are instrumental in establishing lower bounds for computational complexity for certain classes of computational problems and algorithms: the lower bound for worst-case computational complexity is proportional to the largest depth among the decision trees for all possible inputs for a given computational problem. The computation complexity of a problem or an algorithm expressed in terms of the decision tree model is called **decision tree complexity** or **query complexity**.

Contents

Classification by query computational complexity

Simple decision tree

The model in which every decision is based on the comparison of two numbers within constant time is called simply a decision tree model. It was introduced to establish computational complexity of sorting and searching.[1]

The simplest illustration of this lower bound technique is for the problem of finding the smallest number among *n* numbers using only comparisons. In this case the decision tree model is a binary tree. Algorithms for this searching problem may result in *n* different outcomes (since any of the *n* given numbers may turn out to be the smallest one). It is known that the depth of a binary tree with *n* leaves is at least $\log n$, which gives a lower bound of $\Omega(\log n)$ for the searching problem.

However this lower bound is known to be slack, since the following simple argument shows that at least *n* - 1 comparisons are needed: Before the smallest number can be determined, every number except the smallest must "lose" (compare greater) in at least one comparison.

Along the same lines the lower bound of $\Omega(n \log n)$ for sorting may be proved. In this case, the existence of numerous comparison-sorting algorithms having this time complexity, such as mergesort and heapsort, demonstrates that the bound is tight.

Linear decision tree

Linear decision trees, just like the simple decision trees, make a branching decision based on a set of values as input. As opposed to binary decision trees, linear decision trees have three output branches. A linear function $f(x_1, \ldots, x_i)$ is being tested and branching decisions are made based on the sign of the function (negative, positive, or 0).

Geometrically, $f(x)$ defines a hyperplane in $\mathbf{R}^n$. A set of input values reaching any particular nodes represents the intersection of the half-planes defined by the nodes.

Algebraic decision tree

Algebraic decision trees are a generalization of linear decision trees to allow test functions to be polynomials of degree *d*. Geometrically, the space is divided into semi-algebraic sets (a generalization of hyperplane). The evaluation of the complexity is more difficult.

Classification by query computational model

Deterministic decision tree

If the output of a decision tree is $f(x)$, for all $x \in \{0,1\}^n$, the decision tree is said to "compute" f. The depth of a tree is the maximum number of queries that can happen before a leaf is reached and a result obtained. $D(f)$, the **deterministic decision tree** complexity of f is the smallest depth among all deterministic decision trees that compute f.

Randomized decision tree

One way to define a **randomized decision tree** is to add additional nodes to the tree, each controlled by a probability p_i. Another equivalent definition is to select a whole decision tree at the beginning from a set of decision trees based on a probability distribution. Based on this second definition, the complexity of the randomized tree is defined as the greatest depth among all the trees associated with probabilities greater than 0. $R_2(f)$ is defined as the complexity of the lowest-depth randomized decision tree whose result is $f(x)$ with probability at least $2/3$ for all $x \in \{0,1\}^n$ (i.e., with bounded 2-sided error).

$R_2(f)$ is known as the Monte Carlo randomized decision-tree complexity, because the result is allowed to be incorrect with bounded two-sided error. The Las Vegas decision-tree complexity $R_0(f)$ measures the *expected* depth of a decision tree that must be correct (i.e., has zero-error). There is also a one-sided bounded-error version known as $R_1(f)$.

Nondeterministic decision tree

The nondeterministic decision tree complexity of a function is known more commonly as the certificate complexity of that function. It measures the number of input bits that a nondeterministic algorithm would need to look at in order to evaluate the function with certainty.

Quantum decision tree

The quantum decision tree complexity $Q_2(f)$ is the depth of the lowest-depth quantum decision tree that gives the result $f(x)$ with probability at least $2/3$ for all $x \in \{0,1\}^n$. Another quantity, $Q_E(f)$, is defined as the depth of the lowest-depth quantum decision tree that gives the result $f(x)$ with probability 1 in all cases (i.e. computes f exactly). $Q_2(f)$ and $Q_E(f)$ are more commonly known as **quantum query complexities**, because the direct definition of a quantum decision tree is more complicated than in the classical case. Similar to the randomized case, we define $Q_0(f)$ and $Q_1(f)$.

Relationship between different models

It follows immediately from the definitions that for all n-bit Boolean functions f, $Q_2(f) \leq R_2(f) \leq R_1(f) \leq R_0(f) \leq D(f) \leq n$, and $Q_2(f) \leq Q_E(f) \leq D(f) \leq n$.

Blum and Impagliazzo,[2] Hartmanis and Hemachandra,[3] and Tardos[4] independently discovered that $D(f) \leq R_0(f)^2$. Noam Nisan found that the Monte Carlo randomized decision tree complexity is also polynomially related to deterministic decision tree complexity: $D(f) = O(R_2(f)^3)$.[5] (Nisan also showed that $D(f) = O(R_1(f)^2)$.) A tighter relationship is known between the Monte Carlo and Las Vegas models: $R_0(f) = O(R_2(f)^2 \log R_2(f))$.[6] This relationship is optimal up to polylogarithmic factors.[7]

The quantum decision tree complexity $Q_2(f)$ is also polynomially related to $D(f)$. Midrijanis showed that $D(f) = O(Q_E(f)^3)$,[8][9] improving a quartic bound due to Beals et al.[10] Beals et al. also showed that $D(f) = O(Q_2(f)^6)$, and this is still the best known bound. However, the largest known gap between deterministic and quantum query complexities is only quadratic. A quadratic gap is achieved for the OR function; $D(OR_n) = n$ while $Q_2(OR_n) = \Theta(\sqrt{n})$.

It is important to note that these polynomial relationships are valid only for *total* Boolean functions. For partial Boolean functions, that have a domain a subset of $\{0,1\}^n$, an exponential separation between $Q_E(f)$ and $D(f)$ is possible; the first example of such a problem was discovered by Deutsch and Jozsa.

These relationships can be summarized by the following inequalities, which are true up to constant factors:[9]

- $D(f) \leq R_0(f)^2$ [2][3][4]
- $D(f) \leq R_1(f)R_2(f)$[5]
- $D(f) \leq R_2(f)^3$ [5]
- $R_0(f) \leq R_2(f)^2 \log R_2(f)$[6]
- $D(f) \leq Q_2(f)^6$ [10]
- $D(f) \leq Q_E(f)^2 Q_2(f)^2$ [10]
- $D(f) \leq Q_1(f)^2 Q_2(f)^2$ [11]
- $R_0(f) \leq Q_1(f)Q_2(f)^2 \log N$[12]
- $D(f) \leq Q_1(f)Q_2(f)^2$ [9]

See also

- Minimum spanning tree#Decision trees

References

1. "Data structures and algorithms, byAlfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman

2. Blum, M.; Impagliazzo, R. (1987). "Generic oracles and oracle classes". *Proceedings of 18th IEEE FOCS*. pp. 118–126.
3. Hartmanis, J.; Hemachandra, L. (1987), "One-way functions, robustness, and non-isomorphism of NP-complete sets", *Technical Report DCS TR86-796, Cornell University*
4. Tardos, G. (1989). "Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A?". *Combinatorica*. **9**: 385–392. doi:10.1007/BF02125350
5. Nisan, N. (1989). "CREW PRAMs and decision trees". *Proceedings of 21st ACM STOC*. pp. 327–335.
6. Kulkarni, R. and Tal, A. On Fractional Block Sensitivity. Electronic Colloquium on Computational Complexity (ECCC) Vol. 20. 2013.
7. Ambainis, A., Balodis, K., Belovs, A., Lee, T., Santha, M., and Smotrovs, J. (2015). Separations in query complexity based on pointer functions. arXiv preprint arXiv:1506.04719.
8. Midrijanis, Gatis (2004), "Exact quantum query complexity for total Boolean functions", *arXiv:quant-ph/0403168*, arXiv:quant-ph/0403168
9. Midrijanis, Gatis (2005), "On randomized and quantum query complexities", *arXiv:quant-ph/0501142*, arXiv:quant-ph/0501142
10. Beals, R.; Buhrman, H.; Cleve, R.; Mosca, M.; de Wolf, R. (2001). "Exact quantum query complexity for total Boolean functions". *Journal of ACM*. **47**: 778–797.
11. H. Buhrman, R. Cleve, R. de Wolf, and Ch. Zalka. Bounds for Small-Error and Zero-Error Quantum Algorithms. In 40th IEEE Symposium on Foundations of Computer Science (FOCS'99), pp.358-368. cs.CC/9904019, 1999.
12. S. Aaronson. Quantum Certificate Complexity. IEEE Conference on Computational Complexity, pp. 171-178, 2003.

Surveys

- Buhrman, Harry; de Wolf, Ronald (2002), "Complexity Measures and Decision Tree Complexity: A Survey" (PDF), *Theoretical Computer Science*, **288** (1): 21–43, doi:10.1016/S0304-3975(01)00144-X