Search Problems in the Decision Tree Model

László Lovász*
Moni Naor†
Ilan Newman‡
Avi Wigderson§

March 8, 1994

Abstract

We study the relative power of determinism, randomness and nondeterminism for search problems in the Boolean decision tree model. We show that the gaps between the nondeterministic, the randomized and the deterministic complexities can be arbitrary large for search problems. We also mention an interesting connection of this model to the complexity of resolution proofs.

Boolean functions, decision trees
68R05, 94A15, 68Q10.

1 Introduction

Ramsey's theorem asserts that every graph on n vertices has either a complete graph or an independent set of size $\frac{1}{2}\log n$. A natural search problem associated with this theorem is to find such a subgraph. Many other problems, like the ones below, have a similar flavor: Given an assignment of n pigeons into $n-1$ pigeonholes, find two pigeons assigned to the same hole. Given a k-chromatic graph and a coloring of its nodes with fewer than k colors, find two neighbors which have the same color. Given an unsatisfiable 3-CNF formula and an assignment to its variables, find a clause which is not satisfied.

How hard is it to solve such search problems? The answer depends of course on their representation and the computational model. We assume that the input is encoded in binary, and that we are only allowed to probe input bits. This gives the familiar Boolean decision tree

*Princeton University and Eötvos Loránd University Budapest
†Weizmann Institute. Part of this work was done while the author was at the IBM Almaden Research Center.
‡DIMACS center and Hebrew U. Jerusalem
§Princeton University and Hebrew U. Jerusalem

model, adapted to solving search problems rather than computing Boolean functions. We study the relationship between the standard nondeterministic, probabilistic and deterministic variants of this model, and discover that it is drastically different from the case of function computation, where all three measures are polynomially related (see [2, 7, 19, 14]). In all the examples listed above it is easy to guess and verify the solution; hence the nondeterministic decision tree complexity is small (a constant or polylog). If the decision tree was computing a function, this would imply that both the randomized and deterministic complexities are small, by the fact that the deterministic complexity (and thus the randomized too) is at most the square of the nondeterministic complexity [2, 7, 19]. It turns out that for search problems these gaps can be arbitrary large.

Our investigation is partly motivated by a similar study of search problems in the communication complexity setting in the work of Karchmer and Wigderson [11] and Raz and Wigderson [18], where a similar phenomenon occurs but not to the same extent. Another study of search problems was carried out by Papadimitriou [15] where complexity classes defined by search problems were investigated. Some of our examples are inspired by this work and [8, 10].

The examples above may remind the readers of resolution proofs. Indeed, resolution proofs viewed top down yield Boolean decision trees to the search problems above (this fact seems to be folklore and is elaborated in the appendix).

Thus, the exponential length lower bounds on resolution proofs (see e.g. [4]) provide linear deterministic lower bounds even when the nondeterministic complexity is a constant. However, the distinction between the resolution and the decision tree points of view becomes clear when trying to make sense out of the probabilistic model. In resolution, one proves simultaneously that every assignment falsifies some clause in an unsatisfiable formula. This has no natural probabilistic analog. The decision tree approach, where one must find for a given input assignment a clause that it falsifies, has a natural randomized version. Indeed, we will be primarily concerned with the power of probabilistic computation for such problems.

We prove that the probabilistic complexity can be at both ends of the spectrum. We give an explicit search problem for which the probabilistic (and nondeterministic) complexity is constant, but the deterministic is linear. On the other hand we provide two explicit problems for which there is a large gap between the nondeterministic and probabilistic complexities: one in which the first is constant and the second is $\Omega(n^{\frac{1}{6}})$, and another in which the first is $O(\log n)$ while the second is nearly linear. Finally, we present an explicit problem for which there is a simultaneous exponential gap between the nondeterministic versus the randomized and the randomized versus the deterministic complexities. This last example uses an upper bound due to Irani [9] on online coloring algorithms to provide a lower bound on the deterministic complexity. We note here that as far as we know, no such simultaneous gaps are known for any other model of computation.

The special case of nondeterministic complexity 2 deserves a special interest. It corresponds to unsatisfiable 2-CNF formulae. We characterize the deterministic complexity by the structure of the formula, and note that it can never exceed $1 + \log n$. We show that here too the gap between the randomized complexity and the nondeterministic, and the gap between the randomized complexity and the deterministic can be arbitrarily large.

The paper is organized as follows; in section 2 we give the formal definitions of search problems, decision trees and show that the CNF search problem is 'complete' for all the variants of decision trees. In section 3 we construct search problems with a large gap between the deterministic and the randomized complexity, randomized and nondeterministic complexity, and simultaneous nondeterministic - randomized - deterministic complexity gaps. In section 4 we discuss the special case of nondeterministic complexity 2. The exact relationship to the resolution problem appears in the appendix.

2 Definitions

A search problem is specified by n variables and a collection of 'witnesses'. In addition, this collection must have the property that every assignment to the n variables is associated with at least one witness. Given an input string which is an assignment to the n variables, the goal is to find a witness consistent with it. Formally,

Definition 2.1 *A search problem on n variables is a relation $F \subseteq \{0,1\}^n \times W$, such that $\forall x \in \{0,1\}^n$, $\exists w \in W$ for which $(x,w) \in F$. W is a finite set, called the set of witnesses. The goal of the search for input $x \in \{0,1\}^n$, is to find a $w \in W$ such that $(x,w) \in F$, (we call such a w a valid witness for x).*

A special class of search problems come from DNF tautologies. A monomial m is a conjunction of literals (over the variable set X). Let g be a DNF tautology with a set of monomials M. That is, $g = \vee_{m \in M} m$, and so that g is true for every assignment to the variables. g naturally defines a search problem on $\{0,1\}^X \times M$: Each input (a truth assignment to the variables) is associated with all monomials $m \in M$ that are satisfied by it.

The class of search problems defined by DNF tautologies is quite general. Indeed as we shall see, every search problem F gives rise to a search problem on an associated DNF tautology, $g(F)$, that will turn out to be equivalent to F.

With every monomial m we associate the subcube C_m of all the points in $\{0,1\}^X$ that are consistent with m (i.e evaluate to '1' on m).

Definition 2.2 *Let $F \subseteq \{0,1\}^n \times W$ be a search problem. The set of monomials associated with F, denoted by M_F is defined by $m \in M_F$ if $\exists w \in W$ such that $\forall x \in C_m$, $(x,w) \in F$. In words, $m \in M_F$ if all inputs x that are consistent with m share a mutual witness w.*

Claim 2.1 *Let $F \subseteq \{0,1\}^n \times W$ be a search problem and let $g(F)$ be the formula defined by $g(F) = \vee_{m \in M_F} m$, then $g(F)$ is a DNF tautology.*

Proof: Every input x has a witness in W. Thus, in particular, the subcube that contains the single point x defines a monomial $m \in M_F$ that is a witness for x in F'. $\square$

Thus $g(F)$ defines a valid DNF search problem $F' \subseteq \{0,1\}^n \times M_F$, where F' is the set of pairs (x,m) such that x satisfies m. It will be convenient though, for historical reasons, to consider $f(F) = \overline{g(F)}$ which is an unsatisfiable CNF formula with the associated search problem: For each input (an assignment to the variables) find an unsatisfied clause. Observe that there is a natural correspondence between witnesses of F and F'.

2.1 Decision tree complexity for search problems

Let $F \subseteq \{0,1\}^n \times W$ be a search problem. A deterministic decision tree for F is an algorithm that may query the value of an input bit at each step. The goal is to find a consistent witness. Formally, such an algorithm is a rooted binary tree in which every internal node is labeled by a variable and the two outgoing edges are labeled by the two possible values to that variable. Each leaf is labeled with a witness $w \in W$. Every assignment of the variables determines a path from the root to a leaf in a natural way. The tree is a valid decision tree, if for every assignment this path ends in a leaf labeled by a valid witness.

The *deterministic* complexity of F, $D(F)$, is the minimum depth of any decision tree for F.

The *nondeterministic* complexity of F, $N(F)$, is the minimum number of variables that must be probed in order to find a valid witness for the worst case input. Alternatively, it is exactly the maximum size, over all inputs x, of the smallest monomial in M_F that is satisfied by x.

A randomized decision tree for F is a distribution over deterministic decision trees for F. The complexity of a randomized decision tree is the expected path length for the worst case input. The *randomized* complexity of F, $R(F)$, is the minimum over all randomized decision trees for F.

Facts:

1. Let $F \subseteq \{0,1\}^n \times W$ be a search problem and $F' \subseteq \{0,1\}^n \times M_F$ the associated search problem. Then $D(F) = D(F')$, $R(F) = R(F')$ and $N(F) = N(F')$. Furthermore, there is a natural correspondence between any decision tree for F (deterministic, randomized or nondeterministic) and a (corresponding) decision tree for F'. Thus F and F' are indeed restatement of essentially the same problem. Throughout the paper we will not distinguish between F and its associated F'.

2. For every decision problem $F: \ N(F) \leq R(F) \leq D(F)$.

An observation of Chvatal and Szemerédi [5] is that for a search problem F, lower bounds on the (regular) resolution process for the unsatisfiable formula $f(F)$ imply lower bounds on the deterministic decision tree complexity for F. We elaborate on that point in the appendix.

3 The relative power of determinism and randomization versus non-determinism

In this section we present some explicit search problems for which there are large gaps between the different decision tree complexity measures. Our main task is to construct search problems for which $N(F) << D(F)$, $R(F) << D(F)$ or $N(F) << R(F) << D(F)$ simultaneously. Another parameter to consider in each case is $D(F)$ versus the number of variables n, which is the obvious upper bound for all the three measures of complexity.

3.1 Gaps between R(F) and D(F)

We present here an *explicit* search problem for which $R(F) = O(1)$ and $D(F) = \Omega(n)$. Note that the existence of such a problem follows from [4] by probabilistic arguments.
Let $G(U, V, E)$ be a bipartite graph. Define the search problem $\mathbf{DEG}(G)$ on $|E|$ variables in the following way. Each 0-1 assignment to the variables is interpreted as a subgraph G' of G, defined by those edges that are assigned '1'. The search problem is to find a vertex r whose degree in G' is not one. Clearly if the sides of the graph are not equal ($|V| \neq |U|$) such a vertex exists for every subgraph G', thus as long as the sides are not equal, $DEG(G)$ is a valid search problem.

Lemma 3.1 *Let $G = (U, V, E)$ a bipartite graph with with maximum degree d and $|U| = 2n$, $|V| = n$, then $N(DEG(G)) \leq d$ and $R(DEG(G)) \leq 2d(d+1)$.*

Proof: The nondeterministic complexity, $N(DEG(G)) \leq d$ since for every input (subgraph G') one must only check the incident edges of the guessed vertex.

Consider the following random decision tree. Pick at random a vertex $u \in U$ and independently a vertex $v \in V$ ask all edges that are incident to each of the two vertices (i.e $2d$ edges are being checked). If u or v produce a witness stop, otherwise repeat this process until done.

We claim that the probability that a witness is discovered in each iteration is at least $\frac{1}{d+1}$: If there are more than $\frac{2nd}{d+1}$ edges in the subgraph G' defined by the '1'-edges, then at least $\frac{n}{d+1}$ of the vertices in V are of degree at least 2. In this case the fact that $v \in V$ is chosen at random proves the claim. If, on the other hand, G' has less than $\frac{2nd}{d+1}$ edges, then at least $\frac{2n}{d+1}$ of the vertices in U are of degree 0 in G'. Thus, the fact that $u \in U$ is chosen at random proves the claim in this case. We conclude that the expected number of iterations is $d+1$, in each of them $2d$ edges are probed which yields the above upper bound. $\Box$

We will show now an infinite sequence of bipartite graphs for which the deterministic complexity is $\Omega(n)$.

Let $G = (U, V, E)$ be a bipartite graph with maximum degree d, $|U| = 2n$, $|V| = n$ as before, and with the additional 'expansion' property: For every $S \subset U$, $|S| \leq n/4 \Rightarrow |N(S)| \geq 2|S|$. ($N(S) = \{v| \ (u, v) \in E \ and \ u \in S\}$).

Such a graph exists for large enough d and infinitely many $n's$ and can be efficiently constructed using expander graphs [13] (d can be taken to be 30).

Theorem 3.1 *Let G be a graph as defined above then $R(DEG(G)) = O(1)$ and $D(DEG(G)) = \Omega(n)$.*

Proof: The fact that $R(DEG(G)) = 2d(d+1) = O(1)$ was already proved in Lemma 3.1.

We show an adversary strategy that is going to cause any deterministic decision tree to probe $\Omega(n)$ edges. The adversary will be limited to produce a subgraph for which $\forall v \in V,\ deg_{G'}(v) = 1$ and $\forall u \in U,\ deg_{G'}(u) \leq 1$. Thus, the answer the decision tree has to find is a vertex in U.

We need some definitions. For any $S \subseteq U$ and subgraph G' of G, $N_{G'}(S) = \{v \in V \mid (u,v) \in G', u \in S\}$. For stage i (after i edges were probed) let $E_i' = \{e|e \text{ was assigned '0' }\}$ and $E_i" = \{e|\ e \text{ was not probed and } \exists e',\ e' \text{ assigned '1' and } e \cap e' \neq \emptyset\}$. Define $G_i = G - (E_i' \cup E_i")$. In words, G_i contains all the edges that are still possible for the adversary to use in its final subgraph without violating the above limitation.

For each $S \subseteq U$, define $N_i(S) = N_{G_i}(S)$. For any subgraph G' of G, define $S \subset U$ to be unmatchable if $N_{G'}(S) < |S|$. Let $S^*_{G'}$ denote a minimum cardinality unmatchable set in G'. Finally call $S^*_i = S^*_{G_i}$ a minimal unmatchable set in step i.

By the above limitation on the adversary, at step i the subgraph G_i contains a partial matching from U to V. The decision tree cannot know the answer as long as there is no isolated vertex in G_i. Obviously such a vertex is, by itself, a minimum unmatchable set. Initially, by the definition of the graph, $|S^*_0| \geq n/4$. The strategy of the adversary is to make sure that the minimum unmatchable set size does not decrease too fast. Formally, in step i an edge $e = (u,v),\ u \in U,\ v \in V$ is probed. The adversary computes

1. $S^0(e) = S^*(G_i - e)$ i.e the minimum unmatchable set that occurs on '0' answer on e.
2. $S^1(e) = S^*(G_i - \{f = (x,v)|\ f \text{ was not probed and } x \in U\})$, i.e the minimum unmatchable set that occurs on '1' answer on e.

He then chooses the answer on e so as to make S^*_{i+1} the larger of $S^0(e), S^1(e)$.

The heart of the argument is the following claim.

Claim: If $|S^*_{i+1}| = s$ then there is a minimal unmatchable set S^*_i with $|S^*_i| \leq 2s$.

Proof: (of the claim) Assume e is asked in step $i+1$. By the above strategy, $|S^*_{i+1}| = \max(|S^0(e)|, |S^1(e)|)$. It is easy to see that $S = S^0(e) \cup S^1(e)$ cannot be matched into V in G_i. Thus, S contains an unmatchable set for step i of cardinality no more than $|S^0(e) \cup S^1(e)| \leq 2 \cdot \max(|S^0(e)|, |S^1(e)|) = 2s$. $\square$

We can now complete the proof of Theorem 3.1 by the following argument. At the beginning $|S^*_0| \geq n/4$, at the end $|S^*_i| = 1$ and by the claim the cardinality of the minimal unmatchable set does not decrease by more than a factor of 2. We conclude that at some step j, $n/16 \leq |S^*_j| \leq n/8$, with $|N_j(S^*_j)| < |S^*_j|$. However, by the expansion property of G, $|N_G(S^*_j)| \geq 2|S^*_j|$. Since at each step, $N_i(S)$ can drop by at most d for any set S, at least $|S^*_j|/d = \Omega(n)$ edges were probed up to step j. $\square$

3.2 Gaps between $N(F)$ and $R(F)$

In this section we construct two search problems for which the randomized complexity is large while the nondeterministic complexity is small. The nondeterministic complexity of the first problem is constant while its randomized complexity is $\Omega(n^{\frac{1}{6}})$. The second problem has $O(\log n)$ nondeterministic complexity and its randomized complexity is $\Omega(n/\log n)$. The proof of the lower bound on the randomized complexity of the first problem is by proving a lower bound on the distributional complexity. (Yao [21] has shown that this is sufficient.) The proof for the second is by an indirect reduction to a communication complexity game.

3.2.1 A problem with $N(F) = 3$ and $R(F) \in \Omega(n^{\frac{1}{6}})$

Let **GRID** be the following problem on $n = m^2 - 1$ variables. Consider an $(m+1) \times (m+1)$ matrix where the entry at the bottom left corner contains a one and the top row and rightmost column are all zero. The n input bits determine the rest of the entries of the matrix.

For any 0-1 assignment to the rest of the matrix, the goal is to find an entry that is one and its upper and right neighbors are zero. It is not hard to see that such a configuration always exists.

This example is inspired by the lower bound argument of Hirsch, Papadimitriou and Vavasis [8] for finding Brouwer fixed points and discussions with Noga Alon on extending it to the random case.

Theorem 3.2 *$N(GRID) = 3$, $R(GRID) = \Omega(n^{\frac{1}{6}})$ and $D(GRID) = \Theta(\sqrt{n})$.*

Proof: The fact that $N(GRID) = 3$ is clear. The theorem is established by the following lemmata.

Lemma 3.2 *$R(GRID) = \Omega(n^{\frac{1}{6}})$*

Proof: A basic result of Yao [21] asserts that in order to prove lower bounds on randomized decision tree complexity it is sufficient to show a distribution on the inputs such that any deterministic algorithm requires a high expected number of queries. The distribution for which we claim the lower bound is defined as follows: A random upward and rightward path starting from the bottom left corner of the matrix and ending at the top row or right column is picked uniformly from all such paths. The entries along the path receive the value '1' and the rest receive the value '0'.

We claim that any deterministic algorithm requires $\Omega(m^{\frac{1}{3}}) = \Omega(n^{\frac{1}{6}})$ queries on the average to discover the end point of the path, which is the only point where the desired configuration occurs. We need the following claim.

Claim 3.1 *Let $A, B, C_1, .., C_k$ be points in the matrix such that B is of Manhattan distance at least d from A in the downward and left direction, and each of C_i, $1 \leq i \leq k$ is of Manhattan distance at least d from B.*

For the above distribution on paths, and $1 \leq k \leq \frac{\sqrt{d}}{2}$ the probability that a path passes through A given that it passes through B and avoids $C_1, .., C_k$ is at most $\frac{2}{\sqrt{d}}$.

Proof (of the claim)
Let $\mathcal{A}$ be the event that the path passes through point A, $\mathcal{B}$ the event that it passes through B and $\mathcal{C}_i$, $1 \leq i \leq k$ the event that it passes through C_i.

$$Prob(\mathcal{A}|\mathcal{B}, \overline{\mathcal{C}_1}, .., \overline{\mathcal{C}_k}) = \frac{Prob(\mathcal{A} \cap \overline{\mathcal{C}_1} \cap, .., \cap \overline{\mathcal{C}_k}|\mathcal{B})}{Prob(\overline{\mathcal{C}_1} \cap, .., \cap \overline{\mathcal{C}_k}|\mathcal{B})} \leq \frac{Prob(\mathcal{A}|\mathcal{B})}{1 - Prob((\mathcal{C}_1 \cup, .., \cup \mathcal{C}_k)|\mathcal{B})} \leq$$

$$\leq \frac{Prob(\mathcal{A}|\mathcal{B})}{1 - k \cdot \max_{1 \leq i \leq k}\{(Prob(\mathcal{C}_i|\mathcal{B})\}} \leq \frac{\frac{1}{\sqrt{d}}}{1 - \frac{k}{\sqrt{d}}} \leq \frac{2}{\sqrt{d}}$$

($Prob(\mathcal{A}|\mathcal{B})$ and $Prob(\mathcal{C}_i|\mathcal{B})$ are bounded by $\frac{1}{\sqrt{d}}$ since they correspond to $\max_{1 \leq j \leq d'} \frac{\binom{d'}{j}}{2^{d'}}$).
$\square$

Let T be any deterministic decision tree for $GRID$. We will give additional information to queries of T; At any stage i we provide T with a prefix of the path. Informally, the intuition is the following, at each stage a contiguous initial segment of the path will be known to T. If the next query is to a point of distance not too far from that known prefix, we provide T with an additional segment of the path. A segment that is long enough to make sure that knowing it will determine the value of the query. Thus, the conditional probability of future queries will not depend on the answer to that query. If on the other hand, T queries a point 'far away' from the prefix of the path that he already know, then there is a good chance that he will get a '0' answer and learn very little (by Claim 3.1).

Formally, at each stage i we provide T with the prefix of the path of length $j_i m^{2/3}$, i.e. with all the points on the path up to Manhattan distance $j_i m^{2/3}$ from the origin (the bottom left corner). The length of the prefix is determined as follows: We start with $j_0 = 0$. If by the ith stage $\ell \geq 0$ is the largest number such that there exists a set of ℓ queries $q_1, q_2, \ldots q_\ell$ so that for all $1 \leq h \leq \ell$ q_h is to a point of distance D_h with $(j_{i-1} + h - 1)m^{2/3} + 1 \leq D_h \leq (j_{i-1} + h)m^{2/3}$ from the origin, then we provide all the points on the path up to distance $(j_{i-1} + \ell)m^{2/3}$ and set j_i to be $j_{i-1} + \ell$. In such a case we say that $q_1, ..., q_\ell$ contribute to j_i. Note that each query can contribute to q_r for at most one r. It follows that for all i we have that $j_i \leq i$.

Consider now what happens whenever an execution of T discovers the end point of the path with less than $\frac{1}{4}m^{1/3}$ steps. There must be the first step k for which the kth query is at distance larger than $(j_{k-1} + 1) \cdot m^{2/3}$ and was answered '1'. Let B be the end point of the prefix at stage $k - 1$ (i.e of distance $j_{k-1} \cdot m^{2/3}$ from the origin), let A be the k-th query and let C_i, $i \leq k - 1$ be all the previous queries that were answered by '0' and were to distance further than $(j_{k-1} + 1) \cdot m^{2/3}$. By Claim 3.1, we have that for any $1 \leq k \leq 1/4m^{1/3}$ the probability that such an event will occur is at most $\frac{2}{\sqrt{m^{2/3}}} = \frac{2}{m^{1/3}}$. Therefore the probability that such an event will occur for some $1 \leq k \leq 1/4m^{1/3}$ is at most $1/2$ and therefore the expected number of queries is at least $\Omega(m^{1/3})$. $\square$.

By replacing the grid with an expander we can get a problem with sharper bounds. Let G be a 3-regular expander with n edges and let u be some node in G. Associate the n inputs with the edges of G. Thus every assignment to the inputs defines a subgraph G' by the edges that are assigned '1'. The problem **ODD** is: find a node other than u with an odd degree in G' or find that u has even degree in G'.

ODD is a valid search problem by the fact that every graph has an even number of nodes with odd degree. Therefore $N(ODD) = 3$. Using the fact that expanders are rapidly mixing, i.e. that a random walk in an expander gets to a node that is almost random after $O(\log n)$ steps, we can show that $R(ODD)$ is $\Omega(n^{\frac{1}{3}})$. The intuition is that as in the grid, we concentrate the probability on walks of length $n^{2/3}$ that start from u. A query to a 'far enough' point is unlikely to be on the walk, thus the tree would have to ask many vertices along the walk in order to find its end. The advantage here is that 'far enough' becomes quite small ($O(\log n)$ due to the mixing property).

We conjecture however that $R(ODD)$ is $\Theta(n)$.

Lemma 3.3 $D(GRID) = O(m) = O(\sqrt{n})$

Proof: There is a deterministic decision tree of complexity $O(m)$ that solves the problem. It asks all entries in the $\lfloor \frac{m}{2} \rfloor$-row. If there is any '1' entry, there must be an answer in the upper half of the matrix. Otherwise, there must be an answer in the lower part of the matrix. The decision tree probes next the relevant half of the $\lfloor \frac{m}{2} \rfloor$-column and recurses respectively. $\Box$

Lemma 3.4 $D(GRID) = \Omega(m)$

Proof: There is a simple adversary strategy that can force any deterministic algorithm to query at least $m-1$ locations. The adversary maintains a contiguous path of 1's from the bottom left location (initially this path contains just the bottom left point). It also maintains a direction for the path which is either horizontal or vertical. Given a query, if it is not in the same row or column as the end point of the path or it is on the same row (column) and the direction is vertical (horizontal), the adversary answers 0. If it is, say, in the same row as the end point of the path and the direction is horizontal, then if in all the columns between the end point and current query point a query has been made, then the adversary answers 1 and gives away 1's for all the locations between the end point and query point. If not, he answers the query by 0, finds the first column in which a query has not been yet made, fills the row with 1's up to that point and switches the direction to vertical. The other case is treated similarly.

It is easy to see that this strategy maintains the invariant that, at any step, the current path can be augmented to the top row or right column. Moreover, the adversary answers 1 in a row (column) only if all columns (rows) between the current end point and the query have points that had already been queried. So, every 1 in (i,j) position is discovered after at least $max(i,j)-1$ steps. $\Box$

3.2.2 A problem with $N(F) = O(\log n)$ and $R(F) = \Omega(n/\log n)$

Our next example is of nondeterministic complexity $O(\log n)$ and randomized complexity of $\Omega(n/\log n)$. The lower bound on the randomized complexity is based on a reduction from a problem in communication complexity.

Let K_{3m} be the complete graph on $3m$ vertices. Let P_m be the set of all m-matchings in K_{3m} i.e, the set of all m pairwise disjoint edges. Let Q_m denote the set of all $(m-1)$-subsets of vertices of K_{3m}. Note that for every member $p \in P_m$ and every member $q \in Q_m$ there is an edge $e \in p$ such that $e \cap q = \emptyset$. Our search problem is essentially to find such an edge on an input $(p, q) \in P_m \times Q_m$. However, we use some Boolean encoding of the problem.

We encode the sets by permutations (as explained below) and permutations by permutation networks. A permutation network is a graph that 'realizes' permutations. Formally, let G be a directed acyclic graph with k sources called input nodes, k sinks, called output nodes and some other nodes called switches. Each input node has one out-going edge and each output node has one in-going edge. Each switch has two in-going and two out-going edges and can be assigned to select one of the two permutations that map the two in-going edges to the two out-going edges. Clearly, a setting to each switch defines k paths in G, each from an input node to an output node. Thus it defines a mapping from the k inputs to the k outputs. It is easy to see that this mapping is always a permutation.

Let S_k be the symmetric group on k elements. A graph G as above is called a 'k-permutation network' if for every $\pi \in S_k$ there is a setting to the switches so that the mapping defined by it is π. For details of construction some permutation networks see [12, 16].

Fact: For every k, a k-permutation network of size $O(k \log k)$ and depth $O(\log k)$ can be constructed efficiently. (e.g the shuffle-exchange network is a simple construction).

We can formally define now our n variables search problem **MATCH**: Let $k = 3m$. Fix two disjoint k-permutation networks and let n be the total number of switches ($n = O(m \log m)$). The input is an assignment to the switches of each network, interpreted as two permutations π_1, π_2 on k elements. The first permutation encodes an m-matching p by; $\pi_1(i)$ is matched to $\pi_1(i+m)$ for every $1 \leq i \leq m$. The second encodes a set q of size $m-1$ by $q = \{\pi_2(1), .., \pi_2(m-1)\}$. The search problem is to find an edge as above.

Theorem 3.3 *$N(MATCH) = O(\log n)$ and $R(MATCH) = \Omega(n/\log n)$*

Proof : $N(F) = O(\log m)$ since all one has to do is to 'guess' i and find j, r such that $\pi_1(i) = j$, $\pi_1(i+m) = r$ (this is an edge in p). In addition, check that $j,\ r \notin q$ by 'guessing' $s,\ t > m-1$ such that $\pi_2(s) = j$, $\pi_2(t) = r$. This takes $O(\log m)$ probes.

The lower bound on the randomized complexity follows from: (i) the result of Raz and Wigderson [18] on the complexity of the problem of finding the desired edge in the communication complexity setting where one party has $p \in P_m$ as its input and the other party has $q \in Q_m$ as its input. They showed that $\Omega(k)$ bits must be transmitted. (ii) The fact that any lower bound in the communication complexity model is also a lower bound in the decision tree model, since the players can simulate the decision tree for each other (transmit the current bit being probed). We don't give a detailed definition of the communication complexity model here. For further information see [11], [18] and [1].

3.3 Simultaneous large gaps; $N(F) << R(F) << D(F)$

In this section we construct a problem with simultaneous exponential gaps between $N(F)$, $R(F)$ and $D(F)$. We remark here that the deterministic lower bound is based on an interesting application of an upper bounds for an online coloring algorithm.

Let r be an integer. Let $G = (V, E)$ be an m vertex graph that is not r colorable and $n = m \log r$. The r-coloring search problem for G, denoted by $\mathbf{COL}_r\mathbf{(G)}$, is the following n variable problem: Every assignment of the $n = m \log r$ variables is interpreted as an r-coloring of G. The goal is to find two neighbors with the same color. Clearly such a configuration always exists; we call such a configuration a 'monochromatic edge'.

Theorem 3.4 *Let $r = (logm)^2$, $d = 16r^2$, and let $G = (V, E)$ be a d-regular m-vertex Ramanujan expander as constructed by Lubotzky, Phillips and Sarnak [13]. Then,*

1. $N(COL_r(G) = O(\log r) = O(\log \log m)$
2. $R(COL_r(G)) = \Omega(\sqrt{r})$ *and* $R(COL_r(G)) = O(r \log r)$
3. $D(COL_r(G)) = \Omega(2^{\frac{1}{6}\sqrt{r}}) = \Omega(m^{1/6})$

Proof: We first have to show that the search problem is indeed valid, that is, to prove that G is not r-colorable. This, as well as the rest of the proof will follow from the lemmata below.

Lemma 3.5 *For every r-coloring of G there exist at least $8 \cdot r \cdot m$ monochromatic edges.*

Proof: For a set $S \subseteq V$ let $E(S) = \{(u, v) | \ u, v \in S\}$. Let S_i, $1 \leq i \leq r$ be the color classes under a coloring of V with r colors. The number of monochromatic edges is thus $\sum_{i=1}^{i=r} |E(S_i)|$. However by the expansion properties of G, for every $S' \subset V$, $|E(S')| \geq \frac{d \cdot |S'|^2}{m} - 2\sqrt{d}|S'|$. Thus, the number of monochromatic edges is at least

$$\sum_{i=1}^{i=r} (\frac{d \cdot |S_i|^2}{m} - 2\sqrt{d}|S_i|) \geq \frac{d}{m}(\sum_{i=1}^{i=r} |S_i|^2) \ - 2\sqrt{d}m \geq$$

$$\frac{d}{mr}(\sum_i |S_i|)^2 - 2\sqrt{d}m = \frac{dm}{r} - 2\sqrt{d}m = 8 \cdot r \cdot m \quad \Box$$

Lemma 3.6 $R(COL_r(G)) = O(r \log r)$ *and* $R(COL_r(G)) = \Omega(\sqrt{r})$.

Proof: The upper bound follows from Lemma 3.5: there are at least $8 \cdot r \cdot m$ 'monochromatic' edges for every r-coloring; thus selecting an edge at random and probing the $2 \log r$ bits that define its end points' colors results in a witness with probability at least $8mr/dm \in \Omega(1/r)$. Therefore we get that the expected number of queries is $O(r \log r)$.

The lower bound follows by showing a "hard" distribution (as in Yao [21]). The distribution is uniform on all r-colorings. It is easy to see (by the 'birthday paradox'), that any deterministic tree must probe at least $\sqrt{r}$ vertices in order to hit the same color twice with constant probability. (In particular to find two neighbors with the same color).

Lemma 3.7 *Let $k = \frac{2}{3}\log_{d-1} m$. For every integer s, every induced subgraph G' of G on at most $t = s^{k-1}$ vertices has a vertex of degree at most s.*

Proof: By [13] G has no cycles of length less than $2k = \frac{4}{3}\log_{d-1} m$. Let G' be the smallest subgraph for which every vertex has degree at least $s+1$. For a vertex $v \in G'$ let $S_0 = \{v\}$ and $S_i = \{u | (u,v) \in G',\text{ and } v \in S_{i-1}\}$. Every $u \in S_i,\ i < k$ has just one neighbor in S_{i-1} and no neighbors in S_i since otherwise G' has a cycle of length smaller than $2k$. However, since the degree of every such u is at least $s+1$ we get that $|S_i| \geq s \cdot |S_{i-1}|$ for all $i < k$ which gives that G' contains at least s^{k-1} vertices. $\Box$.

Lemma 3.8 $D(COL_r(G)) = \Omega(2^{\frac{1}{6}\sqrt{r}}) = \Omega(m^{1/6})$

Proof: Let $s = \sqrt{r}$. By Lemma 3.6 every induced subgraph G' of size at most $t = s^{k-1} = \Omega(2^{\frac{1}{6}\sqrt{r}})$ has a vertex of degree of at most s. It follows that it can be *online* colored by $s \cdot \log t < r$ colors, since Irani [9] has shown that the greedy algorithm has this performance. This means that as long as the decision tree probes no more than t nodes, the adversary can correctly online color the induced subgraph of the probed nodes so that no monochromatic edge occurs. $\Box$

4 The case of $N(F) = 2$

In this section we investigate decision problems for which $N(F) = 2$, i.e those which correspond to unsatisfiable 2-CNF formulae. It turns out that in that case the situation is different from the general case. Namely, $D(F)$ can be nearly characterized. It follows also that for n-variables problems $D(F) = O(\log n)$ for any such F. However, $R(F)$ may still be small in comparison to $D(F)$.

Let f be an unsatisfiable CNF formula. We say that a sub formula f' of f is critical if it is unsatisfiable but deletion of any clause makes it satisfiable.

Theorem 4.1 *Let F be an n-variables search problem represented by a 2-CNF formula $f(F)$. Let f' be a critical sub formula with minimum number of clauses. Let k be the number of variables in f' and m be the number of clauses in f'. Then $k \geq m/2$ and $\log m \leq D(F) \leq 2 + \log m$.*

Proof: Let T be a decision tree for F. Look at the set of clauses in its leaves. Every input reaches one of those clauses and falsifies it. So, the sub formula F' defined by the clauses of the tree is unsatisfiable. i.e, it has at least m clauses. We conclude that the size of T is at least m and its depth is at least $\log m$ which proves the lower bound on $D(F)$.

To prove the upper bound define for any unsatisfiable 2-CNF formula F the (standard) directed graph $G(F)$, associated with F : $V(G)$ is the set of $2n$ literals. For every clause $(\alpha \vee \beta)$, $(\overline{\alpha} \rightarrow \beta)$, and$(\overline{\beta} \rightarrow \alpha)$ are edges in $E(G(F))$. For every single variable clause x, $(\overline{x} \rightarrow x)$ is an edge of $G(F)$.

Claim 4.1 *For any unsatisfiable 2-sat formula F let $G(F)$ be its graph, then there is a variable x such that there is a directed path from x to $\overline{x}$ and a directed path from $\overline{x}$ to x in $G(F)$.*

Proof: The proof is by induction on the number of variables of F. The claim is easily checked for 2 variables formulae. Assume F is unsatisfiable. Chose any variable x in F and for any possible two clauses $(x \vee y)$, $(\overline{x} \vee z)$ produce the 'resolvant' $(y \vee z)$. The formula F_1 obtained by deleting every clause that contains x or $\overline{x}$ and adding the new clauses is unsatisfiable. By induction hypothesis there is some y such that $G(F_1)$ has a path P_1, from y to $\overline{y}$ and a path P_2, from $\overline{y}$ to y. However every edge (u, v) in $G(F_1)$ is either an edge in $G(F)$ or is the result of resolving two clauses of the form $(x \vee \overline{u})$ and $(\overline{x} \vee v)$. But then, the associated edges $(u, x), (x, v)$ are a path from u to v in $G(F)$. Thus every path from a to b in $G(F_1)$ has a corresponding path from a to b in $G(F)$, in particular so do P_1, P_2. $\Box$

Let f' be a critical sub formula of F and let $G(f')$ be its associated directed graph. By the claim there is a variable x for which there is a directed path P_1 from x to $\overline{x}$ and a directed path P_2 from $\overline{x}$ to x. This leads to the following decision tree for F. First x is probed. If $x = 1$ the decision tree will find an edge in P_1 which is directed from '1' to '0' by binary search along P_1. If $x = 0$ it will do the same thing on P_2. Such an edge (u, v) corresponds to a clause $(\overline{u} \vee v)$ which is falsified (since $u = 1, v = 0$). Every clause contributes two edges to the graph so, the length of each of the paths is no more than $2m$. We get the bound of $2 + \log m$ on the number of probes in the binary search.

We may take P_1, P_2 above to be simple paths, in particular the length of the paths is bounded by k too. So, one gets an upper bound of $1 + \log k$ as well, and so $k \geq m/2$ by the lower bound. $\Box$

Corollary 4.1 *For every 2-CNF search problem F on n variables, $D(F) = O(\log n)$* $\Box$.

We show now that the randomized complexity can be much smaller than the deterministic, and in other cases the largest possible.

Let G_1 be a constant degree Ramanujan expander on n vertices of the type constructed by [13]. $COL_2(G)$ is the 2-coloring search problem as defined section 3.3.

Theorem 4.2 $N(COL_2(G_1)) = 2$, $R(COL_2(G_1)) = O(1)$, $D(COL_2(G_1)) = \Omega(\log \log n)$.

Proof: The fact that $N(COL_2(G_1)) = 2$ is clear from the definition of the problem. The fact that G_1 is not two colorable and $R(COL_2(G_1)) = O(1)$ follows from the same arguments as in the proof of 3.5, 3.6 with $r = 2$. The proof that $D(COL_2(G_1)) = \Omega(\log\log n)$ follows from Theorem 4.1 above since a critical sub formula for $COL_2(G_1)$ corresponds to the edges of a non-two colorable subgraph of G_1 (in particular it must contain an odd cycle). However the cycles of G_1 are of length $\Omega(\log n)$ [13].

Let G_2 be an odd cycle of length n.

Theorem 4.3 $N(COL_2(G_2)) = 2, \quad R(COL_2(G_2)) = \Omega(\log n)$.

Proof $N(COL_2(G_2)) = 2$ is obvious. Following Yao's technique, the distribution will be uniformly concentrated on the n different inputs coloring, the i-th being the one that colors correctly all edges except the i-th edge.

A deterministic tree of average depth d must have at least $1/2$ of the inputs reaching leaves of depth no more than $2d$ (from our special set of n inputs). Since there are no more than 2^{2d} leaves of depth $2d$, at least $\frac{n}{2^{2d+1}}$ inputs arrive at the same leaf. However, no two inputs from our special set can arrive at same leaf since every such input has a different witness. So, $n \leq 2^{2d+1}$ or $d = \Omega(\log n)$ $\square$

References

[1] L. Babai, P. Frankl, and J. Simon, *Complexity classes in communication complexity theory*, Proc. 27th Annual IEEE Symp. on Foundation Of Computer Science, (1986), pp. 337–347

[2] M. Blum and R. Impagliazzo, *Generic oracles and oracles classes*, Proc. 28th Annual IEEE Symp. on Foundations Of Computer Science, (1987), pp. 118–126.

[3] A. Blake, *Canonical expressions in Boolean algebra*, Ph.D. thesis, Department of Mathematics, Chicago Univesity, Chicago, 1937.

[4] V. Chvatal and E. Szemerédi, *Many hard examples for resolution*, J. ACM, 35 (1988), 759–768.

[5] E. Szemerédi, *Personal communication* (1991).

[6] M. Davis and H. Putnam, *A computing procedure for quantification theory*, J. ACM 7 (1960), 210–215.

[7] J. Hartmanis and L. A. Hemachandra, *One-way functions, robustness and the non-isomorphism of* NP*-complete sets*, Proc. of 2nd Structure in Complexity Theory Conference, (1987), pp. 160–173.

[8] M.D. Hirsch, C. H. Papadimitriou and S. Vavasis, *Exponential lower bound for finding Brouwer fixed points*, J. of Complexity, 5 (1989), pp. 379–416,

[9] S. IRANI, *On-line coloring inductive graphs*, Proc. 31 IEEE Annual Symp. of Found. of Comp. Sci. (1990), pp. 470–479.

[10] R. IMPAGLIAZZO AND M. NAOR, *Decision trees downward closure*, Proc. 3rd Structure in Complexity Theory Conference, (1988), pp. 29–38.

[11] M. KARCHMER AND A. WIGDERSON, *Monotone circuits for connectivity require super-logarithmic depth*, Proc. of 20th Annual ACM Symposium on Theory of Computing, (1988), pp. 539–550.

[12] F. T. LEIGHTON, *Introduction To Parallel Algorithms And Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, 1992, Ch-3.2, 3.3, pp. 439–510.

[13] A. LUBOTZKY, R. PHILLIPS, AND P. SARNAK, *Explicit expanders and the Ramanujan conjecture*, Proc. of 18th Annual ACM Symposium on Theory of Computing, (1986), pp. 240–246.

[14] N. NISAN, CREW PRAMS *and decision trees*, Proc. 21st ACM Symp. on Theory of Computing, (1989), pp. 327–335.

[15] C. H. PAPADIMITRIOU, *graph theoretic lemmata and complexity classes*, Proc. 31st IEEE Symp. on Foundations of Computer Science, (1990), pp.D 794–801.

[16] N. PIPPENGER, *Communication Networks*, in Handbook Of Theoretical Computer Science, MIT Press, Cambridge, MA, 1990, CH-15, pp. 807–833.

[17] J. A. ROBINSON, *A machine-oriented logic based on the resolution principle*, J. ACM 12 (1965), pp. 23–41.

[18] R. RAZ AND A. WIGDERSON, *Monotone circuits for matching require linear depth*, Proc. 22th ACM Symp on theory of computing, (1990), pp. 287–292.

[19] G. TARDOS, *Query complexity, or why is it difficult to separate* $NP^A \cap \mathrm{CO}NP^A$ *from* P^A *by a random oracle?*, Combinatorica 9 (1989), pp. 385–392.

[20] L. G. VALLIANT, *General purpose parallel architectures*, in Handbook Of Theoretical Computer Science, MIT Press, Cambridge, MA, 1990, CH-18 pp. 945–971.

[21] A. C. YAO, *Probabilistic computation, towards a unified measure of complexity*, Proc. 18th IEEE Symp. on Foundation of Computer Science, (1977), pp. 222–227.

5 Appendix

5.1 The connection to the Resolution problem

Here we present an observation of V. Chvatal and E. Szemerédi [5] that relates the complexity of the resolution process for an unsatisfiable formula F and its deterministic decision tree, via a generalization of decision trees (branching programs).

A **resolution** for an unsatisfiable CNF formula F is a process that proves that the formula is unsatisfiable. It generates additional clauses which should be satisfiable if F is unsatisfiable until a contradiction is obtained (the empty clause). The resolution process was first defined by Blake [3] and became popular as a theorem proving technique by Robinson [17] and Davis and Putnam [6]. Formally, let F be an unsatisfiable CNF formula with clauses $\mathcal{C} = \{C_1, .., C_m\}$. The resolution is a straight line program. In each step l, a clause C^l is produced if either $C^l \in \mathcal{C}$ or there exist $i, j < l$, and a variable x, $s.t$ $C^i = (x \vee \alpha)$, $C^j = (\overline{x} \vee \beta)$ and $C^l = (\alpha \vee \beta)$ where α and β are disjunctions of literals. In that case we say that C^l was obtained by resolving on x. The resolution is indeed a proof for F if it ends with the empty clause. The size of the resolution is the smallest number of steps to reach the empty clause, denoted here by $RES(F)$. The resolution process for F may be described as a directed acyclic graph G of in degree 0 or 2. The vertices are the clauses that are generated by the resolution process. If C^i, C^j are resolved to obtain C^l then the two corresponding nodes are connected by directed edges into C^l. The 'output' node is the empty clause.

A resolution is said to be **regular** if in every directed path form input to the output node, every variable is resolved at most once.

Theorem 5.1 *[6] For every unsatisfiable CNF formula F there is a finite regular resolution that proves F.*

We denote by $RRES(F)$ the minimum size of a regular resolution for F.

A **branching program** for an unsatisfiable CNF formula F is a directed acyclic graph G with outdegree 0 or 2 and a special source vertex s. Each vertex of out degree 2 is labeled by a variable. The two out-going edges are labeled by the two possible values the variable can take. Every 0-1 assignment of the variables defines a path from the source to a sink in a natural way. Each sink is labeled by a clause of F such that for every assignment of the variables, the path from the source ends in a sink labeled with a clause that is unsatisfied by the assignment.

The size of branching program for an unsatisfiable CNF formula F, $BP(F)$, is the smallest size (number of vertices) of a branching program for F.

A branching program is said to be **Read-Once**, (ROBP), if in any source-sink path every variable appears at most once. Ths size of the smallest ROBP for F is denote by $ROBP(F)$.

Note: A decision tree for F is also a ROBP for F.

Theorem 5.2 *[5] Let F be an unsatisfiable CNF formula, then $RRES(F) = ROBP(F)$.*

Corollary 5.1 $D(F) \geq \log(RRES(F))$.

We will present here the ideas of the proof of Theorem 5.2 for the sake of completeness.
Proof: Assume first that T is a ROBP for F. We will associate a clause to every vertex of T such that T becomes a resolution graph for F. A vertex v labeled by a variable x will be associated with a clause $C(v)$ with the property that every input that reaches v in T falsifies $C(v)$. We associate clauses inductively from the sinks backwards. To each sink we associate the clause it is labeled with by T. Let v be the next vertex to be associated with a clause. v is labeled with a variable x in T and has edges (v, u_0) for $x = 0$ and (v, u_1) for $x = 1$. We assume, by induction, that $u_0, (u_1)$ is labeled with $C_0, (C_1)$ respectively.
Claim; C_0 does not contain $\overline{x}$ and C_1 does not contain x $\square$.

We conclude that either $C_0 = (x \vee \alpha),\ C_1 = (\overline{x} \vee \beta)$ or one of C_0, C_1 does not contain $x, \overline{x}$ at all. In the first case label v with $C(v) = (\alpha \vee \beta)$. In the second case label v with the clause that does not contain $x, \overline{x}$.

By the definition of the labeling, it is clear that the source node is labeled by the empty clause (since every input reaches it). Thus, the tree represents a regular resolution process for F (possibly with some redundant steps that correspond to the second case of the labeling above).

Assume now that we have a resolution graph G for F. G can be transformed to be a branching program for F by reversing the direction of the edges, and labeling each node by the variable used to resolve the clause it is associated with. (except the sinks which are labeled by their clauses).

It can be seen that it gives a branching program for F by asserting the following property. For every node v originally associated with a clause $C(v)$, all inputs that arrive to v (from the root), falsify $C(v)$. Furthermore, if G represented a regular resolution, then it results in a ROBP since along a path each variable appears at most once. $\square$.

Note that the above also proves $BF(F) \leq RES(F)$. We remark that in general there can be an exponential gap between these two measures. (Since any unsatisfiable CNF has a BP of the size of F while RES(F) might be exponential). It is an interesting question to find a concrete model of computation that is polynomially equivalent to resolution.