Unambiguous Computation

Suhas Vijaykumar

Spring 2016

Overview

This survey paper is designed to review the literature on unambiguous computation. Broadly speaking, studying unambiguous computation is an attempt to determine how much of the power of non-determinism comes from the ability to accept many different proofs of the same statement. The literature on this subject is of both practical and philosophical interest, and the Isolating lemma of Mulmuley, Vazirani and Vazirani [5] appears beautifully throughout.

In the first first, we will define the notion of unambiguous computation and the associated complexity classes we will consider. In the second section, we'll discuss unambiguous polynomial-time computation. Most of the results in this section are from the '80s—in particular we'll discuss relationships between the complexity classes **P**, **UP**, and **NP**, and prove the Isolating lemma as well as the Valiant-Vazirani theorem.

In the third and final section, we'll discuss some of the more recent results about unambiguous log-space computation. In particular, we'll prove $\mathbf{UL}/\text{poly} = \mathbf{NL}/\text{poly}$ and discuss the latest attempts to prove the conjecture that $\mathbf{NL} = \mathbf{UL}$.

Basic Definitions

The notion of "unambiguous" computation, defined below, arose from attempts to better understand the essential differences between deterministic and non-deterministic computation. Recall that **NP** is the set of languages L such that

$$x \in \mathsf{L} \iff \exists\, v \;\; P(v, x) \tag{1}$$

for some predicate $P(v, x)$ that can be evaluated by a deterministic Turing machine in time polynomial in $|x|$.

The complexity class **UP** is defined by imposing the restriction that the advice string v in (1) is *unique*. This gives rise to the following definition.

Definition 1. The class **UP** consists of those languages L such that

$$x \in \mathsf{L} \iff \exists!\, v \;\; P(v, x) \tag{2}$$

where $P(v, x)$ can be evaluated by a deterministic Turing machine in time polynomial in $|x|$.

It is evident from this definition that $\mathbf{UP} \subseteq \mathbf{NP}$. It is also very easy to show that $\mathbf{P} \subseteq \mathbf{UP}$.

Lemma 2. $\mathbf{P} \subseteq \mathbf{UP}$.

Proof. Given $\mathsf{L} \in \mathbf{P}$, let M be a polynomial time Turing machine that decides L. Then a polynomial time machine M′ can check both that v is the empty string and that $x \in \mathsf{L}$. It is evident that the predicate

$$P(x, y) = (x \in \mathsf{L}) \wedge (v \text{ is the empty string})$$

satisfies (2). □

Thus **UP** is a natural class of languages which sits between **P** and **NP**. To generalize this construction to other complexity classes, we define an "unambiguous Turing machine."

Definition 3. An *unambiguous Turing machine that decides* L is a non-deterministic Turing machine M that decides L, such that for each $x \in \mathsf{L}$, $\mathsf{M}(x)$ has exactly one accepting path.

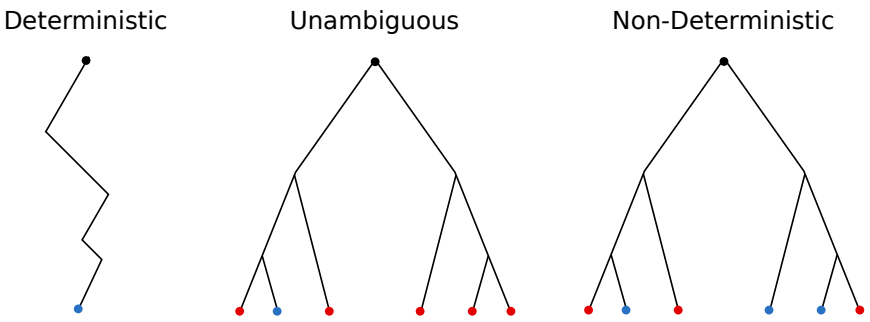

Figure 1: Accepting computation trees for a deterministic (left), unambiguous (middle), and non-deterministic (right) Turing machine. Accepting configurations are in blue, while rejecting configurations are in red. Note that the unambiguous Turing machine's computation tree has many branches, but only one acepting path.

It is evident from Definition 3 that unambiguous computation is at most as powerful as non-deterministic computation, and at least as powerful as determinsitic computation. In particular, we have the following definitions.

Definition 4. Let **USPACE**$[f(n)]$ (resp. **UTIME**$[f(n)]$) denote the set of languages decidable by unambiguous Turing machines in space (resp. time) $O(f(n))$. We define

$$\mathbf{UP} = \bigcup_{k \in \mathbb{N}} \mathbf{UTIME}[n^k], \tag{3}$$

$$\mathbf{UL} = \mathbf{USPACE}[\log n]. \tag{4}$$

It is evident from the inclusions **SPACE**$[f(n)] \subseteq$ **USPACE**$[f(n)] \subseteq$ **NSPACE**$[f(n)]$ that

$$\mathbf{L} \subseteq \mathbf{UL} \subseteq \mathbf{NL}.$$

Likewise the inclusions **TIME**$[f(n)] \subseteq$ **UTIME**$[f(n)] \subseteq$ **NTIME**$[f(n)]$ imply

$$\mathbf{P} \subseteq \mathbf{UP} \subseteq \mathbf{NP}.$$

Similarly, we obtain the non-uniform counterparts of these classes. For completeness, we prove the following lemma.

Lemma 5. *The two definitions of* **UP** *we've given,* (2) *and* (3)*, agree.*

Proof. Suppose $\mathsf{L} \in \bigcup_k$ **UTIME**$[n^k]$ as in (3). Let the advice string v be the computation history for the unique accepting path of an unambiguous Turing machine that decides L in polynomial time. The polynomial time predicate stipulates that each transition of the history is valid, which can be checked deterministically in polynomial time.

Conversely, if L satisfies (2) then an unambiguous Turing machine can guess a string v' and then evaluate $P(v', x)$ deterministically. The unique accepting path corresponds to the guess $v' = v$. □

Unambiguous Polynomial-Time Computation

In this section, we'll spend some time discussing the complexity class **UP**. In particular, we will introduce the Valiant-Vazirani Theorem, a major result in complexity theory related to the power of **UP**. We will also introduce the Isolating lemma of Mulmuley, Vazirani and Vazirani. It turns out that the Valiant-Vazirani Theorem is a straightforward consequence of the Isolating lemma, and many famous open problems in theoretical computer science are related to derandomizing special cases of the Isolating lemma. We will go on to see that determining whether the containment **UL** $\subseteq$ **NL** is strict is one such problem.

We will also discuss an application of the Valiant-Vazirani theorem to cryptography. In particular, it has implications for the existence of one-way functions, which are an important cryptographic primitive.

The Isolating Lemma (cf. [5])

As depicted in Figure 1, making a non-deterministic Turing machine unambiguous amounts to choosing a single accepting branch of the computation tree, when there might be exponentially many of them.

A major obstacle to doing this is that no single branch of the computation tree has any information about the state of any other branch. Thus, while it might initially seem easy to choose a single accepting branch, say, lexicographically, we do not know how to do this.

This very same problem arises often in the context of distributed computing. In that context, it is often the case that many separate parties must make a common decision—for example, elect a leader—but have limited capability to communicate with each other.

The problem of selecting one preferred choice from a set of many equally good choices is often called "symmetry breaking." The Isolating lemma uses randomness to solve the symmetry breaking problem in a very general setting—both for accepting branches of an **NP** decider and for many applications in parralel and distributed computing.

Proposition 6. *Let I be a set of size n and let 2^I denote the set of subsets of I. For each $k \in I$, let a_k be chosen uniformly at random from the set $\{1, 2, \ldots, m\}$. Let $\omega : 2^I \to \mathbb{N}$ be defined by*

$$\omega(S) = \sum_{k \in S} a_k.$$

Then for any collection $\mathcal{C} \subseteq 2^I$, with probability $1 - {}^{n}/_{m}$ there is a unique $S \in \mathcal{C}$ such that

$$\omega(S) = \min_{S' \in \mathcal{C}} \omega(S').$$

Proof. (cf. [10]). For $k \in I$, put $C_k = \{S \in \mathcal{C} \mid k \in S\}$ and $\bar{C}_k = \{S \in \mathcal{C} \mid k \notin S\}$. Write

$$f(k) = \min_{S' \in \bar{C}_k} \omega(S') - \min_{S \in C_k} \omega(S \setminus \{k\}).$$

Since $f(k)$ does not depend on a_k, and since a_k is chosen independently and uniformly at random from $\{1, 2, \ldots, m\}$, the probability that $a_k = f(k)$ is ${}^{1}/_{m}$. By the union bound, the probability that $a_k = f(k)$ for *any* k is at most ${}^{n}/_{m}$.

Suppose that two sets, S and T, both minimize ω in $\mathcal{C}$. Choose some $k \in S \setminus T$, and note that we must have

$$\omega(T) = \min_{S' \in \bar{C}_k} \omega(S'),$$
$$\omega(S \setminus \{k\}) = \min_{S' \in C_k} \omega(S' \setminus \{k\}).$$

Thus we have

$$f(k) = \omega(T) - (\omega(S) - a_k) = a_k.$$

We conclude that this occurs with probability at most n/m. □

It is tempting to ask whether the Isolating lemma can be derandomized. The answer, in general, is no. To make the statement precise, let k be given, and write $[n] = \{1, 2, \ldots, n\}$. In most applications of the Isolating lemma, the parameter m is bounded by some polynomial in n. A "derandomization" of the Isolating lemma should therefore consist of a family of functions $f_n : [n] \to [n^k]$ such that for any $S, T \in 2^{[n]}$ we have $\omega_n(S) \neq \omega_n(T)$, where

$$\omega_n(S) = \sum_{s \in S} f_n(S).$$

Lemma 7. *Let $\{f_n\}$ be a family of functions $f_n : [n] \to [n^k]$. For sufficiently large n there exist $S, T \in 2^{[n]}$ such that $\omega_n(S) = \omega_n(T)$.*

Proof. For large enough n we have $\#2^{[n]} = 2^n > n^{k+1} = \#[n^{k+1}]$. Since the range of ω_n is contained in $[n^{k+1}]$ and the domain is $2^{[n]}$, the pigeonhole principle tells us ω_n is not injective. □

The Valiant-Vazirani Theorem

It turns out that the Valiant-Vazirani theorem is a fairly easy consequence of the Isolating lemma. The theorem says that if there existed an **RP** Turing machine M which could correctly decide instances of CLIQUE that had *zero or one* clique of the desired size, and whose behavior was otherwise arbitrary, then **NP** = **RP**. Thus, under the widely-held assumption that **NP** $\neq$ **RP**, finding a k-clique in a graph that is known to contain at most one k-clique is hard.

As we'll see, the role of randomness in the proof is in using the Isolating lemma to form a predicate that is satisfied by *only one* clique of the desired size. Thus derandomizing the Isolating lemma for the special case where $\mathcal{C}$ is the set of cliques in a graph would yield a proof that **UP** = **NP**.

Before stating the theorem, we a Promise-**UP** oracle. This helps us get around the issue that **UP** is not known (or expected) to contain languages such as USAT or UCLIQUE—the set of boolean formulas (resp. graphs) with exactly one satisfying assignment (resp. k-clique). Even though these languages have unique proofs, a **UP** decider has no obvious way of checking, say, that a given boolean formula φ does not have more than one satisfying assignment.

Definition 8. An Promise-**UP** oracle is defined as follows. Let L be a language in **NP** and D a decider for L. Given D and some string x, the oracle outputs 1 if D has a single acceping path on input x, 0 if no accepting paths, and $P(x)$ if D has two or more accepting paths, and its output is chosen arbitrarily from $\{0, 1\}$ otherwise.

The Valiant-Vazirani theorem states that **NP** $\subseteq$ **RP**$^{\text{Promise-UP}}$. In fact it's very easy to see that USAT and UCLIQUE are hard for co-**NP**, hence the analogous statement that **NP** $\subseteq$ **RP**$^{\text{USAT}}$, which is also a consequence of the theorem, is probably weaker.

Theorem 9 (Valiant-Vazirani, cf. [11])**.**

Proof. The heart of the proof is a randomized procedure that, given $\langle G, k\rangle$, constructs $\langle G', k'\rangle$ with the following properties.

(1) If there is no clique of size k in G, there is no clique of size k' in G'.

(2) If there is a clique of size k in G, then with probability $1/4n^2$ there is *exactly one* clique of size k' in G'.

If we perform this procedure $4n^2$ times, and pass $\langle G', k' \rangle$ to M each time, then if G contains no k-clique, M will always reject. If G contains a k-clique, however, then G' will contain exactly one k' clique with probability $1 - (1 - 1/4n^2)^{4n^2} \geq 1 - 1/e \geq 1/2$, and M will accept. Thus we have constructed an **RP** decider for CLIQUE.

To construct $\langle G', k' \rangle$ we assign each vertex v in G a number $a(v) \in [2n]$, uniformly at random. Then G' is constructed by replacing each vertex v in G with a clique of size $2nk + a(v)$. Additionally, for each edge (u, v) in G and each pair of vertices u' and v' in the cliques that correspond to u and v, G' contains an edge (u', v').

This construction has two important properties. The first is that a k-clique in G corresponds to a clique in G' of size

$$\begin{aligned} 2nk^2 + k \leq k' &\leq 2nk^2 + 2nk \\ &< 2n(k+1)^2 + (k+1). \end{aligned}$$

Thus if there is no k-clique in G, there will be no k'-clique in G' for k' in this range.

The second is that if $\mathcal{C}$ is the set of k-cliques in G, then by the Isolating lemma the smallest corresponding clique in G' has unique size, α, with probability $1/2$. Let r be selected uniformly at random from the range $k, \ldots, 2nk$ and fix $k' = 2nk^2 + r$. Then, with probability greater than $1/2n^2$ we will have $k' = \alpha$, hence G' will have exactly one k'-clique with probability at least $1/4n^2$. □

Corollary 10. $\mathbf{P}^{\text{Promise-}\mathbf{UP}}/\text{poly} = \mathbf{NP}/\text{poly}$.

Proof. Suppose $\mathsf{L} \in \mathbf{NP}/\text{poly}$ and let $n \in \mathbb{N}$ be given. Then, there exists an advice string s_n such that for all x of length n,

$$x \in \mathsf{L} \iff \exists\, v \;\; P(s_n, v, x)$$

where P is a polynomial-time predicate in x.

Since CLIQUE is **NP**-complete, there exists a polynomial time computable mapping reduction that produces a string $\langle G, k \rangle(s_n, x)$ such that

$$\exists\, v \;\; P(s_n, v, x) \iff \langle G, k \rangle(s_n, x) \in \text{CLIQUE}.$$

Given such a pair $\langle G, k \rangle$, the randomized reduction from the proof of the Valiant-Vazirani theorem produces, with probability $1/4n^2$, a pair $\langle G', k' \rangle$ such that G' has a unique clique of size k'. If we run the reduction $4n^3$ times, the probability none of the resulting graphs G' has a unique clique of size k' is at most $(1 - 1/4n^2)^{4n^3} \leq (1/e)^n < 1/2^n$. By the union bound, the probability that this occurs for *any* $x \in \{0, 1\}^n$ is strictly less than 1.

By the probabilistic method, there eixists a string r_n such that simulating the reduction $4n^3$ times using r_n instead of randomness produces a sequence $\langle G', k' \rangle_1, \langle G', k' \rangle_2, \ldots, \langle G', k' \rangle_{4n^3}$ such that at for least one $\langle G', k' \rangle_\ell$, G' has a unique clique of size k' *with certainty*. Thus, a $\mathbf{P}^{\text{Promise-}\mathbf{UP}}$ decider, given r_n, s_n and x, can deterministically produce strings $\langle G', k' \rangle_1, \langle G', k' \rangle_2, \ldots, \langle G', k' \rangle_{4n^3}$. If it calls the Promise-**UP** oracle for each string, it can determine that $x \in \mathsf{L}$ if and only if the oracle accepts at least one string. □

Relativizations

Although not discussed extensively here here, there are many interesting results about what happens to the hierarchy $\mathbf{P} \subseteq \mathbf{UP} \subseteq \mathbf{NP}$ in various relativized worlds. Most prominently, Charles Rackoff in [7] constructed languages A and B such that

$$\mathbf{P}^{\mathsf{A}} = \mathbf{UP}^{\mathsf{A}} \neq \mathbf{NP}^{\mathsf{A}}$$
$$\mathbf{P}^{\mathsf{B}} \neq \mathbf{UP}^{\mathsf{B}} = \mathbf{NP}^{\mathsf{B}}.$$

Thus any proof that separates or collapses the hierarchy will not not reativize. In general, collapsing the hierarchy **L** $\subseteq$ **UL** $\subseteq$ **NL** is thought to be much more likely to be within the reach of current techniques. So, without further ado, we will consider that problem.

Unambiguous Log-Space Computation

In this section, we will consider unambiguous log-space computation. In particular, we will use the fact that the computation graph of an **NL** decider is a polynomially-sized directed acyclic graph to relate the hardness of **UP** to certain graph problems. This has been the subject of much recent study.

In particular, we will show that the problem of finding deciding (s, t) connectivity in a certain class of graphs, namely *min-unique* graphs, is contained in **UP**. Using this fact, we will go on to prove that **UL**/poly = **NL**/poly, and that the conjeture **UL** = **NL** can be proved by derandomizing the Isolating lemma for paths in a directed acyclic graph.

Inductive Counting and Min-Unique Graphs

In this section, we'll prove that the problem of deciding (s, t) connectivity on min-unique graphs is in **UL**. This result will form the foundation for what is to follow.

The theorem, proved by Reinhardt and Allender, builds upon the inductive counting algorithms of Savich's proof that **NSPACE**$[f(n)] \subseteq$ **DSPACE**$[f(n)^2]$, as well as Immerman and Szelepcsényi's proof that **NL** = co-**NL**; both are extremely important results in complexity theory [4, 9]. These algorithms all exploit the fact that a space bounded Turing machine can simulate many non-deterministic branches sequentially and store their results on its tape.

A key aspect of this theorem is that the **UL** decider rejects not only those graphs that have no (s, t) path but also those that have *more than one* (s, t) paths. This differs from the results we discussed for **UP** as the Turing machines we describe do not assume anything about their input.

Definition 11. A *min-unique* graph is a weighted, directed graph G such that there is a unique shortest path between each pair of vertices.

Theorem 12 (cf. [8])**.**

$$\left\{ \langle s, t, G \rangle \,\middle|\, G \text{ is a min-unique graph with an } (s, t) \text{ path} \right\} \in \textbf{UL}.$$

Proof. First, we introduce some notation that we use throughout the proof. For a vertex v of G, let $d(v)$ denote the length of the shortest path from s to v, or else $|G| + 1$ (if there is no such path). Moreover let D_k be the set of vertices v such that $d(v) \leq k$, and write $S_k = \sum_{v \in D_k} d(v)$. First, we introduce a useful lemma, which we will prove at the end.

Lemma 13. If provided the correct values $\#D_k$ and S_k, and if G_k, the subgraph induced by D_k, is min-unique, then a non-deterministic log-space Turing machine can decide if $d(v) \leq k$ unambiguously—in other words, all branches reject except for one branch which has the value of the predicate $d(v) \leq k$ written on its tape.

The important detail about the lemma is that computation can continue as though no branching occured. In light of the lemma, we have the following algorithm. Given $\#D_{k-1}$ and S_{k-1}, the algorithm iterates through all vertices in G and checks whether $d(v) \leq k - 1$. It does so using the procedure from Lemma 13.

The Turing machine writes $d = \#D_{k-1}$ and $s = S_{k-1}$ on it's work tape. For each vertex such that $d(v) \not\leq k-1$, it iterates through vertices such that (v, x) and $d(x) \leq k - 1$, again using Lemma 13. These vertices must have $d(v) = k$. If it

finds just one such vertex, it increments d by one, increments s by k, and proceeds as normal. If it finds a two or more such vertices, it rejects—there are two shortest paths to v, hence graph is not min-unique.

Having successfully completed the iteration, there is a single branch of the computation that hasn't rejected, and this branch has processed all vertices such that $d(v) = k$. Thus $d = \#D_k$ and $s = S_k$, and it has been verified that G_k is min-unique. Thus, the process can be repeated for $k + 1$ up to n, at which point the entire connected component of s has been processed. If the computation hasn't rejected at this point, the graph is min-unique and we an use Lemma 13 to check if $d(t) \leq n$, which is true iff there's an (s, t) path in G. □

Proof of Lemma 13. Suppose that S_k, D_k are as defined, and G_k is min-unique. Given v, the Turing machine keeps two running totals, s and d. For each vertex u, the Turing machine guesses if $d(u) \leq k$ and guesses a path of length $\ell \leq k$ from s to u checks if it is valid. If not, it rejects. Otherwise, it increments s by ℓ and c by 1. If $u = v$, it writes this down on the tape. At the end of the iteration, it checks that $d = D_k$ and $s = S_k$ and rejects otherwise.

First, note that all branches that guess $\ell < d(u)$ reject. If the algorithm incorrectly guesses $d(u) \leq k$ it will necessarily guess $\ell < d(u)$ and reject. Thus we must have $d \leq D_k$, and if any branch incorrectly guesses $d(u) > k$ for any u it will terminate with $d < D_k$.

Thus any branches that do not reject must guess $d = D_k$ and $\ell \geq d(u)$ for each $u \in D_k$. It follows that an accepting branch has $s \geq S_k$, and since G_k is min-unique exactly one branch will have $s = S_k$. This branch checks if it ever guessed $u = v$. This happend if and only if $d(v) \leq k$, and so the machine writes this down on its tape and the procedure is complete. □

The Isolating Lemma: Reprise

In this section, we'll apply the Isolating lemma to show that **UL**/poly = **NL**/poly. We'll also see that derandomizing the Isolating lemma for paths in a directed acyclic graph would let us prove the conjecture that **UL** = **NL**.

Theorem 14 (cf. [8])**.** **UL**/poly = **NL**/poly.

Proof. Let $\mathbf{G}_n$ be the directed acyclic graph with vertex set $[n]$ and an edge (i, j) for each $i < j$, and note that each directed acyclic graph G with n vertices is a subgraph of $\mathbf{G}_n$. If we assign each edge (i, j) a weight uniformly at random in $[3n^3(n+1)^3]$ then the probably that the set of (s, t) paths does not have a unique lowest-weight path, for any choice of (s, t), is at most $\frac{1}{3}n^{-1}(n+1)^{-1}$. This follows from the Isolating lemma, since the set of paths contains at most $n(n + 1)$ edges. Thus the probability that the graph isn't min-unique is at most $1/3$ by the union bound. This holds for any $G \subseteq \mathbf{G}_n$.

Now suppose we repeat this procedure $n(n + 1)$ times to produce weighted graphs $G_1, G_2, \ldots, G_{n(n+1)}$. The probability that none of these graphs is min-unique is $\left(\frac{1}{3}\right)^{\binom{n}{2}}$. By the union bound, the probability that this event occurs for *any* of the $2^{\binom{n}{2}}$ subgraphs of $\mathbf{G}_n$ is at most $\left(\frac{2}{3}\right)^{\binom{n}{2}}$. Thus, by the probabilistic method, there exists a sequence of weight functions $f_1 \ldots f_{n(n+1)}$ such that for all directed acyclic graphs on n vertices at least one of the resulting weighted graphs has a unique shortest path. If we replace each edge of weight ℓ with a path of length ℓ, we obtain $n(n + 1)$ graphs such that at least one must be min-unique.

If we give all of the resulting graphs to the **UL** decider we constructed in Theorem 12, we can successfully decide PATH.

Now we can complete the proof. Take some $\mathsf{L} \in$ **NL**/poly. Then if if $|x| = n$ there exists an a_n such that

$$x \in \mathsf{L} \iff \exists y \quad P(a_n, x, y)$$

for some log-space predicate P. We construct a PATH instance $\langle s, t, G\rangle(a_n, x)$ and then use the string $f_1 \dots f_{n(n+1)}$ to decide if $\langle s, t, G\rangle(a_n, x) \in$ PATH. Thus given the advice string $a_n || f_1 \dots f_{n(n+1)}$, a **UL** Turing machine can decide L, and hence **NL**/poly = **UL**/poly. □

Note that the importance of non-uniform computation in the previous proof was that it allowed us to turn any directed acyclic graph into a min-unique directed acyclic graph such that (s, t) connectivity is preserved. This tempts one to ask if it is possible to introduce a uniformly computable weighting scheme such that an arbitrary directed acyclic graph can be turned made min-unique. The motivation for this question is made explicit below.

Proposition 15. *Suppose that, for some fixed k, there exists a family of log-space computible functions $f_n : [n] \times [n] \to [n^k]$ with the property that for every weighted directed acyclic graph G on n vertices and $s, t \in G$ with edge weights given by f_n, there was a unique lowest-weight (s, t) path. Then* **NL** = **UL**.

Proof. Compute the weights and replace each edge of weight k with an unweighted path of length k. Then, run the **UL** decider from Theorem 12. □

Results in this direction have been meager, although it was recently shown that deciding reachability on planar graphs and 2D rectangular grid graphs is in **UL** [2]. Interestingly, reachability on 3-page graphs and 3D cubic grid graphs is known to be complete for **NL** [6], suggesting an interesting geometric boundary between the two classes.

Acknowledgements

I want to thank Prof. Moshkovitz and the T.A.s Govind and Mohammad for a great semester.

References

[1] Allender, Eric, et al. "Grid graph reachability problems." *Computational Complexity* (2006)

[2] Bourke, Chris, Raghunath Tewari, and N. V. Vinodchandran. "Directed planar reachability is in unambiguous log-space." *ACM Transactions on Computation Theory* (TOCT) 1.1 (2009): 4.

[3] Buntrock, Gerhard, Lane A. Hemachandra, and Dirk Siefkes. "Using inductive counting to simulate nondeterministic computation." *Information and Computation* 102.1 (1993): 102-117.

[4] Immerman, Neil. "Nondeterministic Space Is Closed under Complementation." *SIAM Journal on Computing* 17.5 (1988): 935-38.

[5] Mulmuley, Ketan, Umesh V. Vazirani, and Vijay V. Vazirani. "Matching Is as Easy as Matrix Inversion." *Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing: STOC* '87 (1987).

[6] Pavan, Aduri, Raghunath Tewari, and N. V. Vinodchandran. "On the power of unambiguity in log-space." *Computational Complexity* (2012): 1-28.

[7] Rackoff, Charles. 1982. "Relativized Questions Involving Probabilistic Algorithms." *J. ACM* 29, 1 (January 1982), 261-268.

[8] Reinhardt, Klaus, and Eric Allender. "Making nondeterminism unambiguous." *SIAM Journal on Computing* 29.4 (2000): 1118-1131.

[9] Savitch, Walter J. "Relationships between nondeterministic and deterministic tape complexities." *Journal of computer and system sciences* 4.2 (1970): 177-192.

[10] Spencer, Joel. "Ten lectures on the probabilistic method," *SIAM Regional Conference Series in Applied Mathematics,* Vol. 52.

[11] Valiant, Leslie G., and Vijay V. Vazirani. "NP is as easy as detecting unique solutions." *Proceedings of the seventeenth annual ACM symposium on Theory of computing.* ACM, 1985.

18.405J / 6.841J Advanced Complexity Theory

Spring 2016