The Complexity of Graph Connectivity

Avi Wigderson
Hebrew University and Princeton University

February 11, 2003

Abstract

In this paper we survey the major developments in understanding the complexity of the graph connectivity problem in several computational models, and highlight some challenging open problems.

1 Introduction

If you have ever lost your way, (especially when you had a map and road signs to guide you), you must recognize that graph connectivity is not a completely trivial problem. While in practice the relevant problem is how to get from point A to point B, here (in theory) we are content with finding out if there is such a way at all.

Of all computational problems, this is the one that has been studied on the largest variety of computational models, such as Turing machines, PRAMs, Boolean circuits, decision trees and communication complexity. It has proven a fertile test case for comparing basic resources, such as time vs. space, nondeterminism vs. randomness vs. determinism, and sequential vs. parallel computation.

There seem to be two complimentary reasons for this wide interest in the complexity of graph connectivity. On the one hand it is simple enough to be almost completely understood from the combinatorial point of view. On the other hand, it has rich enough structure to capture (in different variants) several important complexity classes, whose exact computational power is not yet completely understood.

Until about five years ago progress on the complexity of connectivity was sporadic. Highlights are the basic results of the early 70's, such as Savitch's theorem and the DFS and BFS algorithms, and the discovery in the early 80's of the power or randomness in the undirected case. More recently, there has been an upsurge in activity, yielding fundamental (and sometimes surprising) results, such as $NL = coNL$, the short universal traversal sequences, and the lower bounds on the constant-depth and monotone circuits.

The aim of this paper is to summarize this progress, and to point to interesting directions where more progress needs (and is perhaps likely) to be made. The paper reflects my biased knowledge and viewpoint, and does not claim to be encyclopedic. In particular, I will focus on what I consider intrinsic complexity issues, rather than algorithmic efficiency issues. My

hope is to attract researchers to the open problems, and am looking forward to more exciting results.

The paper has four sections. In section 2 we give definitions and basic results. Sections 3 and 4 contain respectively results in the Turing machine and Boolean circuit models. Section 5 discusses three unrelated (structured) settings — projections, decision trees, and logical expressibility. I have tried to give both the motivations for some directions of study, as well as intuition for some of the results. The suggestions for open problems appear in the text in their natural place. Clearly, to seriously pursue them one needs to read the original papers, most of which are fortunately very well written.

2 Definitions and Completeness Results

2.1 Variants of Graph Connectivity

We define below the four variants of the graph connectivity problem that will be studied in the paper. They are defined as languages, i.e. as subsets of (binary) strings, which is suitable when studying Turing Complexity. As usual we shall also think of each language as an infinite family of Boolean functions, one for every input length, which take the value '1' on inputs in the language and '0' otherwise. This is natural for nonuniform models like circuits and decision trees.

We shall deal with both directed and undirected graphs $G(V, E)$ with set of vertices V and edges E. We will assume that graphs are represented by their adjacency matrix (as this representation captures the monotonicity of connectivity), though most results hold in any other standard representation. Finally s and t will denote fixed distinct vertices in V. The problems are ordered in decreasing difficulty (under almost any choice of reductions).
STCONN: Directed st-connectivity
$\{< G, s, t >$ such that there is a directed path from s to t in the directed graph $G\}$
USTCONN: Undirected st-connectivity
$\{< G, s, t >$ such that there is a path connecting s and t in the undirected graph $G\}$
UCONN: Undirected connectivity
$\{< G >$ such that the undirected graph G is connected (i.e. every pair of vertices in G is connected)$\}$
CYCLE: Is a permutation cyclic?
$\{< G >$ such that the permutation G (every vertex has exactly one incoming edge and one outgoing edge) has exactly one cycle$\}$

2.2 Complexity Classes

All complexity classes defined below are standard. For the most updated source see [**?**]. This survey also discusses uniformity issues in circuit classes, that we shall ignore here. For definitions of monotone complexity classes, see [**?**]. We shall think of complexity classes both as classes of languages and classes of functions (for purposes of reductions). For a language L we let $\bar{L}$ denote its complement, and for a class $\mathcal{C}$ we let $co-\mathcal{C} = \{ \bar{L} \mid L \in \mathcal{C}\}$. The class $m\mathcal{C}$ will denote the monotone analog of $\mathcal{C}$. For circuit classes this is easily done by disallowing

negations. For Turing machine classes it requires more work, but can be done in natural ways [?]. All the classes we are interested in are within $\mathcal{P}$, polynomial time. We list (most of) them in obvious inclusion order (the same inclusions hold for the monotone analogs of these classes). For circuit classes the basis is always $\wedge, \vee, \neg$ unless otherwise specified, and that they have bounded fan-in unless the depth is constant. Finally, I will neglect constant factors and even the use of big-O and big-Ω.

$\boldsymbol{AC^0}$: Constant-depth, polynomial size circuits.
$\boldsymbol{TC^0}$: Constant-depth, polynomial-size circuits with threshold gates.
$\boldsymbol{NC^1}$: $\log n$-depth, polynomial-size circuits (or equivalently, polynomial-size formulae)
$\boldsymbol{L}$: Deterministic log space Turing machines.
$\boldsymbol{ZPL}$: Probabilistic zero-error (Las Vegas) log-space, polynomial-time Turing machines.
$\boldsymbol{RL}$: Probabilistic one-sided error (Monte-Carlo) log-space, polynomial-time Turing machines.
$\boldsymbol{NL}$: Non-deterministic log-space Turing machines.
$\boldsymbol{NC^2}$: $\log^2 n$-depth, polynomial size circuits.

A class whose position in this list is not apriori obvious (other than being between L and NL is symmetric log-space [?], defined by non-deterministic log-space Turing machines whose next move relation is symmetric (i.e. if it is legal to move from configuration c to configuration d, it is legal as well to move from d to c).
$\boldsymbol{SL}$: Symmetric log-space Turing machines.

We shall use $\boldsymbol{DSPACE(s(n))}$ and $\boldsymbol{TISP(t(n), s(n))}$ to denote what can be computed by deterministic Turing machines respectively in space $s(n)$ and simultaneously in time $t(n)$ and space $s(n)$. The class SC which contains NL is simply $TISP(poly(n), polylog(n))$.
$\boldsymbol{SC}$: Polylog-space, polynomial-time Turing machines.

2.3 Completeness Results

As mentioned in the introduction, the importance of the above variants of graph connectivity is that they capture complexity classes, and thus understanding the power of these classes boils down to proving upper and lower bounds for these problems.

Theorem 1 ([?]) *STCONN* is NC^1 complete for NL.

Theorem 2 ([?]) *USTCONN* is NC^1 complete for SL.

Theorem 3 ([?]) *CYCLE* is NC^1 complete for L.

We add that the first two problems (which are monotone) also belong to the monotone analogs of the classes they are complete for.

3 Turing Machine Complexity

3.1 Directed Graphs

Two basic results on the complexity of *STCONN* exist for over 20 years. The first follows from the discovery of the very efficient algorithms for graph traversal, BFS (Breadth-First

Search) and DFS (Depth-First Search), which use only linear time and space in n, the number of vertices.

Theorem 4 $STCONN \in TISP(n, n)$

The second is Savitch's theorem, giving an upper bound on the deterministic space complexity of *STCONN*.

Theorem 5 ([?]) $STCONN \in DSPACE(\log^2 n)$

The algorithm uses the recursive doubling techniques, which can be thought of as a depth first search of the circuit which computes *STCONN* by repeated squaring of the adjacency matrix of the input graph. By the completeness of *STCONN*, Savich's theorem resolves the "space analog" of the P vs. NP question in time complexity, i.e that nondeterminism is not superpolynomially stronger than determinism.

Corollary 1 ([?]) $NL \subseteq DSPACE(\log^2 n)$

Whether the quadratic upper bound is tight is the most important question in this area. As current techniques have failed to provide super-logarithmic space lower bounds, such bounds exist only for restricted models. One of the natural models for this problem is the JAG (Jumping Automata on Graphs) of Cook and Rackoff [**?**]. It allows an automaton to place pebbles on the vertex s and move them along edges or jump them to each other's location with the task of reaching t with at least one of them. Space in this model is the logarithm of the number of states in the automaton, plus $\log n$ per each pebble used. They show that this model can simulate Savitch's algorithm in $\log^2 n$ space (using $\log n$ pebbles), and then prove

Theorem 6 ([?]) Every JAG algorithm for *STCONN* requires $\log^2 n / \log\log n$ space.

The same lower bound was proved even for randomized JAGs by Berman and Simon [**?**].

These structured lower bounds do not preclude, of course, a better space bound on real Turing machines. A somewhat easier question arises since the algorithm which gives this upper bound runs in time $n^{\log n}$. It, together with the DFS and BFS algorithms, suggests that there may be a nontrivial trade-off between time and space for this problem, and in particular asks what is the smallest space for which *STCONN* can be computed in polynomial time. The first progress on this problem, an algorithm using sublinear space, came only a couple of months ago.

Theorem 7 ([?]) $STCONN \in TISP(poly(n), n/2^{\sqrt{\log n}})$.

I believe that the space bound can be significantly improved, to $n^{1-\epsilon}$ and even n^{ϵ}, and predict that such improvements will be found soon. Some people believe the opposite, and in fact Tompa [**?**] proves that certain natural approaches cannot yield such space savings in polynomial time.

Another natural question regarding nondeterministic space complexity, as long as the quadratic gap in Savitch's theorem remains, is whether NL is closed under complement. This again took about 20 years to resolve in the affirmative. I prefer to state this result of Immerman and Szelepcsenyi in terms of a reduction from *STCONN* to $\overline{STCONN}$.

Theorem 8 ([?, ?]) *STCONN* is L-reducible to $\overline{STCONN}$.

In words, there is a deterministic log-space procedure (in fact NC^1) that takes the input graph G and outputs a graph H such that there is a directed st-path in G if and only if there is no such path in H. It follows that

Corollary 2 ([?, ?]) $NL = co - NL$

It is interesting to note that the inductive counting method used to prove this theorem seems inherently non-monotone. This was made precise in [?] who showed that the monotone circuit lower bound of [?] (which we shall discuss in the next section) implies that $\overline{STCONN} \notin NL$, and thus

Theorem 9 ([?]) $mNL \neq mco - NL$

3.2 Undirected Graphs

In contrast to the directed case, its friendlier undirected analog received much more attention, and many nontrivial techniques and interesting results were obtained for it.

3.2.1 Probabilistic Algorithms

One of the earliest and most beautiful examples of the power of randomness is the probabilistic log-space algorithm for undirected connectivity of Aleliunas et. al. [?].

Theorem 10 ([?]) $USTCONN \in RL$

The RL algorithm for undirected graphs can be viewed as a randomized version of the NL algorithm for directed graphs. In the directed case a pebble (which requires only $\log n$ space) is placed initially on the start vertex s, and the algorithm moves the pebble along directed edges, each time guessing the next vertex, until it reaches t. Clearly, if an st-path exists there will be a successful sequence of guesses and vice versa. In the undirected case, the same pebble is moving between neighbouring nodes, only that the next move is determined by coin flips. The fact that it is impossible "to get stuck" in an undirected graph allows a fairly simple proof that such a walk will visit all vertices in the connected component of s in expected polynomial time, and hence will determine if there is a path to t.

It is clear that if the pebble does not reach t, it may be (albeit with very small probability) due to unlucky coin tosses, and thus there is one-sided error. The question of eliminating this error altogether was raised in [?] and was answered in the affirmative only about 10 years later, by Borodin et. al. [?]. They employ the same randomized algorithm, together with the inductive counting technique which is used to verify that indeed all vertices in the connected component of s were visited.

Theorem 11 ([?]) $USTCONN \in ZPL$

From the completeness of *USTCONN* for SL we obtain

Corollary 3 ([?]) $SL \subseteq ZPL$

As ZPL is closed under complement (in fact $ZPL = RL \cap co-RL$) this may be considered an indication that SL may be closed under complement. This question is still open. We remark that besides being a natural question, resolving it in the affirmative will collapse Reif's symmetric log-space hierarchy, which contains some interesting problems such as planarity of bounded-degree graphs [**?**].

3.2.2 Universal Traversal Sequences

The notion that induced the largest and deepest set of results in this area is the universal traversal sequences, proposed by Cook in the late 70's. Cook tried to create a deterministic analog of the pebble-moving algorithm used in the nondeterministic and probabilistic algorithm. The idea was that the next move of the pebble will be determined by a sequence of instructions that will be computed deterministically. For this purpose we assume that the edges around every vertex are distinctly labeled by some set of labels, (e.g. 1,2,...,d if the graph is d-regular, or 1,2,...,$|V|$ otherwise), and the given sequence is over the set of labels.

Definition 1 A sequence is n-universal is for every connected graph on n vertices, every labeling of the edges, and every start vertex on which the pebble is placed, the trajectory of the pebble defined by the sequence will visit all the vertices in the graph.

The computational problem associated with this notion is the explicit construction of such sequences.

Definition 2 *UNIVSEQ* is the relation $\{< 1^n, \sigma > \mid n \geq 1, \sigma \text{ is } n\text{-universal}\}$

The hope was and still is that $UNIVSEQ \in L$, which will clearly imply $USTCONN \in L$. However, the simple "greedy" construction of such sequences requires poly (n) space and the resulting sequences have $\exp(n)$ length. (In general, if $UNIVSEQ \in DSPACE(s(n))$ then the sequences produced by the given algorithm must have length at most $\exp(s(n))$). The first indication that the hope may not be too optimistic was given in the aforementioned paper by Aleliunas et. al. (and indeed was one of their motivations).

Theorem 12 ([?]) There exist n-universal sequences of poly(n) length.

The proof however is by a probabilistic argument (essentially the same argument showing that the probabilistic algorithm works) and does no give a space bound for constructing them which is better than poly(n). We briefly remark that there are quite a few papers dealing with upper and lower bounds on the shortest possible length of a universal sequence as a function of the number of vertices and edges in the graph, that we do not mention here, which are very interesting from the combinatorial point of view but less so from the computational one.

The first breakthrough on the complexity of generating universal sequences appears in the remarkable paper of Babai, Nisan and Szegedy [**?**]. This paper obtains a variety of state-of-the-art results for different problems on different computational models, which are all derived from a single lower bound on the multi-party protocols model of [**?**]. In particular they obtain

Theorem 13 ([?]) *UNIVSEQ* $\in DSPACE(exp(\sqrt{\log n}))$.

The proof utilizes pseudo-random generators for logspace, which are more important than this particular corollary as we shall see in the next subsection. In another giant step by Nisan [?] the complexity of these generators was significantly reduced, and with it the upper bound on the complexity of *UNIVSEQ*.

Theorem 14 ([?]) *UNIVSEQ* $\in DSPACE(\log^2 n)$

This is the best known explicit construction of universal traversal sequences for general graphs, and it results in a one-pebble deterministic algorithm for *USTCONN* with the same space complexity of Savitch's algorithm. Better space bounds for universal sequences for special classes of graphs exist. Of these, the most interesting is Istrail's result [?], which constructs in L universal sequences for all labeled cycles. Here for each n there is only one graph, the n-cycle, which is 2-regular, and the label set is 1,2. To stress our ignorance, Borodin points out that if we add a self loop to every vertex on the cycle, and make the label set 1,2,3, nothing better than the general construction is known.

3.2.3 Pseudo-random Number Generators

This subsection is somewhat of a detour, as it does not deal with the connectivity problem directly. However, this direction was, and I expect it will continue to be, a major source of results on the complexity of connectivity. For this subsection only we let $G(V, E)$ be a directed graph again, and s a fixed vertex in V. For a probability distribution D on binary strings $\{0,1\}^t$ let $G(D)$ denote the distribution on V defined by the endpoint of the following process: place a pebble initially on s, and use a random string from D to determine a sequence of moves on G where at each step we use the next few bits to move the pebble along a random edge out of the vertex it occupies (if there is no such edge, it stays put). Let U_t be the uniform distribution on strings of length t. Thus the algorithm of [?] is an instance of such a process, which in general captures probabilistic computation with $\log |V|$ space that uses t (truly) random bits.

The idea of pseudorandom generators, which originated in cryptography [?] and found many uses in complexity theory (see [?] and the references within), is to deterministically compute distributions on t bits from a much shorter random string, that will "behave like" the uniform distribution U_t. We shall consider here only functions $f : \{0,1\}^{r(n)} \to \{0,1\}^{t(n)}$ with $f \in DSPACE(\log n)$, and $\log n \leq r(n) << t(n) \leq poly(n)$. The idea is that if $G(f(U_r))$ is "close" to $G(U_t)$ then the random process on G can be simulated deterministically in $DSPACE(r(n))$, by trying all possible "seeds" of length r to f (which is called a pseudo-random generator).

The first result of this type is due to Ajtai, Komlos and Szemeredi [?]. If we define a restricted version $RL(t(n))$ of random log-space which allows the machine to use only $t(n)$ random bits on length n inputs, their result can be stated as

Theorem 15 ([?]) $RL(\log^2 n / \log\log n) = L$

There is no other computational model for which such a strong result, namely that superlogarithmic number of random bits do not add computational power, holds. The authors devise an ingenious pseudo-random generator f, based on walks on expander graphs, which takes $r(n) = O(\log n)$ bits and outputs $t(n) = \log^2 n / \log\log n$ which has the following "closeness" property for every graph G on n vertices and every vertex $v \in V$. If $Pr[G(U_t)) = v] \geq 1/n$ then $Pr[G(f(U_r)) = v] > 0$. This clearly suffices to simulate the bounded one-way error in RL.

It is beyond the scope of this papers to list the variety of applications this construction had in complexity theory, and we just mention as examples the deterministic amplification of [?, ?] and the ϵ-biased random variables of [?].

Another nice feature of this result is that it suggests partial progress towards proving $RL = L$, by finding faster growing functions $t(n)$ for which $RL(t(n)) = L$ holds. I am quite optimistic that this can be shown for $t(n) = polylog(n)$ in the near future.

The pseudo-random generators developed in [?, ?] are stronger in two respects than those in [?]. First, they output $t(n) = poly(n)$ bits, and hence do not restrict the number of random bits used. Second, they achieve a stronger "closeness" property, namely $|G(f(U_r)) - G(U_t)| \leq 1/n$, (where $|\cdot|$ denotes the L_1 norm), and thus can handle two-sided errors as well. The generator in [?], which is the most efficient generator known, uses $r(n) = \log^2 n$ bits. The construction is based on the properties of pairwise independent (2-universal) hash functions of Carter and Wegman [?]. Both the construction and the proof that it works resembles the recursive doubling of Savitch's algorithm, and indeed can be thought as a strong probabilistic analog of it. Combining this result with the one of [?] to obtain the universal sequence bounds of the previous section is an exercise.

3.2.4 Time-Space Trade-offs

We are back again to undirected connectivity. As even the best universal sequences leave the space bound for this problem at $\log^2 n$, activity shifted to trying to achieve polynomial time with the smallest space penalty possible. The first step was by Barnes and Ruzzo [?], who proved:

Theorem 16 ([?]) For every $\epsilon = \epsilon(n) > 0$,
$USTCONN \in TISP(n^{1/\epsilon}, n^\epsilon \log^2 n)$

Note that this family of algorithms provides a complete trade-off curve with one extreme at DFS/BFS, and the other at Savitch's algorithm. In particular their algorithm uses polynomial time whenever space is n^ϵ with any fixed $\epsilon > 0$. This result survived only a few months, till Nisan [?] showed that if we are willing to pay $\log^2 n$ space, time can be reduced to a polynomial, and there is no trade-off!

Theorem 17 ([?]) $USTCONN \in TISP(poly(n), \log^2 n)$

Again Nisan proves a much more general theorem of which the above is a corollary. He uses the following simple idea, which should be kept in mind whenever we use a pseudo-random generator. Generators are defined such that they work for *every* input, and thus

are used in an oblivious manner. However, all we need is a generator that works for the given input! Nisan shows how, using space $\log^2 n$, a "subgenerator" of the general one in [?] can be constructed, which is guaranteed to "fool" only the input graph. Furthermore, this generator uses only $r(n) = \log n$ random bits as a "seed". Thus he proves

Theorem 18 ([?]) $RL \in TISP(poly(n), \log^2 n)$

which he prefers to state concisely as

Corollary 4 ([?]) $RL \in SC$

3.2.5 Beating the Recursive Doubling Technique

In light of the last upper bound on *USTCONN*, the only thing to improve now is the space bound of $\log^2 n$. In all of the results above that achieve this space bound recursive doubling is implicit in the following sense: A function (such as squaring) which is in L is repeatedly applied $\log n$ times, resulting in the above bound. The reason for the $\log n$ application is that the natural parameter (distance between pairs of vertices in the case of squaring) shrinks only by a constant factor with each application. If we could shrink something at a faster rate, we would have fewer iterations and thus smaller space. (We note that a result of Ajtai described in the next section suggests that it is unlikely that we can shrink the distance parameter faster in L).

Such an algorithm was discovered very recently by Nisan, Szemeredi and Wigderson [?]. The L function takes the input graph $G(V, E)$ and produces another graph $G'(V', E')$, in which s and t are connected if and only if they are connected in G. The parameter which shrinks is simply the number of vertices, which drops by a factor of $exp(\sqrt{\log n})$. Thus only $\sqrt{\log n}$ iterations are required to bring the size down to a constant, where it is trivial to check if s and t are connected, and hence the total space bound is $\log^{1.5} n$.

Theorem 19 ([?]) $USTCONN \in DSPACE(\log^{1.5} n)$

We can describe the above shrinking procedure in a few sentences. The idea is to choose V' a subset of V, have each vertex in V find a reachable representative in V', and to homeomorphically contract each vertex to its representative. This will clearly preserve connectivity. The choice of such small set V' is done in two steps. First, each vertex constructs a "ball" of reachable vertices around it of size $exp(\sqrt{\log n})$. This can be done using universal sequences by scaling down the results in [?]. Second, V' is chosen to be a small hitting set of all the balls. This is implemented using pairwise independent sampling. The representative of v can thus be taken to be the smallest vertex which is both in V' and the ball of v.

We have made some attempts to improve this space bound. In particular a natural approach is to try to apply this algorithm recursively, to obtain a space bound of $\log^{1+\epsilon} n$. There are obvious problems of vertex renaming that suggest themselves when trying to implement this approach, but I am convinced that such a result will not be too difficult to obtain.

Another interesting issue that pops back to life is the time-space trade-offs. The above algorithm requires time $n^{\sqrt{\log n}}$. I think that the techniques of [?] should be powerful enough

to obtain a polynomial time algorithm while maintaining the $\log^{1.5} n$ space bound. However, similar problems of vertex renaming that were mentioned in the previous paragraph will have to be dealt with (unless an altogether different approach is found).

4 Circuit Complexity

It is easy to see that $STCONN \in NC^2$, by considering the circuit which repeatedly squares the adjacency matrix. One central problem (of the "algorithmic efficiency" type) in the area of parallel algorithms comes from the fact that this circuit at best has size larger than $n^{2.376}$, which is the best current bound for matrix multiplication [**?**]. The problem, whether there is a circuit that uses simultaneously $polylog(n)$ depth and $n^2 polylog(n)$ size (which is nearly linear, as n^2 is the input size), is known as the "transitive closure bottleneck". It resisted many attacks in the last 10 years, and resolving it will be of interest both in theory and practice.

We return to complexity problems, and from now on there is no distinction between the directed and undirected case; while we state the results for *STCONN*, they hold for *USTCONN* as well. As it is not known if this problem is in L, and also is L is strictly contained in NC^2, one important direction is to try to reduce the depth below $\log^2 n$, as was done for space. The other direction was to prove it cannot be computed by circuit classes that are contained in L. This section surveys the progress in this direction, which represents considerable effort in the circuit complexity community, employing and developing more difficult arguments than those used in the Turing machine bounds of the previous section. Before elaborating on these beautiful developments, I stress that the current state of knowledge is quite shameful: we don't even know that *STCONN* (or any function in NP for that matter) cannot be computed by threshold circuits of depth 3 and polynomial size!

4.1 Constant-Depth Circuits

Furst, Saxe and Sipser, in the paper which proved the parity lower bounds [**?**], observed that a simple reduction from the parity function to *STCONN* yields

Theorem 20 ([?]) $STCONN \notin AC^0$

This is not too surprising, as perhaps we are asking for too much. If we could just shrink in AC^0 the pairwise distances between vertices by a function $k(n)$ which tends to infinity with n, we could repeat this a sublogarithmic number of times to beat the NC^2 upper bound. So define a family of problems $STCONN(k(n))$, which asks whether there is a path from s to t of length at most $k(n)$, and ask how small a function $k(n)$ has to be so that this problem is in AC^0. It is trivial that this is possible for any constant function $k(n)$. As for $k(n) = n$ it is not, where is the jump?

For another class of functions, the threshold functions, the jump is completely determined. Define $THRESH(k(n))$ to be the function that determines if in a string of n bits there are at least $k(n)$ 1's. Then we have

Theorem 21 $THRESH(k(n)) \in AC^0$ iff $k(n) \leq polylog(n)$.

The upper bound follows from the probabilistic construction of [?], while the tight lower bound from Hastad's paper [?]. In fact, the aforementioned reduction from parity to *STCONN*, together with Hastad's bound give the same lower bound for *STCONN*.

Theorem 22 $STCONN(k(n)) \notin AC^0$ for every $k(n) \not\leq polylog(n)$.

To resolve where the jump occurs for *STCONN* much finer analysis of the random restriction techniques had to be developed. (In fact these techniques were developed towards a more general goal, namely showing the weakness of certain logical proof systems [?, ?]). The following result of Ajtai [?] shows that in fact the trivial constant upper bound is tight, and thus *STCONN* is much more difficult then the symmetric functions in this sense.

Theorem 23 ([?]) If $k(n) \to \infty$ with n, then $STCONN \notin AC^0$

This shows that shrinking distances by more than a constant is impossible in AC^0. However, to beat the NC^2 bound it would suffice to shrink in NC^1. Not anticipating a lower bound in the near future, the reasonable challenge here seems to be to devise an NC^1 algorithm for $STCONN(k(n))$ for a function $k(n)$ which tends to infinity. This of course will be a surprising result (but we have been accustomed in the last few years to surprising upper bounds...). If you take on this challenge, make sure your circuit is non-monotone, as the next section implies in particular that monotone constructions will fail.

4.2 Monotone Circuits

It is clear that the repeated squaring circuit for *STCONN* is in fact monotone, and thus $STCONN \in mNC^2$. The first monotone circuit depth lower bound (which does not follow from a size lower bound) was showing that this upper bound is tight. If we denote by $d_m(f)$ the minimum depth of a monotone circuit computing the function f (as a function of the input size n), then the Karchmer-Wigderson lower bound as stated in

Theorem 24 ([?]) $d_m(STCONN) = \Omega(\log^2 n)$

Note that this lower bound is independent of the circuit size, which is allowed to be superpolynomial. The proof also gives a lower bound on the problems $STCONN(k(n))$, showing that in the monotone world the obvious is optimal.

Theorem 25 ([?]) For every function $k(n)$,
$d_m(STCONN(k(n))) = \Omega(\log n \log k(n)))$.

For proving this lower bound [?] introduced the communication complexity method. This method provide a communication search problem in Yao's 2-player model [?] for every Boolean function f, such that the communication complexity of the problem exactly equals the (monotone) circuit depth of f. For example, the following is the problem that captures the monotone depth of *STCONN*. Player A receives as input an st-path, i.e an arbitrary sequence of nodes starting with s and ending with t. Player B gets an st-cut, i.e. an arbitrary 2-partition of the nodes with s in one part and t in the other. Clearly, there is at

least one edge in the given path whose endpoints belong to different sides of the given cut. The task of the players is to find any such edge. It is easy to see that they can do it in about $\log^2 n$ communication bits using binary search as follows. Player A sends the name of the middle vertex on his path ($\log n$ bits). Player B sends the side of the cut this vertex lies on (1 bit). Now player A can discard one half of his path according to the answer, and so $\log n$ such rounds will suffice. Observe that this protocol is simply a top-down view of the repeated-squaring circuit for $STCONN$.

The proof of [?] shows that there is no protocol for this problem which is asymptotically better. In some sense it finds an instance of binary search of this form in every protocol, and argues that if it does not proceed for enough rounds, the players could not find an answer. While I prefer the original proof that argues on the communication model, mainly due to the intuition it supplied us with, I recommend reading also the proof given in [?], which argues directly on the circuit model. On the other hand, such an elegant [1] transformation of a proof in the communication model to the circuit model may not always be possible – such is for example the case with the lower bound for the monotone depth of matching [?].

One interesting extension of the [?] lower bound appears in [?] – it is shown that even a randomized protocol for the same communication problem will require the same number of random bits, up to a constant factor. While randomized protocols results have no direct circuit complexity analogs, this result can be translated into a strange nonmonotone lower bound – though it is possible that $STCONN \in NC^1$, it is impossible that such a shallow circuit will need the negation of too few on its input variables (It is clear, as the function is monotone, that all variable have to appear positively (unnegated)).

Theorem 26 ([?]) In every NC^1 circuit for $STCONN$, at least a fixed fraction of all input bits have to appear negated.

Grigni and Sipser strengthened the results of [?] in two ways. One, which was mentioned earlier, is that in [?] they observed that the proof of [?] is strong enough to imply $STCONN \notin mco - NL$. As $STCONN \in mNL$, it separates the two classes. Another question was to prove such a lower bound for a function in mL. Even a proper definition of this class is not obvious, let alone the choice of a function for which similar techniques can apply. This was done in [?]. The function $FORK$ can be thought of as a monotone analog of $CYCLE$ [2] . It asks whether in the input graph, a path out of a fixed vertex s ever branches (has more than one continuation). It is easy to see that

Theorem 27 ([?]) $FORK \in mL$

The communication problem associated with this function is particularly elegant. Fix two distinct nodes t_A and t_B. Player A gets a path from s to t_A, and player B a path from s to t_B. Their task is to find a vertex on both paths, from which the paths diverge. Using this formulation they proved

[1] A mechanical transformation is always possible, as follows from the fact that the two models are equivalent

[2] There is a simple (nonmonotone) reduction from $CYCLE$ to $FORK$, and thus it is also complete for L.

Theorem 28 ([?]) $d_m(FORK) = \Omega(\log^2 n)$.

Finally we remark on the unresolved status of the problem *UCONN*. In the absence of a direct reduction [3], say from *USTCONN* to *UCONN*, the above lower bounds do not follow for it, and call for a direct proof. A crucial technical point in all proofs mentioned above is that we control the length of the *st*-path, and choose it to be some n^ϵ, which leave most vertices "unimportant" from the point of view of player A. But in *UCONN* the input to player A is a spanning tree, touching all vertices. Raz and I have announced a $\log^2 n$ monotone lower bound for this problem at the circuit complexity workshop in Durham, 1990, but have discovered a bug in the proof. While monotone lower bounds went out of vogue, I am sure this lower bound is correct, and would be happy to see a proof. There is a nontrivial monotone lower bound for this function, due to Yao [**?**].

Theorem 29 ([?]) $UCONN \notin mTC^0$

5 Other Models

5.1 Monotone Projections

In [**?**], Skyum and Valiant proposed to study the relative complexity of (families of) Boolean functions under the most stringent reduction possible – projection. Informally, a function f is a projection of a function g if we can replace some variables of g with a constants (0 or 1) or with other variables or their negation to produce f. An efficient projection (p-projection) demands that the number of inputs to g is only polynomially larger than that of f. Finally, a projection is monotone if we cannot substitute negated variables. One of the interesting results of the paper imply that under monotone p-projections, *UCONN* is strictly weaker than *USTCONN*. This result has two parts. The first shows a hardness result for *USTCONN*.

Theorem 30 ([?]) Every function in mNC^1 has a monotone p-projection to *USTCONN*.

The second part gives a function in mNC^1 such that every monotone projection from it to *UCONN* requires exponential blow-up in the number of variables, and so by the above theorem such a lower bound holds for any reduction from *USTCONN* to *STCONN*. In particular

Theorem 31 ([?]) There is no monotone p-projection from *USTCONN* to *UCONN*.

5.2 Decision Trees

The simplest computational model is the Boolean decision tree. It counts the minimum number of input bits an adaptive algorithm has to look at in order to determine the function value (all other computation is free). A function that requires all bits to be queried is called

[3] The next section shows that at least two natural reductions are in fact impossible

evasive. It is not difficult (do it yourself, or see e.g. [**?**]) to develop an adversary strategy that will force any adaptive algorithm computing *USTCONN* or *UCONN* to look at all the $n(n-1)/2$ "edge slots" before finding the answer. In fact much stronger and more difficult result show (basically) the same for *every* monotone nontrivial property of graphs [**?**].

But let us return to connectivity. The above seem to show both *USTCONN* and *UCONN* to have the same complexity. To supply more evidence that *UCONN* is weaker, [**?**] suggested to consider reductions between these two problems in the decision tree model. Here we are given an "oracle" for one problem when computing the other. Assume the (unknown) input graph is $G(V, E)$. Then a *UCONN* oracle takes a query which is a subset U of V, and answers whether the induced subgraph on U is connected. An *USTCONN* oracle takes as query a subset U and two vertices $a, b \in U$, and replies whether a and b are connected in the subgraph induced on U. Note that in both cases, if $|U| = 2$, both oracles simply say if the edge between the two nodes exist, and thus these are the standard Boolean queries.

Again the weakness of *UCONN* is demonstrated by two results. The first shows that an *USTCONN* oracle significantly reduce the decision tree complexity of *UCONN*. The second shows that a *UCONN* oracle is practically useless (in comparison to simple Boolean queries).

Theorem 32 ([**?**]) There is a decision tree for *UCONN* with an oracle for *USTCONN* that uses only $n-1$ queries.

The proof is trivial: the $n-1$ queries test that vertex 1 (say) is connected to all other $n-1$ vertices (in all of them $U = V$). The difficult direction is to show

Theorem 33 ([**?**]) Every decision tree for *USTCONN* with an oracle for *UCONN* uses $\Omega(n^2)$ queries.

5.3 Expressibility in Logic

This subsection is different than the rest of the paper in that the model that "recognizes" languages is a logical sentence in some theory. The interest in this model came after Fagin [**?**] discovered that that languages definable by existential second order sentences (the class Σ_1^1), capture in a precise sense the complexity class NP. Many similar logical characterizations of complexity classes followed, most of which are surveyed in [**?**]. Again graph connectivity shows up in several places, and the logical point of view shed a different light on the combinatorics of the problem. In particular, the relative difficulty of some variants of connectivity is reversed! Again I will be informal, and refer the reader to the excellent text of Enderton [**?**] on mathematical logic.

How does a logical sentence recognizes a language? Let us restrict ourselves to languages which are subsets of graphs. The universe of variables for the formula is the vertex set V, and the edge set E of the input graph $G(V, E)$ is thought of as a relation for which membership can be tested. It is best to give an example.

The sentence $\forall x \exists y[(x, y) \in E]$ "accepts" the language of all graphs in which there are no isolated vertices. This sentence is clearly first order (quantifies only over elements in the universe), and thus having no isolated vertex (as well as its complement) is a first-order property.

How about our simplest connectivity problem, *CYCLE*? If you do not know the answer, it is worth while to reflect about this before proceeding, and get a feeling (or proof?!) on whether *CYCLE* is a first-order property or not. If you did not succeed in finding a sentence for it, it may have occurred to you that showing that such a sentence does not exist requires some technology. The most useful way for proving such "lower bounds" is the Ehrenfeucht-Fraisse games, developed independently in [**?**] and [**?**]. We shall not describe it here, only note that it is a two- person game, that bears some resemblance (albeit artificial) to the communication complexity game of [**?**] which was used to prove lower bounds for circuit depth. One of the earliest uses of this technique was proving this lower bound on *CYCLE*. This result can be thought of as a precursor to the AC^0 lower bounds of the previous section, as AC^0, in a very precise sense, is a nonuniform version of first order properties.

Theorem 34 *CYCLE* is not a first order property

A much more expressive set is the monadic existential second order (monadic Σ_1^1) sentences. The motivation for their definition, as well as most of the material below, can be found in [**?**]. These are sentences of the form $\exists A_1 \exists A_2 \cdots \exists A_k \psi$, where ψ is a first-order sentence, and the A_i are subsets of the universe. It is an easy exercise to see that such a sentence describes the set of 3-chromatic graphs. How about connectivity? The first two results, which were used by Fagin [**?**] to separate this class of sentences from its complement (where existential quantifiers are replaced by universal quantifiers) was to show

Theorem 35 ([?]) $\overline{UCONN}$ is a monadic Σ_1^1 property, but *UCONN* is not.

The first part of the theorem is easy, as it suffices to guess a nontrivial component (subset of the vertex set) and verify in first-order that it is nontrivial (not empty or equal V), and that there are no edges in the cut it defines. The second part requires a nice generalization of the Ehrenfeucht-Fraisse game to deal with such sentences.

It was somewhat surprising, and in some sense counter intuitive, when Kanellakis [**?**] observed that *USTCONN*, which we think of as harder than *UCONN*, is a monadic Σ_1^1 property.

Theorem 36 ([?]) *USTCONN is a monadic* Σ_1^1 *property.*

To see this, guess the set which contains the vertices on a shortest *st*-path. It is easy to see that one needs only verify (in first-order) that in the subgraph induced by this set s and t have degree 1, and every other vertex has degree 2. Trying to apply the same trick in directed graphs fails due to "back edges", and the status of *STCONN* was raised. This was resolved in the negative by Ajtai and Fagin [**?**].

Theorem 37 *STCONN* is not a monadic Σ_1^1 property.

Acknowledgments

I wish to thank Moni Naor, Ilan Newman and Noam Nisan for reading and improving an earlier version of this paper.

References

[A1] M. Ajtai, *On the complexity of the pigeonhole principle*, Proc. of the 29th FOCS, pp. 346–355, 1988.

[A2] M. Ajtai, *First-order definability on finite structures*, Annals of Pure and Applied Logic, 45, pp. 211–225, 1989.

[AB] M. Ajtai and M. Ben-Or, *A theorem on probabilistic constant-depth computation*, Proc. of the 16th STOC, pp. 471–474, 1984.

[AF] M. Ajtai and R. Fagin, *Reachability is harder for directed than for undirected finite graphs*, The journal of Symbolic Logic, Vol 55, No 1, pp. 113–150, 1990.

[AKS] M. Ajtai, J. Komlos, E. Szemeredi, *Deterministic simulation in logspace*, Proc. of the 19th STOC, pp. 132–140, 1987.

[AK+] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, C. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, Proc. of the 20th FOCS, pp. 218–223, 1979.

[BeSi] P. Berman and J. Simon, *Lower bounds for graph threading by probabilistic machines*, Proc. of the 24th FOCS, pp. 304–311, 1983.

[Bo] B. Bollobas, Extremal Graph Theory, Academic Press, 1978.

[BBRS] G. Barnes, J. F. Buss, W. L. Ruzzo and B. Schieber, *A sublinear space, polynomial time algorithm for directed $s-t$ connectivity*, Technical report 92-03-05, Dept. of Computer Science, University of Washington, 1992.

[BC+] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo and M. Tompa, *Two applications of inductive counting for complementation problems*, SIAM J. on Computing, Vol 18, pp. 559–578, 1989.

[BI+] P. Beame, R. Impagliazzo, J. Krajicek, T. Pitassi, P. Pudlak and A. Woods, *Exponential lower bounds for the pigeonhole principle*, Proc. of the 24th STOC, pp. 200–220.

[BM] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. on Computing, 13, 4, pp. 850–864, 1984.

[BNS] L. Babai, N. Nisan and M. Szegedy, *Multi-party protocols and logspace-hard pseudo-random sequences*, Proc. of the 21st STOC, pp.1–11, 1989.

[BR] G. Barnes and W. L. Ruzzo, *Deterministic algorithms for undirected st-connectivity using polynomial time and sublinear space*, Proc. 23rd STOC, pp. 43–53, 1991.

[BS] R. Boppana and M. Sipser, *The complexity of finite functions*, Handbook of Theoretical Compluter Science, Vol. A, van Leeuwen (ed.), MIT Press/ Elsvier, pp. 759–804, 1990.

[CaWe] L. Carter and M. Wegman, *Universal hash functions*, J. of Computer Systems and Sciences, 18, 2, pp. 143–154, 1979.

[Co] S. A. Cook, *A taxonomy of problems with fast parallel algorithms*, Information and Computation, 64, pp. 2–22, 1985.

[CW] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification and weak random sources*, Proc. of the 30th FOCS, pp. 14-19, 1989.

[CoWi] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Proc. of the 19th STOC, pp. 1–6, 1987.

[CFL] A. Chandra, M. Furst and R. J. Lipton, *Multi-party protocols*, Proc. of the 15th STOC, pp. 94–99, 1983.

[CKR] M. Chrobak, H. Karloff, T. Radzik, *Connectivity vs. reachability*, Information and Computation, Vol 91, No 2, pp. 177–188, 1991.

[CM] S. Cook and P. McKenzie, *Problems complete for deterministic logarithmic space*, J. of Algorithms, Vol 8, No 3, pp. 385–394, 1987.

[CR] S. A. Cook and C. W. Rackoff, *Space lower bounds for maze threadability on restricted machines*, SIAM J. on Computing, Vol 9, No 3, pp. 636–652, 1980.

[E] H. B. Enderton, A Mathematical Introduction to Logic, Academic Press, 1972.

[Eh] A. Ehrenfeucht, *An application of games to the completeness problem for formalized theories*, Fund. Math., 49, pp. 129–141, 1961.

[Fa] R. Fagin, *Monadic generalized spectra*, Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, Vol 21, pp. 89–96, 1975.

[Fr] R. Fraisse, *Sur les classifications des systems de relations*, Publications Scientifiques de l'Universite d'Alger, Vol 1. pp. 35–182, 1954.

[FSS] M. Furst, J. Saxe and M. Sipser, *Parity, circuits and the polynomial-time hierarchy*, Math System Theory 17, pp. 13–27, 1984.

[GS1] M. Grigni and M. Sipser, *Monotone complexity*, Proceedings of LMS workshop on Boolean function complexity, Durham, M. Paterson (Ed.), Cambridge University Press, 1990.

[GS2] M. Grigni and M. Sipser, *Monotone separation of Logspace from NC^1*, Proc. of the 6th Structures in Complexity Theory conference, pp. 294–298, 1991.

[H] J. Hastad, Computational limitations of small-depth circuits, The MIT Press, 1987.

[Is] S. Istrail, *Polynomial traversing sequences for cycles are constructible*, Proc. of the 20th STOC, pp. 491–503, 1988.

[I1] N. Immerman, *Descriptive and computational complexity*, Computational Complexity Theory, J. Hartmanis (Ed.), Proc. Symp. Applied Math. 38, American Mathematical Society, pp. 75–91, 1989.

[I2] N. Immerman, *Nondeterministic space is closed under complementation*, SIAM J. on Computing, 17, pp. 935–938, 1988.

[IZ] R. Implagliazzo and D. Zuckerman, *How to recycle random bits*, Proc. of the 30th FOCS, pp. 248–253, 1989.

[J] D. Johnson, *A catalog of complexity classes*, Handbook of Theoretical Compluter Science, Vol. A, van Leeuwen (ed.), MIT Press/ Elsvier, pp. 67–162, 1990.

[Ka] P. Kanellakis, Private communication, 1986.

[KSS] J. Kahn, M. Saks, D. Sturtevant, *A topological approach to evasiveness*, Combinatorica 4, pp. 297–306, 1984.

[KW] M. Karchmer and A. Wigderson, *Monotone circuits for connectivity require superlogarithmic depth*, SIAM J. on Discrete Mathematics, Vol 3, No 2. pp. 255–265, 1990.

[LP] H. Lewis and C. Papadimitriu, *Symmetric space-bounded computation*, Theoretical Computer Science 25, pp. 130–143, 1982.

[N1] N. Nisan, *Pseudo-random generators for space-bounded computation*, Proc. of the 22nd STOC, pp. 204–212, 1990.

[N2] N. Nisan, $RL \in SC$, Proc. of the 24th STOC, pp. 619–623, 1992.

[NN] J. Naor and M. Naor, *Small-bias probability spaces: efficient constuctions and applications*, Proc. of the 22nd STOC, pp. 213–223, 1990.

[NSW] N. Nisan, E. Szemeredi and A. Wigderson, *Undirected connectivity in* $O(\log^{1.5} n)$ *space*, submitted to FOCS '92.

[NW] N. Nisan and A. Wigderson, *Hardness vs. Randomness*, Proc. of the 29th FOCS, pp. 2–12, 1988.

[RW1] R. Raz and A. Wigderson, *Probabilistic communication complexity of Boolean relations*, Proc. of the 30th FOCS, pp. 562–567, 1989.

[RW2] R. Raz and A. Wigderson, *Monotone circuits for matching require linear depth*, Proc. of the 22nd STOC, pp. 287–292, 1990.

[Re] J. H. Reif, *Symmetric complementation*, Proc. of the 14th STOC, pp. 201–214, 1982.

[S] R. Szelepcsenyi, *The method of forcing for nondeterministic automata*, Bull. of the European Ass. of Theoretical Computer Science, 33, pp. 96–100, 1987.

[Sa] W. Savitch, *Relashionships between nondeterministic and deterministic tape complexities*, Journal of Computer Systems and Sciences, 4, pp. 177–192, 1970.

[SV] S. Skyum and L. Valiant, *A complexity theory based on Boolean algebra*, Proc. of the 22nd FOCS, pp. 244–253, 1981.

[T] M. Tompa, *Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations*, SIAM J. on Computing, 11, 1, pp. 130–137, 1982.

[Y1] A. C. Yao, *Some complexity questions related to distributive computing*, Proc. of the 11th STOC, pp. 209–213, 1979.

[Y2] A. C. Yao, Private communication.