Decision tree complexity of Boolean functions

Péter Hajnal
JATE, Bolyai Institute
Szeged, Hungary

Suppose we would like to determine the value of a function at an unknown input and we can obtain information only by asking questions of a special form. Depending on the type of functions and questions we consider we obtain several models of computation. These are called decision tree models. The complexity of a computation is the number of questions asked. We investigate some natural problems raised by complexity theory. For example: Exhibit hard functions. How much speed up we can obtain by randomization? We give a survey of the results and open questions about these and similar questions.

0. Introduction

In the *decision tree model* we would like to compute the value of a given function at an unknown input. To do so we collect information on the input by asking questions.

The decision tree model is very suitable for several type of functions. E.g. when the inputs are coming from an ordered set and the output can be the minimal or maximal input element, the median or the sorted order of the input. An other type of suitable functions is where the input is n real values and we want to compute an algebraic function of these inputs ([40], [13], [1]). We will consider Boolean functions. A Boolean function is a function $f : \{0,1\}^n \to \{0,1\}$. Several basic computational tasks lead to computing a Boolean function.

Another "ingredient" of the decision tree model is the type of queries we are allowed to ask. We will investigate several possibilities and so several versions of the decision tree model. All the types we consider will be binary queries, i.e. the possible answers to the query will be 0 and 1.

Finally we must define the way we generate our questions. In the standard interpretation each question asked depends only on the information gained so far. This model is called the deterministic model. We obtain a strengthen model if we allow randomized or nondeterministic generation of the questions. We will define and discuss the corresponding models in the later chapters.

Definition 0.1. A (deterministic) *decision tree* is a rooted binary tree with labels on each node and edge. Each inner node is labeled by a query. One of the two edges leaving the node is labeled 0, the other is labeled 1. The two labels represent the two possible answers to the query. The two subtrees at a node describe how the algorithm proceeds

after receiving the corresponding answer. Each leaf is labeled 0 or 1. These labels give the output, i.e. the value of the function.

Clearly, each truth-assignment to the variables determines a unique path, the *computation path*, from the root to a leaf of the tree. The Boolean function computed by the given decision tree takes the label at this leaf as the value on the given input.

Definition 0.2. Let $\mathrm{cost}(\mathcal{A}, x)$ be the number of queries asked when the decision tree $\mathcal{A}$ is executed on input x. This is the length of the computation path forced by x.

$\max_x \mathrm{cost}(\mathcal{A}, x)$ is the worst case complexity of $\mathcal{A}$, i.e. the depth of the tree.

The *decision tree complexity* of a Boolean function f is $\mathcal{C}(f) = \min_{\mathcal{A}} \max_x \mathrm{cost}(\mathcal{A}, x)$, where the first minimum is taken over all decision trees $\mathcal{A}$ computing the function f.

So the cost of a computation is just the number of queries asked. We ignore the time needed for the generation of queries and the computation of the output. The main topic of this paper how the complexity of a function changes if we vary the model.

1. Deterministic decision trees

1.1. Boolean decision trees.

In the *Boolean decision* tree model we allow questions of the form "What is the value of input x_i?" (shortly "x_i?"). The corresponding complexity measure is $C_b(f)$, where the b subscript stands for Boolean.

It is obvious, that any function $f : \{0,1\}^n \to \{0,1\}$ can be computed by asking n questions. (The underlying tree will be a full binary tree of depth n. The nodes of the i-th level of the tree are labeled by x_i?. It is easy to check that the parity function requires a full binary tree to compute it and only the parity and its negation are the functions with that high complexity.)

In the next paragraphs we discuss the known results on the deterministic Boolean decision tree complexity.

Specific functions.

At the begining of this line of research there were many results stating that a given function has complexity n ([21], [27], [28], [4]). Some of these functions are now standard examples and homework assignments in introductory complexity courses. These examples suggest the following notation. A Boolean function f on n variables is called *evasive* if its decision tree complexity is n. A few example for evasive Boolean functions: parity, majority, graph connectivity, having isolated node in a graph. A good survey on this topic is [7] and chapter VIII. of [5].

Random functions.

Let f be a random Boolean function on n variables, all Boolean functions on n variables being equally likely. Let P be a certain property of Boolean functions. If the probability that a f has property P is p_n and $\lim_{n\to\infty} p_n = 1$, then we say a random Boolean function has property P.

Theorem 1.1.1. (R. Rivest and J. Vuillemin [30]) *A random Boolean function is evasive.*

The theorem suggests that there are so many evasive functions that we should look for a class of functions and prove uniformly that its members are all evasive.

Transitive functions.

The automorphism group of a Boolean function f is the group $Aut(f)$ of those permutations of its variables, that preserve the function. We say that a function is *transitive* if its automorphism group is transitive i.e. for any two variables x and y there is an element $\pi \in Aut(f)$ such that $\pi x = y$. Roughly speaking a function is transitive if we cannot distinguish its variables. This class is quite wide: it includes the symmetric functions and graph properties. The importance of this class is shown by the following theorem.

Theorem 1.1.2. (R. Rivest and J. Vuillemin [30]) *Let n be a prime power, and f be a transitive function on n variables. If $f(0,\ldots,0) \neq f(1,\ldots,1)$, then f is evasive.*

It is known [18] that the theorem becomes false if we do not assume n to be a prime power. A Boolean function is *monotone* if changing the value of a variable from 0 to 1 cannot change the value of the function from 1 to 0. A Boolean function is *non-trivial* if it is not constant. It is still an open problem, whether monotone, non-trivial, transitive Boolean functions are evasive (without any assumption on the number of variables).

Monotone graph properties.

One important subclass of transitive functions is the class of *graph properties*. We can identify graphs with $0-1$-strings of length $\binom{v}{2}$, where v is the number of vertices. The graph properties are Boolean functions $f : \{0,1\}^{\binom{v}{2}} \to \{0,1\}$ taking equal values on isomorphic graphs. Theorem 1.1.2. does not apply here, since $\binom{v}{2}$ is never a prime power if $v > 3$. J. Kahn, M. Saks and D. Sturtevant [19] succeded in proving the analogous theorem. Their proof is based on a topological idea. An input assignment can be considered as the subset of the variables which have value 1. The inputs where the function is 0 give us a set system. If the graph property P is monotone then this set system Δ_P is a *(abstract) simplicial complex*, i.e. $B \subset A \in \Delta$ implies $B \in \Delta$. Evasiveness of P can be approximated by several topological properties of Δ_P. Hence our computational question can be "translated" to a topological problem. Along this line they proved the following theorem.

Theorem 1.1.3. (J. Kahn, M. Saks, D. Sturtevant [19]) *If v is a prime power then every non-trivial monotone graph property on v vertices is evasive.*

It is a central open question whether the theorem remains true when we drop the assumption that the number of vertices is a prime power.

Functions with high symmetry.

There are several other classes of Boolean functions of high symmetry. We mention only the class of bipartite graph properties. The monotone, non-trivial bipartite graph properties are proven to be evasive. A. Yao was who realized that the topological method can be applied without any assumption on the number of vertices in the bipartite graph.

Theorem 1.1.4. (A. Yao [43]) *Every monotone, non-trivial, bipartite graph property is evasive.*

For discussion on digraphs see [19], on directed bipartite graphs see [20], on partially ordered set properties see [9] and [10].

1.2. Linear decision trees.

In the *linear decision tree model* we allow queries of the form "$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \geq \beta$?" The queries of a Boolean decision tree can be expressed in this model, hence any Boolean function can be computed with n queries in the linear decision tree model. The following improvement is due to Gy. Turán [37].

Proposition 1.2.1. *If f is a Boolean function on $n \geq 3$ variables, then $\mathcal{C}_l(f) \leq n - 1$.*

The idea of the proof is that any function on 3 variables can be computed by asking only 2 questions.

Simple counting argument gives a lower bound on the complexity of random functions.

Proposition 1.2.2. *For a random Boolean function f on n variables, $\mathcal{C}_l(f) > n - \alpha \log_2 n$.*

It is still an open problem whether $\mathcal{C}_l(f_n) < n - \alpha' \log_2 n$ is true for all f_n Boolean functions on n variables and some α' constant.

We note that the linear decision tree complexity of connectivity is still unknown (see [12]).

1.3. Miscellaneous models.

Further we mention a model which is stronger than Boolean decision trees, but it is not as strong as linear decision trees. It was introduced by A. Hajnal, W. Maass and Gy. Turán in [14]. They allow questions like "$x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_k} =$", where $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ an arbitrary subset of variables. They called this *generalized decision tree model*. The corresponding complexity measure will be denoted by $C_g(f)$.

In [14] they proved the following theorem.

Theorem 1.3.1. (A. Hajnal, W. Maass, and Gy. Turán [14])
(i) $C_g(Connectivity) = \Theta(v \log v)$,
(ii) $C_g(s - t - Connectivity) = \Theta(v \log v)$,
(iii) $C_g(Bipartiteness) = \Theta(v \log v)$.

2. Randomized decision trees

In the manner common in complexity theory one can introduce decision trees using extra power like nondeterminism, randomization or alternation (see [24], [26], [35], [38]). Now we consider the power of randomization.

2.1. Randomization.

A *randomized decision tree* is a rooted, not necessarily binary, tree. Each of its inner nodes is labeled a variable, i.e. by a query. The edges leaving a node are labeled 0 or 1. The subtrees which can be reached from a given node by an edge labeled 0 are the possible continuations of the algorithm after receiving answer 0. The role of the edges labeled 1 is symmetric. During the execution of the algorithm the next step will be chosen randomly.

An alternative definition might be the following. Let us say that the random choice is based on coin tossing. If one fixes the outcome of the coin tosses than we have a

deterministic computation. In this way we can describe the probabilistic decision tree as a probability distribution on the set of deterministic trees.

We face the question: how to define that a randomized decision tree computes a function?

There are many different ways to answer this questions. We use the simplest convention when we require that the algorithm always give the correct answer. Using the second formalization of the randomized decision tree, it computes a function f iff the distribution is non-zero only on deterministic trees computing f.

Definition 2.1.1. Let $\{ \mathcal{A}_1, \ldots, \mathcal{A}_N \}$ be the set of all the deterministic decision trees computing the function f. Let $\mathcal{R} = \{ p_1, \ldots, p_N \}$ be a randomized decision tree computing f, where p_i is the probability of $\mathcal{A}_i$.

The cost of $\mathcal{R}$ on input x is $\text{cost}(\mathcal{R}, x) = \sum_i p_i \text{cost}(\mathcal{A}_i, x)$.

The *randomized decision tree complexity* of a function f is

$$\mathcal{C}^R(f) = \min_{\mathcal{R}} \max_x \text{cost}(\mathcal{R}, x),$$

where the minimum is taken over all randomized decision trees computing the function f.

There are alternative definitions in which we allow errors. We obtain different models, depending on what kind of errors we allow (1-way or 2-way).

Let $\{ \mathcal{A}_1, \ldots, \mathcal{A}_N \}$ be the set of all the deterministic decision trees (not necessarily computing a given function f). Let $\mathcal{R} = \{ p_1, \ldots, p_N \}$ be a probability distribution on deterministic decision trees, where p_i is the probability of $\mathcal{A}_i$.

$\mathcal{R}$ is λ-*tolerant* for f if $\sum_{\mathcal{A}_i}$ doesn't output $f(x)$ on $x\, p_i \leq \lambda$, for all possible inputs x.

The cost of $\mathcal{R}$ on input x is $\text{cost}(\mathcal{R}, x) = \sum_i p_i \text{cost}(\mathcal{A}_i, x)$.

The *2-way error randomized decision tree complexity* of a function f with error λ is

$$\mathcal{C}^{R2}_\lambda(f) = \min_{\mathcal{R}} \max_x \text{cost}(\mathcal{R}, x),$$

where the minimum is taken over all λ-tolerant randomized decision trees computing the function f.

Let $\mathcal{C}^{R2}(f) = \mathcal{C}^{R2}_{\frac{1}{3}}(f)$.

The constant $\frac{1}{3}$ doesn't have an important role. If we neglect constants in the complexity than we can substitute it with anything less than $\frac{1}{2}$.

The possible algorithms can output anything. The mistake can be either way. This fact is indicated by the superscript 2. If our randomized algorithm is restricted to produce deterministic trees where the mistake occurs in only one direction (it might output 0 instead of the real value 1 but not the other way around) then it is called *1-way error computation* (the corresponding complexity measure is denoted by $\mathcal{C}^{R1}$). For further information we refer the reader to [38], [29] and [34].

The main question is this: how much can we save by adding the extra power of randomization?

2.2. Boolean decision trees.

First we mention some basic inequalities on the relation between deterministic and randomized complexity.

Theorem 2.2.1. (M. Blum [3]) *For any Boolean function f*

$$\sqrt{\mathcal{C}_b(f)} \leq \mathcal{C}_b^R(f) \leq \mathcal{C}_b(f).$$

Using the $\mathcal{C}^{R1}(f)$, resp. $\mathcal{C}^{R2}(f)$ notation for the randomized complexity of f allowing 1-way and 2-way errors, resp. Noam Nisan obtained the following results.

Theorem 2.2.2. (Noam Nisan [29]) *For any Boolean function f*
(i) $\sqrt{\frac{1}{2}\mathcal{C}_b(f)} \leq \mathcal{C}_b^{R1}(f)$,
(ii) $\frac{1}{2}\sqrt[3]{\mathcal{C}_b(f)} \leq \mathcal{C}_b^{R2}(f)$.

These theorems give a lower bound for the power of randomization. We refer to them as the basic bounds.

Transitive functions.

There are several known examples of transitive functions where randomization does help.

Example 2.2.3. (Snir [35]) Let f be the following Boolean function on $n = 2^d$ variables. First let us build a binary tree based on these variables as leaves. Plug a NAND gate into each inner node. The circuit that we get in this way will compute f.

It is not hard to see that the deterministic complexity of this function is n (see Theorem 1.1.2.). However, there is a randomized algorithm which computes f faster on average. Choose a child of the root at random and evaluate its subtree recursively. If it evaluates to 0, then the value of f is 1. Otherwise recursively evaluate the other child of the root.

The complexity of this algorithm is $\Theta(n^\alpha)$, where $\alpha = \log_2\left(\frac{1+\sqrt{33}}{4}\right) = 0.753\ldots$. As it turns out this is exactly the randomized complexity of f. For more details see [33].

R. Boppana exhibited another example of a function where randomized and deterministic complexities differ in the exponent (this construction is described also in [33]).

It is conjectured that the 2.2.3. example above are the best possible up to a constant factor.

Conjecture 2.2.4. (M. Saks and A. Wigderson [33]) *For any Boolean function f*

$$\mathcal{C}_b^R(f) = \Omega(\mathcal{C}_b(f)^{0.753\ldots}).$$

Graph properties.

Example 2.2.5. (M. Saks and A. Wigderson [33]) Consider the digraph property "every vertex has an incoming arc".

Deterministically, this is an evasive property, so its deterministic complexity is $v(v-1)$.

Let us examine the following randomized algorithm. It considers each vertex one at a time in random order and it scans the possible incoming edges into that vertex until it finds one or realizes that there aren't any. It is easy to see that the complexity of this algorithm is at most $\frac{v(v+1)}{2}$. So randomization can save a constant factor.

The analog graph property example is "having isolated node". The undirected version of the algorithm above gives a constant saving although the analysis of the algorithm is more complex. Up to now these are the most effective savings.

Conjecture 2.2.6. (attributed to R.M. Karp by [33]) *For any non-trivial, monotone graph property P*

$$\mathcal{C}_b^R(P) = \Omega(\mathcal{C}_b(P)) = \Omega(v^2).$$

Only in the case of graph properties are there results better than the basic inequalities known (Theorem 2.2.1.). (In this case we know that the deterministic complexity is of the order of v^2. Blum's bound shows that the randomized complexity of any graph property is at least linear in v.) The first step to prove a non-trivial lower bound was done by A. Yao [42] who proved an $\Omega(v \log^{\frac{1}{12}} v)$ lower bound on the randomized deciosion tree complexity of any non-trivial, monotone graph property. Later this lower bound was improved to $\Omega(v^{\frac{5}{4}})$ by V. King [20]. So far the best improvement is the following.

Theorem 2.2.7. (P. Hajnal [15]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}_b^R(P) = \Omega(v^{\frac{4}{3}}) = \Omega(\mathcal{C}_b(P)).$$

There is a little progress when we assume that the graph property is "G has a certain subgraph". In this case H.D. Gröger proved [11] an $\Omega(v^{\frac{3}{2}})$ lower bound.

Functions with other symmetries.

One can consider several other symmetries like 3-uniform set system properties or partially ordered set properties. Very little is known about these questions.

2.3. Linear decision trees.

We mention only one result. It gives an $\Omega(n)$ lower bound on the randomized complexity of the inner product mod 2 of two n-bit vectors.

Theorem. *2.3.1.* (H.D. Gröger and GY. Turán [12])

$$\mathcal{C}_l^R(\textit{Inner product mod } 2_n) = \Omega(n).$$

Unfortunately, very little is known about the randomized linear decision tree complexity of other functions, for example of graph properties.

3. Nondeterministic decision trees

3.1. Nondeterminism.

Definition 3.1.1. A *nondeterministic decision tree* is a rooted tree. Each of its inner nodes is labeled by a variable. This label represents a query. Each edge leaving the node is labeled 0 or 1. The subtrees which can be reached from a given node by an edge labeled 0 are the possible continuations of the algorithm after getting answer 0. The role of the edges labeled by 1 is symmetric. During the execution of the algorithm the next step will be chosen nondeterministically.

The definition above describes the notion of a nondeterministic decision tree and its execution on an input. But this execution is nondeterministic. So what function is computed by this tree? We say that an input is *accepted* if there exists a computation path leading to a leaf labeled 1. The *function f is computed by a nondeterministic decision tree* when $f(x) = 1$ if and only if x is accepted.

Definition 3.1.2. The *nondeterministic decision tree complexity* of a Boolean function f is the minimum depth of the nondeterministic decision trees computing f. This complexity is denoted by $\mathcal{C}^{ND}(f)$.

3.2. Boolean decision trees.

Let eq be the 'equality' function on the variables $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$ i.e. $eq = (x_1 = y_1) \wedge (x_2 = y_2) \wedge \ldots \wedge (x_n = y_n)$. Then the nondeterministic complexity of eq is $2n$ while the $\neg eq$ has complexity 2. We can make the nondeterministic decision tree complexity notion to be symmetric by considering the $\tilde{\mathcal{C}}_b^{ND}(f) = \max\{\mathcal{C}_b^{ND}(f), \mathcal{C}_b^{ND}(\neg f)\}$ complexity measure.

The nondeterministic complexity of a function f can be expressed the following way.

Definition 3.2.1. 1-*certificate* of a Boolean function is a partial assigment to its variables that forces the value of the function to be 1. 0-*certificate* is a partial assigment that forces the value to be 0. The size of a certificate is the size of the domain of the partial assignment. *The certificate complexity of f on an input w*, $cert_w(f)$ is the size of the smallest certificate that agrees with w. The *certificate complexity* of f, $cert(f)$ is the maximum of $cert_w(f)$ over all w inputs.

It is easy to check that this is a simple reformulation of nondeterministic Boolean decision tree complexity i.e $\tilde{\mathcal{C}}_b^{ND}(f) = cert(f)$. The basic relation between deterministic and nondeterministic Boolean decision tree complexity was discovered independently by several people.

Theorem 3.2.2. ([3], [36])

$$\tilde{\mathcal{C}}_b^{ND}(f) \leq \mathcal{C}_b(f) \leq (\tilde{\mathcal{C}}_b^{ND}(f))^2.$$

Next we discuss the basic lower bound technique for the nondeterministic complexity. First we introduce a useful notion and its generalization by N. Nisan.

Definition 3.2.3. Let f be a Boolean function and w is an input string. We say that f *is sensitive to the i-th variable on w* if $f(w) \neq f(w^{(i)})$, where $w^{(i)}$ is the input that we obtain from w by changing the i-th input bit. *The sensitivity of f on w* is $s_w(f)$, the number of variables f is sensitive to on the input w. The *sensitivity* of f, $s(f)$ is the maximum of the $s_w(f)$'s over all w inputs.

Definition 3.2.4. Let f be a Boolean function and w is an input string and S is a subset of the variables. We say that f *is sensitive to S on w* if $f(w) \neq f(w^S)$, where w^S is the input that we obtain from w by changing the value of the variables in S. The *block sensitivity of f on w*, $bs_w(f)$ is the maximum number b such that there exists disjoint subsets of variables $S_1, \ldots, S_b$ such that f is sensitive to S_i $(i = 1, 2, \ldots, b)$ on the input w. The *block sensitivity* of f, $bs(f)$ is the maximum of the $bs_w(f)$'s over all w inputs.

The following proposition says that these notions give a lower bound on the nondeterministic Boolean complexity.

Propsition 3.2.5. (N. Nisan [29])

$$s(f) \leq bs(f) \leq \tilde{C}_b^{ND}(f).$$

An important question is how good these lower bounds are. The block sensitivity is proven to be 'close' to the nondeterministic Boolean decision tree complexity.

Theorem 3.2.6. (N. Nisan [29])

$$\sqrt{\tilde{C}_b^{ND}(f)} \leq bs(f) \leq \tilde{C}_b^{ND}(f).$$

The same question corresponding to the sensitivity is still open. The biggest gap between the sensitivity and block sensitivity of a function is quadratic [32].

4. Conclusion

We summarized the basic notions, results and open problems related to the decision tree complexity of Boolean functions. The decision tree model is a very simple model of computation. But even the most basic questions are far from being solved. We hope that the simplicity of the model gives us a chance to develope a "theory" and answer the natural questions suggested by complexity theory. A similar program for more general models of computation seems extremely hard and hence unrealistic in the near future.

Acknowledgement. The author would like to thank L. Csirmaz for fruitful discussions and helpful suggestions

References

[1] M. Ben-Or, Lower bounds for algebraic computational trees, *Proc. 15th ACM STOC* (1983), 247–248.

[2] M. R. Best, P. van Emde Boas and H. W. Lenstra, Jr., A sharpened version of the Aanderaa−Rosenberg conjecture *Report ZW 30/74, Mathematish Centrum*, Amsterdam (1974).

[3] M. Blum and R. Impagliazzo, Generic oracles and oracle classes, *Proc. 28th IEEE FOCS* (1987), 118–126.

[4] B. Bollobás, Complete subgraphs are elusive, *J. Combinatorial Theory Ser. B* **20**(1976), 1–7.

[5] B. Bollobás, *Extremal Graph theory*, Academic Press, London, 1978.

[6] B. Bollobás, *Random graphs*, Academic Press, London, 1985.

[7] B. Bollobás and S. E. Eldridge, Packing of graphs and applications to computational complexity, *J. of Combinatorial Theory Ser. B* **25**(1978), 105–124.

[8] M. Dietzfelbinger and W. Maass, Two lower bound arguments with 'inaccessible' number, Structure in Complexity Theory, Lecture Notes in Computer Science, **223**, Springer, Berlin - New york, 1986, 163–183.

[9] U. Faigle and Gy. Turán, The complexity of interval orders and semiorders, *Discrete Math* **63**(1987), 131–141.

[10] U. Faigle and Gy. Turán, Sorting and recognition problems for ordered sets, *SIAM J. Comput.* **17**(1988), 100–113.

[11] H. D. Gröger, On the randomized complexity of monotone graph properties, submitted for publication.

[12] H. D. Gröger and Gy. Turán, On linear decision trees computing Boolean functions, *University of Illinois at Chicago, Research report in computer science,* No. 44, September 1990.

[13] E. Győri, An n-dimensional search problem with resticted questions, *Combinatorica* **1**(1981), 377–380.

[14] A. Hajnal, W. Maas and Gy. Turán, On the communication complexity of graph properties, *Proc. 20th ACM STOC* (1988), 186–191.

[15] P. Hajnal, The complexity of graph problems, *Ph.D. Thesis, University of Chicago, TR88-13.*

[16] P. Hajnal, On the power of randomness in the decision tree model, *Proc. of 5th Structure in Complexity Theory* (1990), 66–77.

[17] P. Hajnal, An $\Omega(n^{\frac{4}{3}})$ lower bound on the randomized complexity of graph properties, *Combinatorica* **11**(1991), 131–143.

[18] N. Illies, A counterexample to the generalized Aanderaa−Rosenberg conjecture, *Info. Proc. Letters* **7**(1978), 154–155.

[19] J. Kahn, M. Saks and D. Sturtevant, A topological aproach to evasiveness, *Combinatorica* **4**(1984), 297–306.

[20] V. King, Lower bounds on the complexity of graph properties, *Proc. 20th ACM STOC* (1988), 468–476.

[21] D. Kirkpatrick, Determining graph properties from matrix represantation, *Proc. 6th SIGACT Conf.* (1974), 84-90.

[22] D. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm, *SIAM J. Comput.* **15**(1986), 287–299.

[23] D. J. Kleitman and D. J. Kwiatkowski, Further results on the Aanderaa−Rosenberg conjecture, *J. Combinatorial Theory* **28**(1980), 85–95.

[24] U. Manber and M. Tompa, The complexity of problems on probabilistic, nondeterministic and alternating decision trees, *J. ACM* **32**(1985), 732–740.

[25] F. Meyer auf der Heide, Fast algorithms for n-dimensional restrictions of hard problems, *J. Assoc. Comput. Mach.* **35**(1988), 185–203.

[26] F. Meyer auf der Heide, Non-deterministic versus probabilistic linear search algorithms, *Proc. 26th IEEE FOCS* (1985), 65–73.

[27] E. C. Milner and D. J. A. Welsh, On the computational complexity of graph theoritical properties, *Univ. of Calgary, Res. Paper No.232* 1974.

[28] E. C. Milner and D. J. A. Welsh, On the computational complexity of graph theoritical properties, Proc. Fifth British Combinatorial Conf. (ed: C.St.J.A. Nash-Williams and J. Sheehan), Utilitas Math., Winnipeg, Ontario, Canada, 1976, 471-487.

[29] N. Nisan, CREW PRAMs and decision trees, *Proc. 21th ACM STOC* (1989), 327–335.

[30] R. Rivest and S. Vuillemin, On recognizing graph properties from adjacency matrices, *Theor. Comp. Sci.* **3**(1976), 371–384.
[31] A. L. Rosenberg, On the time required to recognize properties of graphs: A problem, *SIG ACT News* **5**(1973), 15–16.
[32] D. Rubinstein, personal communication.
[33] M. Saks and A. Wigderson, Probabilistic Boolean decision trees and the complexity of evaluating game trees, *Proc. 26th IEEE FOCS* (1986), 29–38.
[34] M. Santha, On the Monte Carlo Boolean Decision Tree Complexity of Read-Once Formulae, manuscript, 1991.
[35] M. Snir, Lower bounds for probabilistic linear decision trees, *Theor. Comp. Sci.* **38**(1985), 69–82.
[36] G. Tardos, Query complexity, or Why is it difficult to seperate $NP^A \cap coNP^A$ from P^A by a random oracle A, manuscript, 1987.
[37] Gy. Turán, personal communication.
[38] A. Yao, Probabilistic computation: towards a unified measure of complexity, *Proc. 18th IEEE FOCS* (1977), 222-227.
[39] A. Yao, Some complexity questions related to distributed computations, *Proc. 11th ACM STOC* (1979), 209–213.
[40] A. Yao, A lower bound to finding convex hulls, *J. ACM* **28**(1981), 780–789.
[41] A. C. Yao, Lower bounds by probabilistic arguments, *Proc. 24th IEEE FOCS* (1983), 420–428
[42] A. Yao, Lower bounds to randomized algorithm for graph properties, *Proc. 28th IEEE FOCS* (1987), 393–400.
[43] A. Yao, Monotone bipartite graph properties are evasive, *SIAM J. Comput.* **17**(1988), 517–520.