

Planar and Grid Graph Reachability Problems

Eric Allender* David A. Mix Barrington† Tanmoy Chakraborty‡ Samir Datta§
Sambuddha Roy¶

Abstract

We study the complexity of restricted versions of s - t -connectivity, which is the standard complete problem for NL. In particular, we focus on different classes of planar graphs, of which grid graphs are an important special case. Our main results are:

- Reachability in graphs of genus one is logspace-equivalent to reachability in grid graphs (and in particular it is logspace-equivalent to both reachability and non-reachability in planar graphs).
- Many of the natural restrictions on grid-graph reachability (GGR) are equivalent under AC^0 reductions (for instance, undirected GGR, outdegree-one GGR, and indegree-one-outdegree-one GGR are all equivalent). These problems are all equivalent to the problem of determining whether a completed game position in HEX is a winning position, as well as to the problem of reachability in mazes studied by Blum and Kozen [BK78]. These problems provide natural examples of problems that are hard for NC^1 under AC^0 reductions but are not known to be hard for L; they thus give insight into the structure of L.
- Reachability in layered planar graphs is logspace-equivalent to layered grid graph reachability (LGGR). We show that LGGR lies in UL (a subclass of NL).
- Series-Parallel digraphs (on which reachability was shown to be decidable in logspace by Jakoby et al.) are a special case of single-source-single-sink planar directed acyclic graphs (DAGs); reachability for such graphs logspace reduces to single-source-single-sink acyclic grid graphs. We show that reachability on such grid graphs AC^0 reduces to undirected GGR.
- We build on this to show that reachability for single-source multiple-sink planar DAGs is solvable in L.

1 Introduction

Graph reachability problems play a central role in the study and understanding of subclasses of P. The s - t -connectivity problem for directed graphs (STCONN) is complete for nondeterministic logspace (NL); the restriction of this problem to undirected graphs, called USTCONN, was shown by Reingold to be complete for logspace (L) [Rei05]; thus this problem has the same complexity as the s - t -connectivity problem for graphs of outdegree 1 (and even for graphs of indegree and outdegree at most 1 [CM87, Imm87, Ete97]). It follows from [Bar89] that reachability in directed graphs of width $O(1)$ (or even width five, with outdegree 1) is complete for NC^1 .

*Department of Computer Science, Rutgers, the State University of NJ. Supported in part by NSF Grant CCF-0514155. email: allender@cs.rutgers.edu.

†Computer Science Dept., University of Massachusetts Amherst. Supported in part by NSF Grant CCR-9988260. e-mail: barrington@cs.umass.edu.

‡Computer and Information Science Department, University of Pennsylvania e-mail: tanmoych1985@gmail.com.

§Chennai Mathematical Institute, Chennai, India. e-mail: sdatta@cmi.ac.in.

¶IBM India Research Lab, New Delhi, India. Supported in part by NSF Grant CCF-0514155. email: sambuddha@in.ibm.com.

1.1 Planar Graphs

Our focus in this paper is the restriction of `STCONN` to planar (directed) graphs: `PLANAR.STCONN`. This problem is hard for `L` under uniform projections, as a consequence of [Ete97], and it lies in `NL`. Although there are a number of papers presenting efficient algorithms for connectivity in planar graphs (such as [Hus95, EIT⁺92, HKRS97]), little is known about the computational complexity of this problem. Prior to our work, the best upper bound known for `PLANAR.STCONN` was `NL`. Building on our work, Bourke, Tewari, and Vinodchandran recently showed that the GGR problem studied here is in Unambiguous Logspace (`UL`) [BTV07]. It follows that the class of problems \leq_m^{\log} -reducible to `PLANAR.STCONN` can be viewed as a complexity class lying between `L` and `UL`.

Grid graphs are an important restricted class of graphs for which the reachability problem has significant connections to complexity classes. (The vertices in a grid graph are a subset of $\mathbb{N} \times \mathbb{N}$, and all edges are of the form $(i, j) \rightarrow (i + b, j)$ or $(i, j) \rightarrow (i, j + b)$, where $b \in \{1, -1\}$.) In [BLMS98], Barrington *et al.* showed that the reachability problem in (directed or undirected) grid graphs of width k captures the complexity of depth k `AC0`. In this paper we study grid graphs without any width restrictions. The construction of [BLMS98, Lemma 13] shows that GGR reduces to its complement via uniform projections. (The problems `STCONN` and `USTCONN` also reduce to their complements via uniform projections, as a consequence of [Imm88, Sze88, Rei05, NTS95].) Reachability problems for grid graphs have proved easier to work with than the corresponding problems for general graphs. For instance, the reachability problem for *undirected* grid graphs (`UGGR`) was shown to lie in `L` in the 1970's [BK78], although more than a quarter-century would pass before Reingold proved the corresponding theorem for general undirected graphs.

We show that `PLANAR.STCONN` is logspace-equivalent to GGR (and consequently it is logspace-reducible to its complement). (It had already been shown in [HRS93] that a special case of `PLANAR.STCONN` is logspace-reducible to its complement.) We do not know whether this reduction can be accomplished by uniform projections or even by `NC1` reductions; in contrast to the case for `STCONN`, `USTCONN`, and GGR. We also show that the s - t -connectivity problem for graphs of genus one is logspace reducible to `PLANAR.STCONN`; the generalization for graphs of higher genus remains open.

1.2 Restrictions of Grid Graphs

We consider several natural restrictions of GGR in this paper. We have already mentioned `UGGR` (undirected grid graph reachability). Buss has studied `UGGR` in connection with tautologies arising from the game of HEX [Bus06] (namely, the tautology that every completed game board of HEX has a winner); he credits Barrington with the observation that `UGGR` is equivalent to the problem of determining whether a given completed HEX board position is a win for one player. Reachability in grid graphs of outdegree one (`1GGR`) is another restriction on GGR that is clearly solvable in logspace.

One of our theorems is that `UGGR` and `1GGR` are equivalent under `AC0` reductions (and even under first-order projections). We show that these problems are hard for `NC1`, and thus this gives a cluster of natural problems that are candidates for having complexity intermediate between `NC1` and `L`, since even the general GGR problem is not known to be hard for `L` under `AC0` reductions.

A graph is said to be *layered* if the vertex set is partitioned into “layers”, where all edges from vertices in layer i have destinations in layer $i + 1$. Just as general GGR is logspace-equivalent to reachability in planar digraphs, we observe that reachability in *layered* planar digraphs is logspace equivalent to the “layered” grid graph reachability problem (`LGGR`). In an instance of `LGGR`, all edges are directed either “east” or “south”. Thus without loss of generality, the start node is in the top left corner. If such a grid graph is rotated 45 degrees counterclockwise, one obtains a graph whose “columns” correspond to the diagonals of the original graph, where s is the only node in the first “column”, and all edges in one column are directed “northeast” or “southeast” to their neighbors in the following column. This is consistent with the usual usage of the word “layered” in graph theory.

We show that `LGGR` lies in a subclass of `NL` known as `UL`. That is, `LGGR` is accepted by a nondeterministic logspace machine that never has more than one accepting computation path on a given input. Note that the improvement from `NL` to `UL` is, at best, a very slight improvement; it is known [RA00] that the non-uniform versions of `UL` and `NL` are the same, and it is entirely plausible that the classes themselves are the same. In particular, it is shown in [ARZ99] that `NL` = `UL` if there is any problem in `DSPACE(n)` that requires circuits of exponential size. We actually show that `LGGR` lies in `UL` \cap `coUL`, since (in contrast to nearly all of the other reachability problems we consider) it remains open whether `LGGR` reduces to its complement. (Note also that it remains open whether `UL` = `coUL`.) The work of Bourke *et al.* [BTV07] showing that `PLANAR.STCONN` \in `UL` builds on this theorem of ours. Subsequently, `UL` has been shown to contain other graph-theoretic problems of interest [TW08]. Some other examples of reachability problems in `UL` were presented earlier by Lange [Lan97];

these problems are obviously in UL (in the sense that the positive instances consist of certain graphs that contain only one path from s to t), and the main contribution of [Lan97] is to present a *completeness* result for a natural subclass of UL. In contrast, positive instances of LGGR can have many paths from s to t . We know of no reductions (in either direction) between LGGR and the problems considered in [Lan97].

1.3 Reachability Problems in Logspace

Jakoby, Liskiewicz, and Reischuk showed that reachability in series-parallel digraphs is solvable in logspace [JLR06], thus solving the reachability question for an important subclass of planar directed graphs. (They also show the much stronger result that *counting* the number of paths between s and t can be carried out in logspace for series-parallel graphs.) Series-parallel digraphs are a special case of planar directed acyclic graphs having a single source and single sink. Motivated by a desire to solve the reachability problem for a larger class of planar DAGs, we introduce the following three classes of DAGs:

- *Single-Source Single-Sink Planar DAGs* (SSPDs): the class of DAGs having one vertex of indegree zero and one vertex of outdegree zero. Reachability in SSPDs generalizes the problem of reachability in series-parallel digraphs studied in [JLR06].
- *Single-Source Multiple-Sink Planar DAGs* (SMPDs): the class of DAGs having one vertex of indegree zero. Reachability in such graphs is clearly equivalent to reachability in Multiple-Source Single Sink DAGs (MSPDs) by simply reversing all of the edges.
- *Multiple-Source Multiple-Sink Planar DAGs* (MMPD). This is simply the class of all planar DAGs.

We show that the SMPD reachability problem (and hence also that for MSPD) lies in logspace. In addition, reachability in SSPDs, restricted to grid graphs, is reducible to UGGR. Our algorithmic approach for SMPD extends to certain classes of graphs that are not acyclic. This is discussed in more detail in Section 9.

The rest of the paper is organized as follows. After a few preliminaries (Section 2), we begin by presenting results related to PLANAR.STCONN. In Section 3 we show that PLANAR.STCONN reduces to a special case where s and t lie on the external face. This is useful in presenting our reduction from PLANAR.STCONN to GGR in Section 4. In Section 5 we prove a closure property of the class of sets logspace reducible to PLANAR.STCONN, and then in Section 6 we show that planar reachability is equivalent to reachability in genus one graphs. Next, in Section 7 we introduce the various restricted grid graph problems that we will be considering, and present reductions showing how these problems relate to each other. In Section 7.3 we present a generic reduction showing that, for many of the problems we consider, it is no loss of generality to assume that s and t appear on the external boundary of the graph. (In some sense this is reminiscent of the results of Section 3). Our hardness results are presented in Section 8. Our logspace algorithms for SSPD and SMPD are presented in Section 9. We conclude with open questions in Section 10.

2 Classes and Reductions

We assume familiarity with the following important subclasses of nondeterministic logspace (NL): L, NC^1 , TC^0 , and AC^0 . When defining notions of reducibility and completeness in order to investigate the structure of such small complexity classes, some form of AC^0 reducibility is usually employed. We will frequently make use of the terminology and notation employed by Immerman [Imm98], which exploits the close connections between AC^0 and first-order logic. In particular, AC^0 -Turing reducibility ($\leq_T^{AC^0}$) to a set A can be defined equivalently in terms of AC^0 circuits augmented with “oracle gates” for A , or in terms of first-order formulae with A as a built-in predicate symbol applied to a structure defined in first-order. For details refer to [Imm98]. For this reason, we sometimes refer to $\leq_T^{AC^0}$ reductions as FO reductions. The class of problems $\leq_T^{AC^0}$ reducible to A is sometimes denoted as $FO + A$.

Immerman also gives good motivation for studying a restricted form of $\leq_m^{AC^0}$ reductions called *first-order projections* (\leq_{proj}^{FO}). These can be visualized as many-one reductions computed by first-order uniform circuits *having no gates* (other than NOT gates); thus each bit of the output is either a constant or is a copy (or a negated copy) of one bit of the input. For example, the class depth- k AC^0 is closed under these reductions.

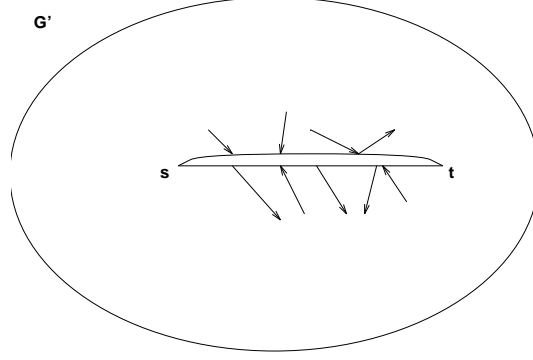


Figure 1. Cutting along a st path

3 Planar Reachability: Reduction to a Special Case

Theorem 1 PLANAR.STCONN is logspace reducible to the special case where the vertices s and t both lie on the external face of the planar graph.

Proof: Let G be a directed graph. Testing whether G is planar reduces to the undirected s - t -connectivity problem [AM04] and thus can be done in logarithmic space [Rei05]. Furthermore, if a graph is planar then a planar combinatorial embedding (i.e., a cyclic ordering of the edges adjacent to each vertex) can be computed in logarithmic space [AM04]. Given a combinatorial embedding, it is easy to check whether two vertices lie on the same face. (The vertices on each face adjacent to a vertex v can be enumerated by starting at some (undirected) edge adjacent to v and starting a walk from v along that edge; each time a new vertex w is entered along some edge e the walk continues along the edge that succeeds e in the cyclic ordering of edges around w .) Thus in logspace we can check whether s and t lie on the same face. If so, then the graph G is already in the desired form, since we can consider any face to be the “external” face in the embedding.

If s and t do not lie on the same face, then by use of the undirected connectivity algorithm we can determine whether there is an undirected path from s to t . If there is no such path, then clearly there is no directed path, either. Otherwise (as observed in [AM04]) we can find a simple undirected path $\mathcal{P} = (s, v_1, v_2, \dots, v_m, t)$ in logspace. First, we construct a new face with s and t on it, by “cutting” along the path \mathcal{P} . That is, we replace each vertex v_i on \mathcal{P} by vertices $v_{i,a}$ and $v_{i,b}$. For any vertex v_i on \mathcal{P} , let u and x be the vertices appearing before and after v_i on \mathcal{P} ; that is, $u \in \{s, v_{i-1}\}$ and $x \in \{t, v_{i+1}\}$. Let e_1, \dots, e_{d_a} be the edges embedded “above” the edges connecting v_i to u and x in the cyclic ordering around v_i , and let e'_1, \dots, e'_{d_b} be the edges embedded “below” the edges between v_i and u and x . That is, if an observer moves along the undirected path from s to t , edges e_1, \dots, e_{d_a} appear on the observer’s left and edges e'_1, \dots, e'_{d_b} appear on its right. Let \mathcal{L} be the set of all edges adjacent to \mathcal{P} embedded on the observer’s left, and let \mathcal{R} be the set of all edges adjacent to \mathcal{P} embedded on the observer’s right. In the new graph, the edges in \mathcal{L} that were connected to v_i are connected to $v_{i,a}$ and those in \mathcal{R} are connected to $v_{i,b}$. Edges between v_i and $\{v_{i+1}, v_{i-1}\}$ are duplicated, with edges between $v_{i,c}$ and $\{v_{i+1,c}, v_{i-1,c}\}$ for $c \in \{a, b\}$. Similarly, edges between s and v_1 (and t and v_m) are duplicated, with edges between s and $v_{1,a}$ and $v_{1,b}$ (and edges between t and $v_{m,a}$ and $v_{m,b}$, respectively). This is illustrated in Figure 1.

This new graph G' is planar, and has vertices s and t on the same face (the only new face created). Since we can embed any planar graph such that any specific face is the outer face, we re-embed our graph G' such that s and t are now on the outer face. From now on we assume G' has this embedding.

In the process of going from G to G' we may have changed the connectivity of the graph; s and t may be connected in G but not connected in G' . In particular, any directed path in G from s to t that uses edges from *both* \mathcal{L} and \mathcal{R} is not replicated in G' . We solve this problem by pasting together copies of the graph G' , as follows. The outer face of G' consists of two undirected paths from s to t : $s, v_{1,a}, v_{2,a}, \dots, v_{m,a}, t$ and $s, v_{1,b}, v_{2,b}, \dots, v_{m,b}, t$. The operation of “pasting” two copies of G' together consists of identifying the vertices $v_{1,a}, v_{2,a}, \dots, v_{m,a}$ in one copy with the vertices $v_{1,b}, v_{2,b}, \dots, v_{m,b}$ in the other copy. (Note that this amounts to “sewing together” two copies of the path that were “cut apart” in creating G' from G .) The graph G'' consists of $2n + 1$ copies of G' pasted together in this way: the “original copy” in the middle, and n copies pasted in sequence to the top boundary of the outer face, and n copies pasted in sequence to the bottom boundary.

G'' has (the original copies of) s and t on the outer face. A simple inductive argument shows that there is a directed path from s to t in G if and only if there is a directed path from (the original copy of) s to one of the copies of t in G'' .

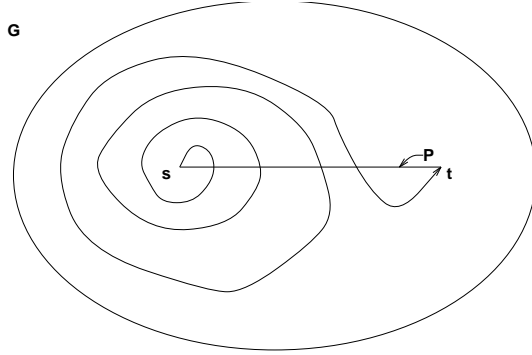


Figure 2. A pathological case

A pathological example showing that many copies of G' are needed is shown in Figure 2. To complete the reduction, we construct a graph H that consists of G'' along with a new vertex t'' with directed edges from each copy of t to t'' . The vertices s and t'' appear on the external face of H , and there is a directed path from s to t in G if and only if there is a directed path from s to t'' in H .

□

Later in the paper, we find it useful to prove a similar theorem about grid graphs. Hence the material in Section 7.3 has a very similar flavor to the material presented here.

4 Grid Graphs

In this section, we present a \leq_m^{\log} reduction of PLANAR.STCONN to GGR.

Using the reduction of Section 3, we may assume that we are given a planar graph G with s and t on the external face. By the following simple local transformation we can eliminate any bidirectional edges: If (x, y) and (y, x) are both edges in the graph, introduce two new vertices u and v and replace those two edges with (x, u) , (u, y) , (y, v) , and (v, x) – note that this transformation preserves planarity of the graph. We may also assume without loss of generality that G has no vertex of degree (indegree + outdegree) greater than 3, and that s has degree two. (To see this, observe that if v is a vertex of degree $d > 3$, then we may replace v with d vertices arranged in a directed cycle, with each one adjacent to one of the d edges that were connected to v . In order to compute this transformation it is important to note that we can compute the planar embedding in logspace. If the vertex s has degree three, then an additional vertex of degree two can be inserted into this cycle, and re-named s .)

Compute an (undirected) spanning tree T of G ; it follows from [NTS95, Rei05] that this can be done in logspace. The vertex s is a vertex of T , and we can consider it to be the root of T ; without loss of generality s has two children in T . By our assumptions on G , the tree T is a binary tree; the planar embedding of G imposes an ordering on the children of each node in T . As observed in [AM04], we can compute the height $h(v)$ of each node v in T in logspace (by counting the number of vertices that are ancestors of v). For notational convenience, define the height of the root s to be 1, and if v has child u then $h(u) = h(v) + 1$.

At this point, we are ready to assign each vertex of G to a grid point. Our grid graph will consist of a “fine grid” and a “coarse grid”. The coarse grid consists of points placed at the corners of large squares (of size $(4n+1) \times (4n+1)$) of the fine grid. (The fine grid will be used to route non-tree edges between vertices placed on the coarse grid.) For any node x , define $w(x)$ to be the number of leaves of T that appear strictly to the left of x . In other words, given a node x in the tree, consider the path from the root s to the parent of x . Among these proper ancestors of x consider the subset $S_R(x)$ of those nodes which contain x in their right subtree. Then $w(x)$ is the sum of the number of leaves in each subtree rooted at a node $r \in S_R(x)$. For two nodes x and y at the same height, where $l(x, y)$ is the least common ancestor of x, y , then (assuming that x lies in the left subtree of $l(x, y)$), $S_R(x) - S_R(y)$ consists of some proper descendants of $l(x, y)$ while $S_R(y) - S_R(x)$ certainly contains $l(x, y)$. So in fact, $w(x) \leq w(y)$. In fact, the right-most leaf of the left subtree of $l(x, y)$ is counted in $w(y)$ but not in $w(x)$. Hence, $w(x) < w(y)$. Notice that the assumption that $h(x) = h(y)$ is crucial because if, for example, x is the left child of y then $w(x) = w(y)$. Thus the partial order on nodes x imposed by the lexicographic order on $(h(x), w(x) + 1)$ is, in fact, a total order. It is easy to see that $w(x)$ can be computed easily in logspace by traversing T .

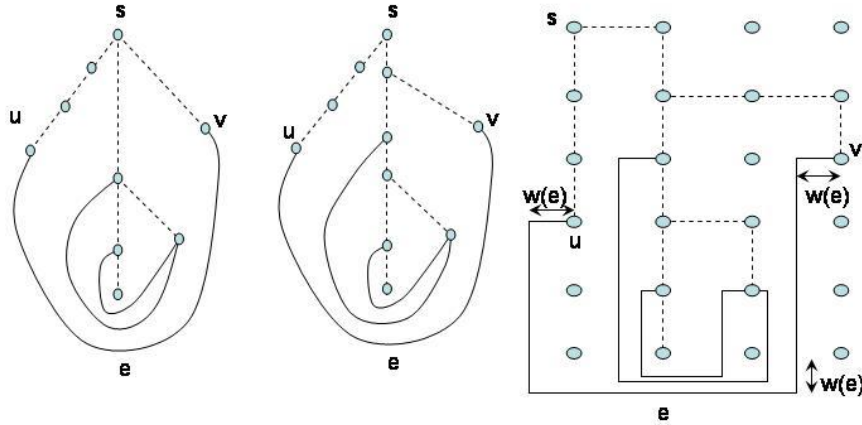


Figure 3. Embedding a graph on the grid. Edges used in the spanning tree are shown as dashed lines; non-tree edges are solid.

Each vertex x is assigned to the point $(h(x), w(x) + 1)$ in the coarse grid; note that the root s is placed at the top left corner $(1, 1)$. (Note that here and elsewhere in the paper, we consider the positive x direction to be “down” or “south”, and the positive y direction to be “right” or “east”.) If node x is at position (i, j) in the coarse grid, then the tree edge from x to its left child is embedded as a vertical path to point $(i + 1, j)$ in the coarse grid. If x also has a right child y , then this edge is embedded as a horizontal path to location $(i, w(y) + 1)$ followed by a vertical path to location $(i + 1, w(y) + 1)$ in the coarse grid. This is illustrated in Figure 3.

For every non-tree edge e in the tree we can find the number $w(e)$ of non-tree edges enclosed by the unique cycle formed by adding e to the tree. (For edge $e = (u, v)$, $w(e)$ can be computed by finding the least common ancestor y of u and v and determining for each non-tree edge connected to a descendant of y whether it is embedded to the right or left of the paths between y and u and v .) For any non-tree edge $e = (u, v)$, note that u and v have degree at most two in the tree T , and thus there is no tree edge attached horizontally adjacent to u or v . The embedding determines whether the path representing e should be attached to the east or west sides of u and v . If the embedding goes around a leaf z of the tree T , then the path is routed horizontally from u to a location $w(e)$ fine grid points to the east or west of the column containing z , and vertically down to a point $w(e)$ fine grid points below the level of the leaf of maximum height, and from there horizontally to a point $w(e)$ fine grid points east or west of the column containing v , then vertically to the level of v , and then horizontally to attach to v . If the embedding does not go around a leaf, then a simpler path can be drawn: horizontally to a point $w(e)$ fine grid points east or west of v , then vertically to the level of v , and then horizontally to connect to v . It is easy to verify that no paths collide in this way. See Figure 3 for an example.

Thus we have the following theorem.

Theorem 2 $\text{PLANAR.STCONN} \leq_m^{\log} \text{GGR}$

Combining this theorem with [BLMS98, Lemma 13], we obtain this corollary:

Corollary 3 PLANAR.STCONN is \leq_m^{\log} reducible to its complement.

5 More Closure Properties for PLANAR.STCONN

We have identified an important problem PLANAR.STCONN that may or may not be in the class L . If it is not, it is by definition complete under logspace reductions for a complexity class. Is this class robust under different types of logspace reductions? We are able to give a partial affirmative answer. Let us first remind the reader about some relevant definitions.

Different types of logspace reductions were introduced and studied by Ladner and Lynch [LL76], who showed that logspace Turing and truth-table reducibilities coincide ($A \leq_T^{\log} B$ iff $A \leq_{tt}^{\log} B$). They also introduced a more restrictive version of logspace-computable truth-table reducibility, known as logspace Boolean formula reducibility \leq_{bf-tt}^{\log} . $A \leq_{bf-tt}^{\log} B$ if there is a logspace computable function f such that $f(x) = (q_1, q_2, \dots, q_r, \phi)$ where each q_i is a query and ϕ is a Boolean formula with r variables y_1, \dots, y_r , such that $x \in A$ if and only if ϕ evaluates to 1 when the variables y_i are assigned the truth value of the statement “ $q_i \in B$ ”. Additional results about this type of reducibility can be found in [BST93, BH91].

Corollary 4 $A \leq_m^{\log} \text{PLANAR.STCONN}$ if and only if $A \leq_{bf-tt}^{\log} \text{PLANAR.STCONN}$.

Proof: One direction is trivial; thus assume that $A \leq_{bf-tt}^{\log} \text{PLANAR.STCONN}$. For a given input x , let $f(x) = (q_1, q_2, \dots, q_r, \phi)$ be the result of applying the reduction to x . Without loss of generality, the formula ϕ has negation operations only at the leaves (since it is easy in logspace to apply DeMorgan’s laws to rewrite a formula). Using closure under complementation, we can even assume that there are no negation operations at all in the formula. By the results of Section 3, we can assume that each graph q_i is a planar graph with s and t on the external face. Given two such graphs G_1, G_2 , note that both G_1 and G_2 are in PLANAR.STCONN if and only if the graph with the terminal vertex of G_1 connected to the start vertex of G_2 is in PLANAR.STCONN , and thus it is easy to simulate an AND gate. Similarly, an OR gate can be simulated by building a new graph with start vertex s connected to the start vertices of both G_1 and G_2 , and with edges from the terminal vertices of G_1 and G_2 to a new vertex t . These constructions maintain planarity, and they also maintain the property that s and t are on the external face. Simulating each gate in turn involves only a constant number of additional vertices and edges, and it is easy to see that this gives rise to a \leq_m^{\log} reduction. \square

A natural open question is whether the preceding corollary also holds, when \leq_{bf-tt}^{\log} is replaced by \leq_T^{\log} .

6 Higher Genus

In this section we prove that the s - t -connectivity problem for graphs of genus one reduces to the planar case. Throughout this section, we will assume that we are given an embedding Π of a graph G onto a surface of genus one. (Unlike the planar case, it does not appear to be known if testing whether a graph has genus $g > 0$ can be accomplished in logspace, even for $g = 1$ [MV00].) Given such an embedding, using [AM04], we can check in logspace whether the minimal genus of the graph is one.

We introduce here some terminology and definitions relating to graphs on surfaces. It will be sufficient to give informal definitions of various notions; the interested reader can refer to [MT01] for more rigorous definitions.

A *closed orientable* surface is one that can be obtained by adding handles to a sphere in 3-space. The genus of the resulting surface is equal to the number of handles added; see also the text [GT87]. Given a graph G , the genus of the graph is the genus of the (closed orientable) surface of least genus on which the graph can be embedded.

Given a graph G embedded on a closed orientable surface, and a cycle of the graph embedded on the surface, there are two (possibly intersecting) subgraphs, called the two *sides* of the cycle with respect to the embedding. Informally, a side of a cycle is the set of vertices of the graph that are path-connected (via a path in the graph, each edge of the graph being considered regardless of direction) to some vertex on the cycle, such that this path does not cross the cycle itself. (In the considerations below, we are concerned only with genus one graphs for which this notion of path-connectivity suffices.) A cycle thereby has two sides, which are called the *left* and the *right* sides. If the left and right sides of a cycle have nonempty intersection, then we call the cycle a *surface-nonseparating cycle*. Note that a graph embedded on a sphere (i.e., a planar graph) does not have any surface-nonseparating cycles. Also, it is easy to see that a *facial* cycle (one that forms the boundary of a face in the embedding of the graph on the surface) cannot be surface-nonseparating. Given a cycle C in an embedded graph, it is easy to check in logspace whether C is surface-nonseparating: merely check whether there is a vertex $v \in G$, such that v is path-connected to both sides of C (on the embedding).

Lemma 5 *Let G be a graph of genus $g > 0$, and let T be a spanning tree of G . Then there is an edge $e \in E(G)$ such that $T \cup \{e\}$ contains a surface-nonseparating cycle.*

Proof: The proof follows ideas from [Tho90] which introduces the “3-path condition”:

Definition 6 *Let \mathcal{K} be a family of cycles of G as follows. We say that \mathcal{K} satisfies the 3-path condition if it has the following property. If x, y are vertices of G and P_1, P_2, P_3 are internally disjoint paths joining x and y , and if two of the three cycles $C_{i,j} = P_i \cup P_j$ ($1 \leq i < j \leq 3$) are not in \mathcal{K} , then also the third cycle is not in \mathcal{K} .*

We quote the following from [MT01].

Proposition 7 (*Proposition 4.3.1 of [MT01]*) *The family of Π -surface-nonseparating cycles satisfies the 3-path condition.*

Suppose, that $\forall e, (T \cup \{e\})$ does not have a surface-nonseparating cycle. We will prove that no cycle C in the graph G can be surface-nonseparating, by induction on the number k of non-tree edges in C . This contradicts the fact that every non-planar graph has a surface-nonseparating cycle ([MT01, Lemma 4.2.4 and the following discussion]) and thus suffices to prove the claim.

The basis ($k = 1$) follows from the above supposition.

For the inductive step from $k - 1$ to k , let a cycle C be given with k edges not in T .

Take any non-tree edge $e = (x, y)$ on C . Consider the tree path P between x and y . If P never leaves the cycle C , then C is a fundamental cycle and we are done by the assumption for $k = 1$. Otherwise, we can consider a maximal segment S of P not in C . Let S lie between vertices u and v of C . Now, we have three paths between u and v : the two paths between u and v on C (call these C_1, C_2), and path S . Note that both $S \cup C_1$ and $S \cup C_2$ have fewer than k non-tree edges. Hence they are not surface-nonseparating cycles by the induction assumption. So, by the 3-path condition, neither is $C = C_1 \cup C_2$.

This completes the induction, and the proof. \square

At this point we are able to describe how to reduce the s - t -connectivity problem for graphs of genus one to the planar case.

Given a graph G of genus one and an embedding Π of G onto the torus, construct an (undirected) spanning tree T of G . For each edge e of G that is not in T , determine whether the unique cycle C_e in $T \cup \{e\}$ is surface-nonseparating, as follows.

Let $C_e = \{v_1, v_2, \dots, v_r\}$. Let G_e be the graph obtained from G by *cutting along* the cycle C_e (as described in [MT01, p. 105]). (For the purposes of visualization, it is useful to imagine cycles as embedded on an inner tube. Cutting along a surface-separating cycle amounts to cutting a hole in the inner tube (resulting in two pieces). In contrast, if C_e is surface-nonseparating, then it is embedded either like a ribbon tied around the tube, or like a whitewall painted on the inner tube. In the former case, cutting along C_e turns the inner tube into a bent cylinder with a copy of C_e on each end; in the latter case cutting along C_e results in a flat ring with one copy of C_e around the inside and one around the outside. In this latter case, the graph is again topologically equivalent to a cylinder with a copy of C_e on each side.) More formally, the graph G_e has two copies of each of the vertices $\{v_1, v_2, \dots, v_r\}$, which we denote by $\{v_{1,1}, v_{2,1}, \dots, v_{r,1}\}$, and $\{v_{1,2}, v_{2,2}, \dots, v_{r,2}\}$. For every edge (u, v_j) (or (v_j, u)) on the right side of C_e (according to Π), G_e has the edge $(u, v_{j,1})$ ($(v_{j,1}, u)$, respectively), and for every edge (u, v_j) ((v_j, u) , respectively) on the left side of C_e we have the edge $(u, v_{j,2})$ (or $(v_{j,2}, u)$) in G_e . The graph G_e also has two copies of the cycle C_e , which we denote by $C_{e,1}$ and $C_{e,2}$. That is, we have edges between $v_{j,b}$ and $v_{j+1,b}$ for each $b \in \{1, 2\}$ and each $1 \leq j \leq r$, directed as in C_e . An important property of cutting along the cycle C_e is that if C_e was surface-nonseparating, then the resulting graph G_e is planar, and the the cycles $C_{e,1}$ and $C_{e,2}$ are *facial* cycles ([MT01, p. 106, Lemma 4.2.4]). (Otherwise, G_e will not be planar.) Thus in logspace we can determine whether C_e is surface-nonseparating.

By Lemma 5, we are guaranteed to find a surface-nonseparating cycle by testing each edge e that is not in T . The graph G_e does not have the same connectivity properties as G ; s and t might have been connected in G but not in G_e . In particular, any directed path in G from s to t that uses edges from *both* the right and left sides of C_e is not replicated in G_e . As in Section 3, we solve this problem by pasting together copies of the graph G_e , as follows. The operation of “pasting” two copies of G_e together consists of identifying the vertices $v_{1,1}, v_{2,1}, \dots, v_{r,1}$ in one copy with the vertices $v_{1,2}, v_{2,2}, \dots, v_{r,2}$ in the other copy. (Note that this amounts to “sewing together” two copies of the path that were “cut apart” in creating G_e from G .)

Now construct the graph G' consisting of $2n + 1$ copies of G_e pasted together in this way: the “original copy” in the middle, and n copies along each side, forming one long cylinder. Since this cylinder has genus zero, it is easy to see that G' is planar.

As in Section 3, a simple inductive argument shows that there is a directed path from s to t in G if and only if there is a directed path from (the original copy of) s to one of the copies of t in G' . Thus we have presented a logspace-computable disjunctive truth-table reduction to the planar directed s - t -connectivity problem. We obtain a many-one reduction by appeal to Corollary 4. Thus we have proved the following theorem.

Theorem 8 *The s - t -connectivity problem for graphs of genus one is \leq_m^{\log} reducible to the planar directed s - t -connectivity problem.*

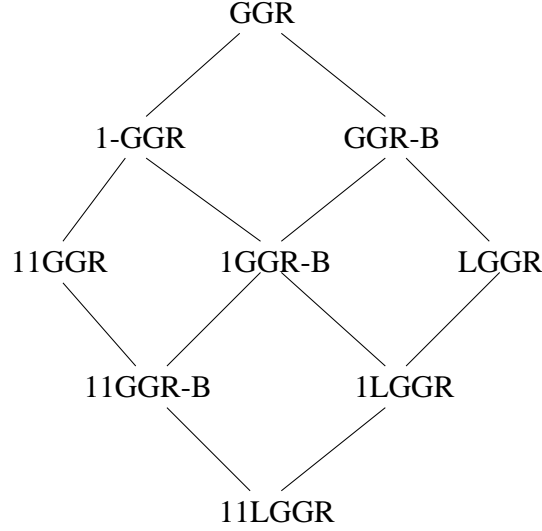


Figure 4. Nine GGR problems.

7 Versions of the GGR Problem

Up until this point in the paper, we have not needed to be very careful about the way that grid graphs are encoded. However, many of the results in this and later sections discuss completeness and equivalence under first-order projections – and in order for us to easily establish reducibility via projections in some instances, we must impose some restrictions upon the encoding. It will be clear to the reader that the encodings that we use are equivalent to other, more natural, encodings, under $\leq_m^{AC^0}$ reductions. As a consequence of Propositions 15 and 16, it follows that the more natural encodings are, in fact, reducible to these more restricted versions under projections, and hence they are equivalent under first-order projections.

An instance of the general GGR problem consists of a graph G on an n -by- m grid, along with two distinguished vertices s and t . However, for the rest of this paper, we will restrict attention to graphs on a grid of size $3n$ -by- $3n$ with s in position (n, n) and t in position $(2n, 2n)$. It is easy to see how to transform an arbitrary grid graph instance into an instance with s and t in these designated positions via an $\leq_m^{AC^0}$ reduction, by “stretching” (and possibly reflecting) the original grid.

We continue by defining and exploring a number of special cases of the GGR problem, based on a variety of restrictions on the grid graphs and on the vertices s and t .

7.1 Nine Problems

We first consider two restrictions on the global structure of a GGR problem, and two local restrictions:

- The problem GGR-B is the set of directed grid graphs G where s and t are vertices on the **boundary** of G , and there is a path from s to t in G . (Equivalently, G is an n -by- n grid, with s in position $(1, 1)$ and t in position (n, n) .)
- The problem LGGR is the set of **layered** directed grid graphs G , having **only east and south edges**, where there is a path from s to t . (Again, we use the convention that s is in position $(1, 1)$ and t is in position (n, n) .)
- The problem 1GGR is the set of directed grid graphs G of **outdegree at most 1** where there is a path from s to t . (Since a cycle of length two can not contribute to a path from s to t , and since the existence of such cycles makes certain of our reductions more complicated, and since such cycles are easy to eliminate via a syntactic test, we assume that there is no cycle of length 2 in an instance of 1GGR.)
- The problem 11GGR is the set of directed grid graphs G of **indegree and outdegree at most 1** where there is a path from s to t .

It is obvious that 11GGR is a special case of 1GGR and LGGR is a special case of GGR-B. The local and global restrictions are orthogonal, so that the three global conditions (general, boundary, and layered) and three local conditions (general,

outdegree 1, both degrees 1) give us nine special cases of the GGR problem: GGR, 1GGR, 11GGR, GGR-B, 1GGR-B, 11GGR-B, LGGR, 1LGGR, and 11LGGR. Even the easiest of these problems, 11LGGR, is non-trivial, as we will show in Section 8 that it is hard for the class TC^0 .

There are other natural ways to define a layered graph. We could forbid only one of the four directions of edges rather than two. Or we could allow diagonal edges but force them to go only northeast, east, or southeast, making each north-south column a layer according to the standard definition. But it is an easy exercise to construct a first-order projection from a graph satisfying any one of these restrictions to one satisfying any of the others. (We prove a very similar result in Proposition 26.)

7.2 Undirected GGR

One of the most natural local restrictions on a graph is **undirectedness**. Long before Reingold [Rei05] showed that the undirected reachability problem is in L, Blum and Kozen [BK78] showed that the UGGR problem, testing reachability in undirected grid graphs, is in L. Here we show that UGGR is equivalent to *four* of the nine versions of GGR we have just defined:

Theorem 9 *The problems UGGR, UGGR-B, 1GGR, 1GGR-B, 11GGR, and 11GGR-B are all equivalent under first-order projections.*

Proof: We will show that $1GGR \leq_{\text{proj}}^{\text{FO}} UGGR \leq_{\text{proj}}^{\text{FO}} UGGR-B \leq_{\text{proj}}^{\text{FO}} 11GGR-B \leq_{\text{proj}}^{\text{FO}} 1GGR$, appealing to Section 7.3 for the second reduction and observing that the last reduction is trivial.

Lemma 10 $1GGR \leq_{\text{proj}}^{\text{FO}} UGGR$

Proof: The well-known general reduction from outdegree one reachability to undirected reachability works without modification for grid graphs. Given an outdegree one grid graph G and vertices s and t , create an undirected graph H by modifying G to delete the edge (if any) out of t and change each directed arc to an undirected edge. Since the vertices with paths to t in G form a directed tree, the corresponding vertices in H are simply t 's connected component. So s has a directed path to t in G if and only if it has an undirected path to t in H . The reduction is clearly a first-order projection. \square

Lemma 11 $UGGR-B \leq_{\text{proj}}^{\text{FO}} 11GGR-B$

Proof: We merely have to formalize the familiar “right-hand rule” for exploring mazes – if we place our right hand on the wall and keep walking with our hand on the wall, we will return to our starting place having gone completely around the connected component of wall to our right. If both our starting place and our goal are on the boundary of the entire maze, they are on the boundary of their connected component.

More formally, given an undirected grid graph G and vertices s and t on its boundary, we define a grid graph H of indegree and outdegree at most 1 as follows. The vertices of H will be points $(a/3, b/3)$ where a and b are integers – when both coordinates are integers we identify this vertex of H with the corresponding vertex of G . (Note that the positive x direction is east, and the positive y direction is south.) The directed edges of H will have the property that there is an edge of G $1/3$ unit to their right in their direction of travel, unless they are turning a corner:

- If there is an edge in G between (u, v) and $(u + 1, v)$, then there are directed arcs in H from $(u + 1/3, v - 1/3)$ to $(u + 2/3, v - 1/3)$ and from $(u + 2/3, v + 1/3)$ to $(u + 1/3, v + 1/3)$.
- If there is an edge in G between (u, v) and $(u, v + 1)$, then there are directed arcs in H from $(u - 1/3, v + 2/3)$ to $(u - 1/3, v + 1/3)$ and from $(u + 1/3, v + 1/3)$ to $(u + 1/3, v + 2/3)$.
- If (u, v) is a vertex of G with *no* edge in G to $(u + 1, v)$, then H has edges from $(u + 1/3, v - 1/3)$ to $(u + 1/3, v)$ and from $(u + 1/3, v)$ to $(u + 1/3, v + 1/3)$.
- If (u, v) is a vertex of G with *no* edge in G to $(u - 1, v)$, then H has edges from $(u - 1/3, v + 1/3)$ to $(u - 1/3, v)$ and from $(u - 1/3, v)$ to $(u - 1/3, v - 1/3)$.
- If (u, v) is a vertex of G with *no* edge in G to $(u, v + 1)$, then H has edges from $(u + 1/3, v + 1/3)$ to $(u, v + 1/3)$ and from $(u, v + 1/3)$ to $(u - 1/3, v + 1/3)$.

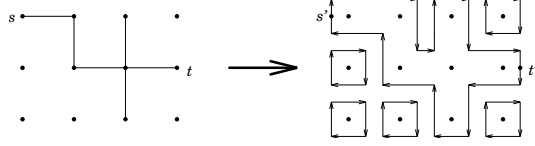


Figure 5. An undirected grid graph and its in-1-out-1 graph.

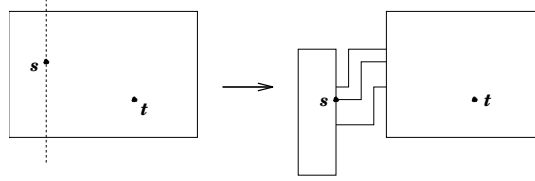


Figure 6. Putting s and t on the same row.

- If (u, v) is a vertex of G with no edge in G to $(u, v - 1)$, then H has edges from $(u - 1/3, v - 1/3)$ to $(u, v - 1/3)$ and from $(u, v - 1/3)$ to $(u + 1/3, v - 1/3)$.

We define vertices s' and t' in H by moving $1/3$ unit away from the rest of G from s and t respectively. It is clear that H has both indegree and outdegree at most one, and that there is a directed path from s' to t' in H if and only if there is an undirected path from s to t in H . Figure 5 shows the result of this construction on a small undirected graph. \square

Thus all these versions of the problem are equivalent under first-order projections. \square

7.3 The Boundary Construction

In this section we show that each of the problems GGR, UGGR, and IGGR reduces via first-order projections to the special case where s and t are on the external boundary. For simplicity, we first consider GGR. (The reader will note some similarity with the proof presented in Section 3.)

Theorem 12 $GGR \stackrel{FO}{\leq}_{proj} GGR-B$.

Proof: Let G_0 be a grid graph. We modify G_0 to obtain a new graph G , in which s and t appear on the same horizontal row of G ; call this row m ; this is accomplished by adding some paths to effect a vertical shift of part of the grid, as illustrated in Figure 6. We may assume without loss of generality that there is no vertical edge out of s or into t , and may also assume that s is a source and t is a sink; note also that s appears to the left of t in the grid. Modify G by inserting a new row of “dummy” vertices just above row m of G , to obtain a new graph G' . In G' there are no horizontal edges in row $m + 1$, and all edges that enter row $m + 1$ vertically from above continue on below, and vice-versa.

Now build a new graph H by cutting G' horizontally along row $m + 1$ to obtain two grids G'_{top} and G'_{bottom} . There is a copy of row $m + 1$ in each of G'_{top} and G'_{bottom} . In H , the graph G'_{bottom} appears above G'_{top} . For each vertex v in row m_1 to the left of s or to the right of t , there is a path connecting the two copies of v , going around the closest side boundary, and directed the same way as the edge that passes through v in G , as illustrated in Figure 7. Also as illustrated in Figure 7,

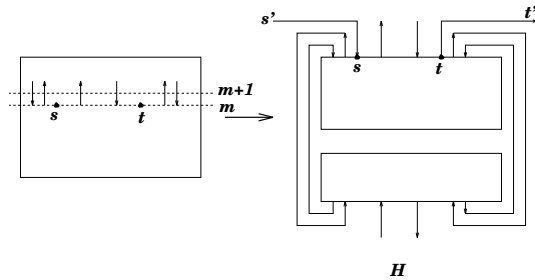


Figure 7. The basic gadget H

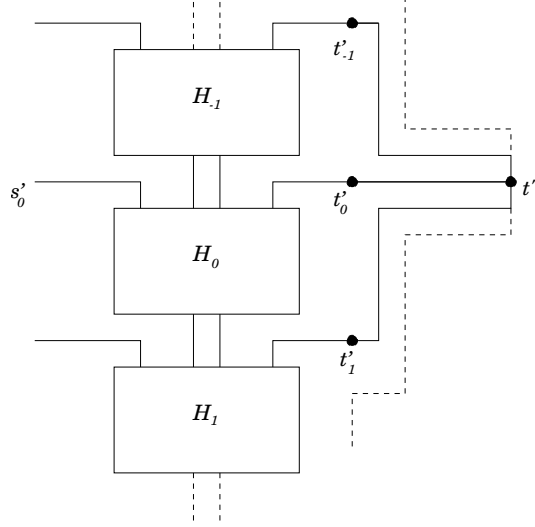


Figure 8. Connecting multiple copies of H

add new vertices s' and t' at the top right and left corners, respectively, connected via paths to s and t . For the vertices in row $m + 1$ that appear between s and t , add vertical paths that we will use to connect different copies of H together.

Let there be n vertices in G . Create $2n + 1$ copies of H , labeled $H_{-n}, H_{-n-1}, \dots, H_{-1}, H_0, H_1, \dots, H_n$, and connected vertically with H_0 in the middle, where the connections are made at the vertical paths between the copies of s and t in the bottom row of H_{i-1} and the corresponding paths in the top row of H_i . (See Figure 8.) A simple inductive argument shows that there is a path from s to t in G_0 iff there is a path from s'_0 to one of the vertices t'_i . The vertex s'_0 is on the external face, as is each of the vertices t'_i . The construction is completed by creating a new vertex t'' and adding paths from each t'_i to t'' . Call the resulting grid graph H' . It is easy to see that this reduction can be accomplished by means of a first-order projection, *i.e.*, the presence or absence of each edge in the resulting graph depends on the presence or absence of a single edge in the input graph. \square

Corollary 13 $\text{UGGR} \leq_{\text{proj}}^{\text{FO}} \text{UGGR-B}$ and $\text{1GGR} \leq_{\text{proj}}^{\text{FO}} \text{1GGR-B}$

Proof: If G_0 has outdegree one, then the graph H' also has outdegree one. If G_0 is undirected, then the graph H' will also be undirected, if we modify the construction by adding *undirected* paths from s' to s and from t to t' , as well as from each t'_i to t'' . \square

We conclude this section with the observation that a much simpler construction is sufficient if we wish to put *one* specified vertex on the boundary, instead of two.

Proposition 14 *For any given grid graph G and vertex v , there is a graph H that can be expressed as a first-order projection of G , that has the same connectivity properties as G , but has vertex v on the boundary of H .*

Proof: The three figures Figure 9, Figure 10 and Figure 11 illustrate how any given cell E of the grid graph G (containing vertex v) can be “stretched” to become the boundary of H (with the subgrids that surrounded E being flipped over into the interior of E). \square

7.4 Five Problems

The results of the preceding section and of Section 7.3 reduce our nine problems to five. If we close each under first-order reductions, we get a hierarchy of complexity classes within NL and (as we shall see in Section 8) above TC^0 . Since each problem has a number of interesting alternate formulations, we spend some time looking at each in turn:

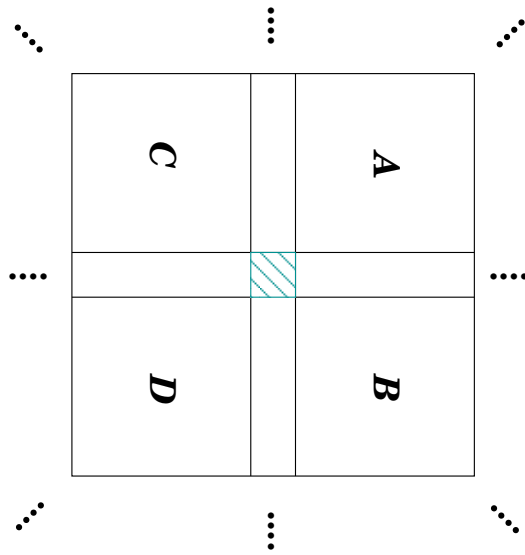


Figure 9. Grid graph G with cell E in the center.

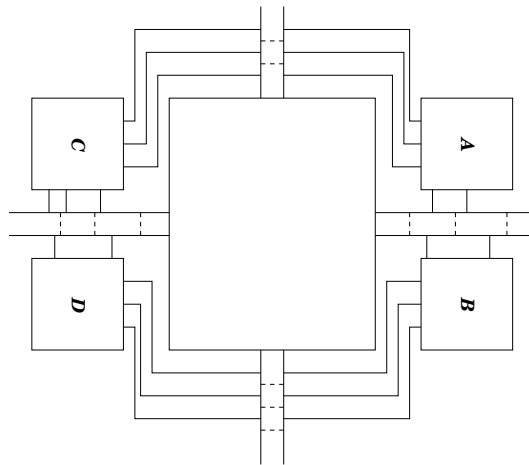


Figure 10. Cell E "stretched" to full size.

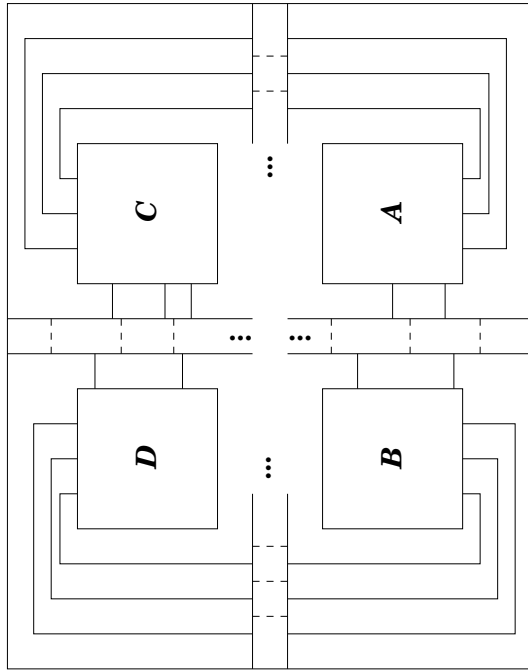


Figure 11. The final graph H .

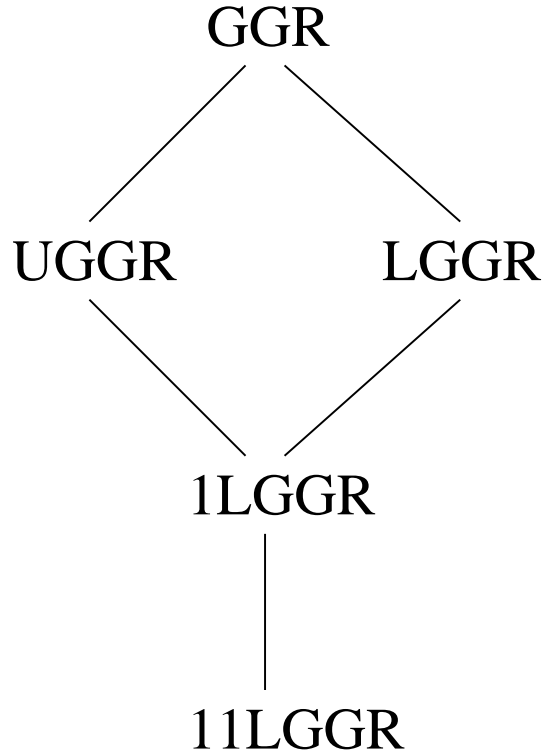


Figure 12. The five surviving GGR problems.

7.4.1 GGR

We have already presented most of our theorems regarding the general GGR problem (namely, that GGR is equivalent to reachability in planar graphs under logspace reductions). We showed in Section 7.3 that GGR and GGR-B are equivalent under first-order reductions. We have also already mentioned that $\text{GGR} \in \text{UL}$ [BTV07].

It is worth spending a paragraph discussing whether GGR is complete for NL. Since many would conjecture that $\text{NL} = \text{UL}$, the fact that $\text{GGR} \in \text{UL}$ is not strong evidence that GGR is not complete for NL, although it certainly qualifies as circumstantial evidence. Additional circumstantial evidence for its not being complete for NL comes from the following observations:

- GGR is not even known to be hard for L under first-order reductions; for all other known examples of complete problems for NL hardness for L is essentially trivial.
- The proof of the Immerman-Szelepcsényi theorem [Imm88, Sze88] showing that NL is closed under complement bears little relation to the simple argument showing that GGR reduces to its complement. (Namely there is *no* path from s to t in a grid graph G iff there *is* a path, from some boundary vertex on one path from s to t to a boundary vertex on the other path, in the *complement-dual* grid graph. For details see [BLMS98].)

It has been noticed by Jakoby and Tantau [JT06] that GGR is equivalent under first-order projections to a restriction of GGR that they call “tournament grid graphs”. Such graphs are obtained from a complete undirected $n \times n$ grid by assigning a direction to each edge.

We close this section with an observation about closure under first-order reductions: ζ

Proposition 15 *A language is in the class $\text{FO} + \text{GGR}$ iff it projection-reduces to GGR.*

The proof is identical to the proof of Proposition 16 in the next section.

7.4.2 UGGR

We found above that UGGR, undirected grid graph reachability, has a number of equivalent formulations including its boundary version UGGR-B. To these we may add the problem of determining the winner in a completed game of HEX [Bus06], because a hexagonal grid can easily be mapped by a projection reduction to the Euclidean grids we have defined here. Like GGR-B, UGGR-B projection-reduces directly to its complement by taking a complement-dual graph. This gives it another robustness property:

Proposition 16 *A language is in the class $\text{FO} + \text{UGGR}$ iff it projection-reduces to UGGR.*

Proof: We show that the set of languages that projection-reduce to UGGR-B, and hence (by Section 7.3) to UGGR, is closed under $\leq_T^{\text{AC}^0}$ reductions. We give an inductive argument on the depth of the circuits computing the $\leq_T^{\text{AC}^0}$ reduction (where without loss of generality the circuits for different lengths have the same structure, and all gates on the same level are of the same type). The inductive hypothesis is that the value of each wire w leading into a top-level gate can be represented as the answer to the question of whether or not a graph G_w is in UGGR-B where G_w is a projection of the input graph G . This is clearly true if the only gates are NOT gates, which establishes the basis for the induction. If the top-level gate is an AND gate, then it suffices to connect the graphs G_w in series. Similarly, if the top-level gate is an OR gate, then it suffices to connect the graphs G_w in parallel. If the top level gate is a NOT gate, then as we observed above, the complement-dual graph lets us represent the negation of a UGGR-B problem as the OR of polynomially many UGGR-B problems (and thus again we can connect these graphs in parallel.) If the top level gate is an oracle gate g , then we can replace each wire w (representing an edge (x, y) in the encoding of the grid graph H presented as input to g) by a small sub-grid encoding the graph G_w , identifying the source vertex as x and the sink vertex as y . The details are straightforward to fill in; by simple padding we may assume that all of the graphs G_w are the same size. \square

In its incarnation as 11GGR, UGGR can be seen to have the following *counting* property:

Proposition 17 *If G is a directed grid graph of indegree and outdegree each at most one, then the following predicate projection-reduces to UGGR: $\text{DIST}(s, t, k) \leftrightarrow$ the path out of s reaches t in exactly k steps.*

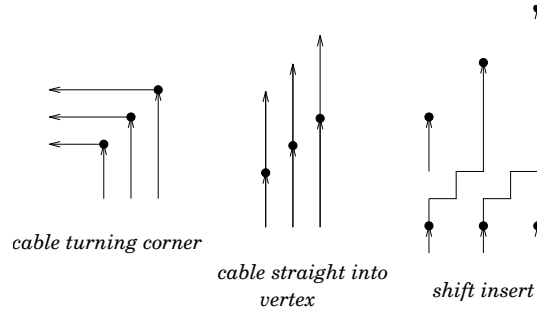


Figure 13. The Construction of Proposition 17

Proof: We first note that we can determine many properties of the *directed* path out of s in G by using FO + UGGR to answer questions about related *undirected graphs*. By looking at the undirected graph obtained by erasing the arrows in G , we can tell whether the path passes through a given vertex v . By removing an edge from this undirected graph and retesting, we can determine whether a given directed edge occurs on the path in G . Similarly, we can tell whether the path turns a corner at a given vertex.

If the path does turn a corner at a vertex v , then either the path cuts across the diagonal line at v running from northeast to southwest, or else it cuts across the diagonal line at v running from northwest to southeast. Let us call v an “NE” vertex in the first case, and an “NW” vertex in the second case.

Now we can define a graph $H_{s,t,k}$ where each vertex along the path is replaced by $k + 1$ copies. These copies are placed in a diagonal line from northeast to southwest if v is an “NE” vertex or if the path does not turn a corner at v , and the copies are placed in a diagonal line from northwest to southeast otherwise. Each edge along this path is replaced by a “cable” of $k + 1$ parallel straight paths. The copies of each vertex and edge are numbered so that copy k is to the left and copy 0 is to the right in the direction of the path’s travel.

Finally, on each incoming cable, we insert a shift component so that the path forming the i ’th copy of each edge now connects the i ’th copy of its source to the $i + 1$ ’st copy of its destination. (See Figure 13.) Note that this graph H also has indegree and outdegree at most 1. Then $\text{DIST}(s, t, k)$ is true iff there is a path in H from copy 0 of s to copy k of t . We can define H in FO + UGGR, and thus by Proposition 16 we can define H as a first-order projection of G .

□

In Section 9 we will be interested in the depth-first search of a directed tree embedded in a grid graph. If we convert the directed tree to an undirected tree and then to a graph of indegree and outdegree one by the constructions of this section, we produce a tour of the vertices of the tree that exactly follows the order in which they are visited by the depth-first search. Because we can count the length of paths in this final graph, we conclude:

Theorem 18 *Let T be a directed tree embedded in a grid graph and consider the depth-first search of T that visits children of a node in the left-to-right order given by the embedding. Then the following properties of the search are each computable in FO + UGGR: start time of a vertex, finish time of a vertex, depth of a vertex, and whether one vertex is an ancestor of another.* □

7.4.3 LGGR

We begin with a simple proposition::

Proposition 19 *Reachability in layered planar digraphs is logspace-reducible to LGGR.*

Note: When we refer to “layered planar digraphs” we mean layered graphs that are presented with vertices already partitioned into layers, with a vertex labeled as the “bottom” of each layer, and a planar embedding of the edges into the vertical space between each layer of vertices.

Proof: Given a layered planar graph G (with vertices already partitioned into layers as discussed above), first check to see whether there are any vertices with indegree or outdegree greater than 2. If so, then add some dummy layers, replacing nodes with large fan-in or fan-out with small trees of fan-in or fan-out at most 2. This is easy to accomplish in logspace.

Since each layer has a vertex labeled as the “bottom” vertex in the layer, we can easily compute an ordering on the vertices in each layer that is consistent with the planar embedding; if we know that v is the i th vertex in the layer, then the vertex at position $i + 1$ can be found by walking around the appropriate face that is adjacent to vertex v . (Any vertex that is omitted in this way must be unreachable from the start vertex.) This yields a natural assignment of vertices to a coarse grid (since each vertex occupies a known layer and the vertices have a known ordering within the layer). Now it is a simple matter to embed edges along a fine grid, as in the proof of Theorem 2. In this embedding, none of the edges point to the west. As observed at the end of Section 7.1, instances of GGR where no edges point west can easily be reduced to LGGR. \square

We now present our main theorem that deals with layered grid graphs.

Theorem 20 LGGR \in UL.

Proof: Let G be a layered $n \times n$ grid graph, with vertex s in column 1 and vertex t in column n . We define a weight function w on the edges of G as follows. If e is directed vertically (that is, from (i, j) to $(i + 1, j)$), then e has weight zero. Otherwise, e is directed horizontally and is of the form $(i, j) \rightarrow (i, j + 1)$. In this case, the weight of e is i . This weight function induces a natural weight function on paths; the weight of a path is the sum of the weights of its edges. (It is a helpful approximation to think of the weight of a path as the number of boxes of the grid that lie above the path.)

The minimal-weight simple path from s to any vertex v is unique. This is because if there are two paths P_1 and P_2 from s to v that have the same weight, there must be some column in which P_1 is higher than P_2 and another column in which P_2 is higher than P_1 . Since G is a layered grid graph, this means that there is some point in between these two columns in which the two paths intersect. The path from s to v that follows the two paths until they diverge, and then follows the path closer to the top of the grid until they next intersect, and continues in this way until v is reached, will have smaller weight than either P_1 or P_2 , and thus they cannot have had minimal weight.

At this point, we are able to mimic the argument of [RA00].

Let C_k be the set of all vertices in column k that are reachable from s . Let $c_k = |C_k|$. Let Σ_k be the sum, over all $v \in C_k$ of the minimal weight path from s to v . Exactly as in [RA00], there is a UL algorithm that, given (G, k, c_k, Σ_k, v) , can determine whether there is a path from s to v or not. (We emphasize the words “or not”; if there is no path, the UL machine will determine this fact; the algorithm presented in [RA00] has this property.) Furthermore, this algorithm has the property that, if v is reachable from s , then the UL machine can compute the weight of the minimal-weight path from s to v . (Informally, the machine tries each vertex x in column k in turn, keeping a running tally of the number of vertices that have been found to be reachable, and the total weight of the guessed paths. For each vertex x , the machine guesses whether there is a path from s to x ; if it guesses there is a path, then it tries to guess the path, and increments its running totals. If $x = v$, then it remembers the weight of the path that was guessed. At the end, if the running totals do not equal c_k and Σ_k , then the machine concludes that it did not make good guesses and aborts. By the properties of the weight function, there will be exactly one path that makes the correct guesses and does not abort.)

It suffices now to show that a UL machine can compute the values c_k and Σ_k . Observe first of all that c_1 is easy to compute (by simply walking down column 1 from s and counting how many vertices are reachable), and $\Sigma_1 = 0$.

Assuming that the values c_k and Σ_k are available, the numbers c_{k+1} and Σ_{k+1} can be computed as follows. Initialize c_{k+1} and Σ_{k+1} to zero. For each vertex v in column $k + 1$, for each edge of the form $x \rightarrow y$ to a vertex y in column $k + 1$ such that there is a path in column $k + 1$ from y to v , if $x \in C_k$ via a minimal-weight path of weight w_x , then compute the weight w'_x of the path to v through x . Let w_v be minimum of all such values w_x . Increment c_{k+1} by one (to indicate that v is reachable) and increase Σ_{k+1} by w_v . (This algorithm is actually more general than necessary; it is easy to show that the minimal-weight path to v will always be given by the “topmost” vertex $x \in C_k$ for which there is an edge $x \rightarrow y$ to a vertex y that can reach v in column $k + 1$.)

This completes the proof. \square

We observe that we have shown that a UL algorithm can also determine whether there is *not* a path from s to t , and thus LGGR is in UL \cap coUL.

One interesting question regarding LGGR is whether it is any easier than general GGR. It seems plausible that searching for a path that must always make progress in a given direction would be easier than searching for one that could double back upon itself arbitrarily. But the evidence we have for this is rather thin. Now that PLANAR.STCONN has also been shown to lie in UL \cap coUL ([BT07]) we have no upper bounds for the layered case that are not also known to hold for the general case.

Another interesting question is the relationship, if any, between LGGR and reachability for general grid graphs that happen to be acyclic. The two restrictions seem similar, but nothing is known.

It is not clear whether LGGR reduces to its complement. The complement-dual of a grid graph whose edges go only east and south is a grid graph that contains *all possible* north and east edges, and some edges going south and west. There may be a way to reduce this problem to LGGR, but we don't know of one.

LGGR is also a special case of evaluating a **layered monotone planar circuit**, where the circuit has only OR gates and constant 0 gates (except for one constant 1 gate). Limaye *et al.* [LMS06] give a nice survey of the various versions of this problem along with some new results.

7.4.4 1LGGR

The 1LGGR problem has some alternate characterizations, which we find useful in proving our results about this problem.

Definition 21 *An outdegree exactly-one layered grid graph is an instance of 1LGGR where every vertex not appearing on the boundary has outdegree 1. That is, the only sinks are on the boundary. The reachability problem on these graphs is denoted by E1LGGR.*

Lemma 22 *E1LGGR is equivalent (via projections) to the reachability problem on directed grid graphs that have some east edges, all possible south edges, and no north or west edges.*

Proof: We first reduce this new problem to E1LGGR. Let G be a layered grid graph with some east and all south edges. Without loss of generality let s be the northwest corner and t the southeast corner. Define the following instance H of E1LGGR. The vertices of H are the same as those of G . If vertex v has an east edge out of it in G , it has an east edge out of it in H . Otherwise it has a south edge out of it in H . Clearly, every vertex of H that is not on the south boundary has outdegree one. It is easy to show by induction that the path out of s in H reaches or passes directly north of every vertex reachable in G . Either this path ends at a vertex on the south boundary that has no east edge, or it reaches the east boundary and thus goes south to t . So the path in G exists iff the path in H does.

For the other reduction, let G be an instance of E1LGGR. Define H to be a copy of G with all possible south edges added. Define G^T to be the layered grid graph obtained from G by reflecting about the northwest-to-southeast diagonal, and let H' be a copy of G^T with all possible south edges added. Finally, let I be a *series connection* of H and H' – a layered grid graph, with all south edges present, obtained by placing H in the northwest quarter and H' in the southeast quarter of a single graph, identifying the southeast corner of H with the northwest corner of H' . It is easy now to verify that there is a path from the northwest corner of I to the southeast corner iff the unique path from s in G reaches t , rather than some other sink on the boundary of G . \square

Proposition 23 *The language of problems projection-reducible to E1LGGR is closed under complement.*

Proof: The complement-dual of a layered grid graph with some east edges and all south edges has all possible north and east edges, some south edges, and no west edges. But the north edges are of no additional use in making a path from north to south, so this is equivalent to a problem with some south and all east edges, clearly isomorphic to the problem with all south and some east. \square

Theorem 24 *1LGGR and E1LGGR are equivalent under projections (and thus, by the preceding proposition, 1LGGR projection-reduces to its complement).*

Proof: Since E1LGGR is a special case of 1LGGR, it suffices to reduce 1LGGR to E1LGGR. First, we present a first-order reduction. Let G be an instance of 1LGGR. Let H be a graph with the same set of vertices and containing all of the edges of G , but with the property that if v is an internal sink in G , then v has an edge leading out to the east in H . H is clearly an instance of E1LGGR, and there is path from s to t in G if and only if (there is a path from s to t in H and, for every sink v of G , there is not a path from s to v in H).

It remains to simulate this reduction with a projection. Note that H can be formed as a projection from G ; although the condition that v is a sink depends on *two* bits of G , we can phrase this condition equivalently by saying that there is an east edge out of v iff there is not a south edge out of v . Next note that the first-order reduction is the AND of a reachability question on H with polynomially-many conditions of the form C_v : “ v is not a sink or there is not a path from s to v in H ”. C_v is equivalent to the negation of the condition “ v is a sink and there is a path from s to v in H ”, which can be expressed by a reachability question in a graph with two components: the first component is a two-by-two grid graph containing the negations of the two edges out of v , and the second component is the subgraph of H with v as terminal node. It is easy to

see that the negation of C_v can thus be expressed as a projection of E1LGGR, and thus by the preceding proposition, each condition C_v can be posed as a positive query to E1LGGR.

All of the polynomially-many reachability conditions of our first-order reduction can be combined in series to form a single instance of E1LGGR. (That is: form a grid with the queried graphs along the main diagonal, with vertex s in one graph identified with vertex t in the next. Vertices along the boundaries of the queried graphs are connected to paths running east or south to the boundary of the large graph, to maintain the property that the only sinks are on the boundary.) This yields the desired projection. \square

Theorem 25 *Any language first-order reducible to 1LGGR is projection-reducible to it.*

Proof: We follow essentially the same strategy as in the proof of Proposition 16 – but we cannot use the same construction of simulating an OR gate by a parallel connection, since that construction does not have outdegree 1. However, using DeMorgan’s laws, we can assume that a first-order reduction to 1LGGR is computed by a constant-depth circuit with only AND and NOT gates, in addition to oracle gates for 1LGGR. The inductive argument now proceeds in exactly the same way as in the proof of Proposition 16, but we need to be more careful in the way that oracle gates are simulated. Let G be the n -by- m grid corresponding to the input wires of an oracle gate, where by induction we are assuming that we have instances of 1LGGR G_w for each of these wires. For each possible horizontal edge (u, v) of G represented by wire w , we can place G_w diagonally between u and v , so that all edges of G_w are running northeast or southeast. For each vertical edge (u, v) represented by wire w , we place G_w diagonally between u and v so that all edges of G_w are running southwest or southeast. If we rotate this graph 45 degrees counterclockwise, we obtain a grid graph with outdegree one having no west edges, such that there is a path from s to t if and only if there is a path from s to t in G . The proof is completed by showing that reachability in graphs of this type is projection-reducible to 1LGGR; see Proposition 26. \square

Proposition 26 *The restriction of 1GGR to instances having no west edges projection-reduces to 1LGGR.*

Proof: Consider a directed n by n grid graph G with no west edges, a vertex s on the west boundary, and a vertex t on the right boundary. We describe how to successively recast this GGR instance as a sequence of GGR-like instances, the last of which is a 1LGGR instance.

- Our first graph G' is n by $n(n+1)$ and has edges that go northeast, east, and southeast. We embed the vertices of G in G' so that there are n columns of new vertices between each column of G vertices. For each east edge in G , we make a corresponding path of $n+1$ east edges in G' . For each north or south edge in G , we put northeast or southeast edges respectively on the corresponding vertex in G' and each of the next $n-1$ new vertices in the same row. Note that G' also has outdegree one. We can now see that if the path in G from vertex u first reaches a particular column at vertex v , then the path out of u in G' also goes to v .
- We now make G'' by doubling the size of G' and replacing each east edge with a path of length two consisting of a northeast and a southeast edge. Northeast and southeast edges in G' become paths of two northeast or two southeast edges in G'' .
- Finally, we make a 1LGGR instance H by rotating G'' 45 degrees clockwise so that its edges go east and south.

\square
As we will see in Section 8, the complexity class of problems first-order reducible to 1LGGR lies somewhere between L and NC^1 . These two classes exemplify one contrast between sequential computation (L) and parallel computation (NC^1). The question of whether $L = NC^1$ is the question of whether sequential computations using only log space can be parallelized to a certain extent. (Of course L problems can be solved in $O(\log^2)$ parallel bit operations because $L \subseteq NC^2$, but the question is whether we can get depth $O(\log n)$.)

Here is a problem that looks to be inherently somewhat sequential, in that a polynomial number of operations *appear* to be necessary in sequence. Let A be an n by n Boolean array and consider the following Java code fragment:

```
int count = 0;
for (int i=0; i < n; i++)
    if (A[i, count]) count++;
```

Determining whether the value of `count` at the end of this fragment is some value k is easily projection-reduced to 1LGGR. If $1LGGR \in NC^1$, then this code can be parallelized in some way that is not readily apparent to get $O(\log n)$ time instead of the $O(\log^2 n)$ time from pointer doubling.

7.4.5 11LGGR

The easiest problem in our hierarchy, 11LGGR, has an interesting alternate formulation. Consider a data structure holding a varying number of items and supporting the following two operations:

- $insert(i)$ places a new element in position i and moves all higher-numbered elements up by one position, and
- $delete(i)$ removes the element in position i and moves all higher-numbered elements down one position.

Given such a structure A , a sequence s of inserts and deletes, and a position i in A , define the predicate $Preserves(A, s, i, j)$ to be true iff the item in position i at the beginning still exists and is in position j after s is executed.

This problem is reducible to 11LGGR, because we can make a grid where each row represents a time step, each column represents a position, each vertex represents an item at some time, edges go southwest, south, or southeast to represent the movement or non-movement of an item, and a path thus traces the history of a given item.

What is interesting is that this $Preserves$ problem is *complete* for the class of problems first-order reducible to 11LGGR. Given an arbitrary 11LGGR instance, we can interpret a layer as a time step in the history of a similar but more complicated data structure, where several vertices might be inserted or deleted at the same time, corresponding to the sources or sinks among that layer of vertices. But these operations may be sequentialized into single inserts and deletes as above. If we do this for each layer G , we get a $Preserves$ problem equivalent to the 11LGGR instance.

7.5 Acyclicity and Single-Source

We have no logspace algorithm to test whether a given directed grid graph is **acyclic**, because this problem is hard for LGGR (which is not known to lie in L). But in Section 9 we will present logspace algorithms for two special cases of general acyclic GGR.

These are the **single-source** problem SMGGR and the **single source, single-sink** problem SSGGR. (In each case we will assume, as per Proposition 14, that the source occurs on the boundary of the grid graph.)

Even the latter problem is non-trivial in our hierarchy:

Lemma 27 $11LGGR \leq_{proj}^{FO} SSGGR$

Proof: Appealing to Lemma 22, let G be a layered grid graph with some east edges and all possible south edges, with northwest corner $(0, 0)$ and southeast corner (a, b) . We form a graph H by adding one new row each north and south of G and one new column each east and west of it. H will include all possible south edges, and its east edges will be those of G plus all those in the two new rows. These changes do not affect reachability between vertices of G , but in H $(-1, -1)$ is the only source and $(a + 1, b + 1)$ is the only sink. \square

Since most of our arguments in Section 9 apply to any graphs embedded in the plane, we will present them in general form and note where the L constructions may be carried out in $FO + UGGR$ in the case of grid graphs.

8 Lower Bounds

8.1 A TC^0 Lower Bound For 11LGGR

Even the easiest version of GGR we have considered has nontrivial complexity:

Theorem 28 *The problem 11LGGR is hard for TC^0 under first-order reductions.*

Proof: Our reduction is from the complete problem EXACTLY-HALF, the set of binary strings with exactly the same number of zeroes and ones. Given a string $w = w_0 \dots w_{n-1}$ of length n , with n even, we construct a grid graph G that is an $n/2 + 1$ by $n/2 + 1$ square with vertices numbered $(0, 0)$ through $(n/2, n/2)$. The edge out of vertex (i, j) is to the east (to $(i + 1, j)$) if $w_{i+j} = 0$ and south (to $(i, j + 1)$) if $w_{i+j} = 1$. Thus all of the vertices in each diagonal have edges all in the same direction (where the vertices in diagonal k are the vertices $v_{i,j}$ such that $i + j = k$). On the east and south boundary, a vertex is a sink if its edge, by this rule, would leave the graph.

It is clear that this graph is layered and has both maximum indegree and outdegree of 1, and thus is an instance of 11LGGR once we set $s = (0, 0)$ and $t = (n/2, n/2)$. Equally clearly, the unique path out of s will take one edge east for every zero in

w and one edge south for every one, until or unless it reaches the east or south boundary of G . It reaches t if and only if the input string is in the language EXACTLY-HALF. The reduction is a simple first-order projection. \square

We can define a special case of 1LGGR that is *complete* for TC^0 . Suppose that the indegree and outdegree of every vertex is *exactly* one, except for vertices on the boundary. This condition forces all the edges from vertices on a given $i + j = k$ diagonal to go in the same direction. Thus it must be exactly the encoding of some string under our reduction from EXACTLY-HALF to 1LGGR. Given two vertices $s = (i, j)$ and $t = (i', j')$, we need only find the substring $w_{i+j} \dots w_{i'+j'-1}$ of this string, and determine whether the number of zeroes in this string is exactly $i' - i$. This is clearly easy to do by reduction to EXACTLY-HALF and is thus in the class TC^0 . Since our earlier reduction always produces 1LGGR problems falling within the special case, the special case is complete for TC^0 .

8.2 An NC^1 Lower Bound: Series-Parallel Graphs

We now show that except for the minimal problem 1LGGR, each of our versions of GGR is hard for the class NC^1 . Our proof constructs a graph with a *particular* series-parallel decomposition. (By contrast, Jakoby *et al.* [JLR06] deal with graphs that *admit* such a decomposition.) While the GGR problem for such pre-decomposed graphs is in NC^1 , we have no NC^1 upper bound for any of the versions of GGR we have defined above.

Theorem 29 *The problem 1LGGR is hard for the class NC^1 under first-order projections.*

Proof: Our reduction is from a special case of the Boolean sentence value problem, proved to be both in NC^1 and hard for NC^1 by Buss, Cook, Gupta, and Ramachandran in [BCGR92]. A Boolean sentence is an infix Boolean formula with constants 0 and 1 and binary operators \wedge , \vee , and \neg , and BSVP is the set of such formulas that evaluate to 1. In Theorem 5.1 of [BCGR92], they construct a Boolean sentence whose value is equivalent to that of an arbitrary $O(\log n)$ time alternating Turing machine on a given input string of length n . Here we will use the fact that the sentence they construct is always:

- monotone (has no \neg operators),
- fully balanced (every constant occurs at the same depth), and
- alternating (\wedge and \vee operators alternate).

We describe a general inductive construction that takes a monotone Boolean sentence ϕ and produces a square grid graph G_ϕ that contains all possible south edges, some east edges, and no north or west edges, such that there is a path from the northwest to the southeast corner of G_ϕ if and only if ϕ is true. Figure 14 illustrates the construction.

As we observed in Section 7.2, 1LGGR can be defined in terms of reachability from the northwest to the southeast corner of such graphs. In the special case of a monotone, fully balanced, and alternating formula, our construction can be simulated by a first-order projection. This will show that the 1LGGR problem is hard for NC^1 under such projections.

We map constants to 2 by 2 graphs, with no east edges for a constant 0 and an east edge on the south boundary for a constant 1. Clearly a path from northwest to southeast exists for G_1 and not for G_0 .

If ϕ is the formula $\alpha \wedge \beta$, and α and β are already represented by square graphs G_α and G_β with sides of length a and b respectively, then G_ϕ is a square graph with side length $a + b$ with G_α in its northwest corner and G_β in its southeast corridor. The rest of G_ϕ has only the required south edges, except for a single east edge from $(a - 1, a)$ to (a, a) , the northwest corner of the copy of G_β . If there are paths from the northwest to southeast corners of G_α and G_β respectively, there is a path from the northwest corner $(0, 0)$ of G_ϕ to $(a - 1, a - 1)$, south one step, across the east edge to (a, a) , and across G_β to $(a + b - 1, a + b - 1)$. But the only way from column $a - 1$ to column a is across this east edge, and thus the only way to get from $(0, 0)$ to $(a + b - 1, a + b - 1)$ is to cross both G_α and G_β from northwest to southeast corner. The path across G_ϕ thus exists if and only if both α and β are true, that is, if ϕ is true.

Similarly, suppose that $\phi = \alpha \vee \beta$ and α and β are already represented as above. We make a square graph of G_ϕ of side $a + b$ as before, placing G_α and G_β as before. This time, our added east edges form two paths, from $(a - 1, a - 1)$ to $(a + b - 1, a - 1)$ and from $(0, a)$ to (a, a) . We must show that a path exists from $(0, 0)$ to $(a + b - 1, a + b - 1)$ in G_ϕ iff a path exists *either* across G_α *or* G_β . If the path exists across G_α , we may take it and then go due east to column $a + b - 1$ and then south to our goal. If the path exists across G_β , we can go from $(0, 0)$ south to $(0, a)$, then east to (a, a) and across this path to our goal. Conversely, suppose there is a path from $(0, 0)$ to $(a + b - 1, a + b - 1)$. Since there are only two edges from column $a - 1$ to column a , the path must use one of them. If it uses the edge from $(a - 1, a - 1)$ to $(a, a - 1)$ it must have previously crossed G_α , and if it uses the edge from $(a - 1, a)$ to (a, a) it must then cross G_β .

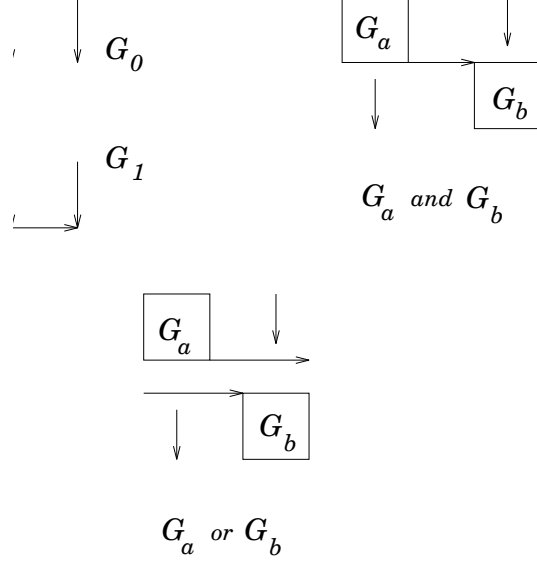


Figure 14. The construction of G_ϕ . All south edges are present.

If ϕ is a monotone, fully balanced, alternating Boolean sentence of depth d , this construction produces a square graph G_ϕ of side 2^{d+1} . To construct G_ϕ from ϕ , we need only place the east edges. For the i 'th of the 2^d constants in ϕ , we add an edge from $(2i + 1, 2i)$ to $(2i + 1, 2i + 1)$ iff this constant is 1. Without loss of generality, assume that the lowest-level operators in ϕ are \wedge 's. Then the east edges corresponding to \wedge operators go from $(i2^j - 1, i2^j)$ to $(i2^j, i2^j)$ whenever i and j are both odd. And the east paths corresponding to the \vee operators go from $(i2^j - 1, i2^j - 1)$ to $((i + 1)2^j - 1, i2^j - 1)$ and from $((i - 1)2^j, i2^j)$ to $(i2^j, i2^j)$ whenever i is odd and j is even. It should be clear that G_ϕ can be produced from such a ϕ by a first-order projection, since each edge of G_ϕ depends on at most one character of ϕ . \square

9 Acyclic Single-Source Graphs

In this section, we present our results for Single-Sink Single-Source planar DAGs (SSPDs) and Single-Source Multiple-Sink Planar DAGs (SMPDs).

Definition 30 An embedding of a planar DAG is said to be “Bimodal” if, for every vertex v , all incoming edges appear consecutively in the cyclic ordering around v . The embedding is said to have “SSPD faces” if each face (viewed as a subgraph) has a single source and a single sink.

Some properties of SSPDs and SMPDs are summarized below:

- Fact 31**
1. There is a path from the source to every vertex in every SMPD (and thus in every SSPD).
 2. There is a path from every vertex to the sink in every SSPD.
 3. Every embedding of an SSPD is Bimodal and has SSPD faces. (see [Yan91]).
 4. There is a logspace algorithm that, given any SMPD G , constructs a directed spanning tree T for G , rooted at the source. (The algorithm simply selects (arbitrarily) one incoming edge for each vertex; it is easy to see that this is a directed spanning tree.)
 5. Preorder and postorder numberings yielding the discovery time (**Discover**(x)) and finishing time (**Finish**(x)) for each vertex x , with respect to a depth-first search of the spanning tree G , can be computed by a L-transducer.

It is easy to see that forward edges in a depth-first search of T can be deleted without affecting the reachability predicate. (A non-tree edge (x, y) is a forward edge if y is a descendant of x in T .) Since it is easy to delete such edges in logspace

(and, in the case when G is a grid graph, this can also be done in $\text{FO} + \text{UGGR}$), we assume from now on that *there are no forward edges*. We classify edges with respect to the spanning tree obtained above as follows:

Definition 32 *Given an embedding of an SMPD and one of its spanning trees, all edges in the SMPD fall in one of the following classes:*

- *Tree Edges*
- *Local Edges: non-tree edges such that the unique undirected cycle formed by adding the edge to the tree does not enclose any vertex strictly within its boundary.*
- *Jump Edges: non-tree edges that are not local edges.*

We observe the following:

Observation 33 *If a subgraph of an SMPD does not contain any jump edges, then it has all its sinks on the external face.*

Proof: Any sink not on the external face must be contained strictly within some undirected cycle – but, by definition, any undirected cycle that contains no jump edges does not strictly contain any vertex. \square

Definition 34 *Given G and a spanning tree T as above, then for any vertex $x \neq s$ we define the left-most (right-most) path starting from x to be the path such that every edge (y, z) on the path is the last (resp. first) edge among all outgoing edges from y enumerated in the clockwise order, starting from the unique edge into x in T .*

9.1 Reachability in SSPDs

Theorem 35 *SSPD reachability is in L .*

Proof: We first state a lemma regarding the set of vertices reachable from a fixed vertex in a given SSPD.

Lemma 36 *Let R be the closed region bounded by the left-most and right-most paths from a vertex x to the sink t . The set of vertices in R is exactly the set of vertices reachable from x .*

This lemma tells us that, in order to determine whether there is a directed path from u to v , it suffices to consider the left-most and right-most paths from u to t and find whether either of them intersects an arbitrary path from s to v . More precisely, take the reverse of the left-most path from v to s in the SSPD formed by reversing all edges in the given SSPD, and call this path p . There is a path from u to v if and only if p intersects either the leftmost or the rightmost path from u to t . This yields a logspace algorithm and proves the theorem; it remains only to prove the lemma. \square

Proof: (of Lemma 36)

To see that each such vertex y is indeed reachable from x , we note that the subgraph in this region is itself an SSPD, and then appeal to Fact 31.

To see that no vertex other than those in region R is reachable from x , suppose to the contrary there is such a vertex y and a directed path P from x to y . Then since $x \in R$, let the path P exit the region R for the first time at vertex w , *i.e.*, let (w, z) be an edge in P such that $w \in R$ but $z \notin R$. But since the “left-most” outgoing edge from w is part of the boundary, it follows that all the other outgoing edges end in vertices lying either strictly within R or on its right boundary, contradicting the choice of w . \square

Corollary 37 *The problem SSGGR is in $\text{FO} + \text{UGGR}$.*

Proof: Let G be a single-source, single-sink grid graph, with the source on the boundary. We can easily construct the directed tree of Fact 31 as a first-order projection of G , and then by Theorem 18 we can compute all the predicates necessary to define the depth-first search of this tree in $\text{FO} + \text{UGGR}$. The argument of Theorem 35 refers only to reachability in graphs of outdegree one, which is computable in $\text{FO} + \text{UGGR}$ by Lemma 10. \square

9.2 Reachability in SMPDs

Theorem 38 SMPD reachability is in L.

Proof: We defer to later the question of how to recognize whether a given graph is an SMPD. Assume for now that we are given a DAG G that is an SMPD with source s , and we are trying to determine whether there is a path from u to v .

Construct a directed spanning tree T , as in Fact 31. It is easy to check in logspace whether u is an ancestor of v (in which case there is a path from u to v) or u is a descendant of v (in which case there is no path, since G is a DAG). So we assume that u is neither an ancestor nor a descendant of v . Hence, there is some first edge (p, q) on the tree path from s to u that is not on the tree path from s to v . Also note that no path from u to v can visit any descendant of v ; thus we can delete all proper descendants of v , so that v is a leaf. Embed G so that v is on the external face.

Visualize the planar graph with the source s at the center of a circle with tree paths from s to the leaves spread out approximately as radial lines to its circumference. Further, let the path with v as a leaf be embedded as a straight line below s , and with the edge (p, q) embedded so that q is directly above s , with the rest of the path to u embedded as a straight line above s . Pick some arbitrary path (say, the leftmost path) from u to a leaf and embed this path straight above u . This visualization is only for the sake of providing a vivid description of the algorithm; it will be straightforward that the necessary calculations can be done in logspace.

Note that the tree is arranged in a disk around s , with u , s , and v arranged along the vertical axis. The axis divides the graph into the “left” and “right” regions. Any vertex x in the left region divides it into the ancestors of x , the descendants of x , vertices between x and u , and vertices between x and v . A vertex in the right region also divides it in the same fashion.

For any two vertices x and y such that x is neither an ancestor nor a descendant of y , and both x and y lie on the same side of the axis, this notion also determines which of x and y is closer to u and which is closer to v . For instance, x is closer to u than y if the region between x and u is contained in the region between y and u .

Again in this view, consider a non-tree (local or jump) edge (x, y) , that does not cross the axis. If it is on the right (left) side of the axis, then it is said to be directed *toward* v if it is embedded clockwise (counterclockwise) around the disk, else it is said to be directed *away from* v . It is easy to see that this notion of direction can be determined for any edge (x, y) in logspace, by comparing the discovery and finishing times of x and y with that of u and v .

Definition 39 An edge is called useless if it is directed away from v .

Fact 40 If there is a path from u to v , then there is a path that uses no useless edges.

Proof: Consider a path p from u to v that uses the fewest number of useless edges among all paths from u to v . Suppose p contains some useless edge, and let (x, y) be the first useless edge on p . If a and b are two vertices appearing on p , then let $p(a, b)$ be the part of p that leads from a to b .

If $p(u, x)$ intersects the tree path from s to y at z , then we can clearly construct a path p' composed of $p(u, z)$, followed by the tree path from z to y , followed by $p(y, v)$. Thus p' is a path from u to v with fewer useless edges (since it avoids (x, y)), contradicting our hypothesis.

Otherwise, y is in the closed region bounded by the tree paths from s to x and from s to u , along with $p(u, x)$. Since v is embedded on the external face, v lies outside this region. Thus $p(y, v)$ must intersect the boundary of this region, which would create a directed cycle, contrary to the fact that G is a DAG. \square

In logspace we can detect and remove useless edges; we therefore assume that G has no useless edges. (Recall that we have already removed the forward edges from G .)

We need to define some basic search routines.

Definition 41 Given an SMPD G and a vertex x , let $\mathbf{ReachLocal}(x)$ be the set of vertices reachable from x using only tree edges and local edges.

Lemma 42 The predicate $y \in \mathbf{ReachLocal}(x)$ is in L.

Proof: Remove all of the jump edges from G , and call the resulting graph G' . Since there are no jump edges, all the sinks in G' lie on the external face (by appealing to Observation 33). Construct a new graph G'' by adding a new sink to G' along with an edge from each old sink to this new sink. Clearly G'' is an SSPD. Note that $y \in \mathbf{ReachLocal}(x)$ if and only if there is a path from x to y in G'' . The lemma now follows by Theorem 35. \square

As a consequence, we are able to make the following definition.

Definition 43 Given vertex $x \neq u$, define **FarthestReach**(x) to be the vertex in **ReachLocal**(x) that is on the same side of the axis as x and which is farthest from u (i.e. closest to v). Note that **FarthestReach**(x) can be found via a logspace computation, given x , as a consequence of Lemma 42.

For $x = u$, we can similarly enumerate **ReachLocal**(u) and find the farthest vertex from u in the enumeration, on the left and right side of the axis in \mathbb{L} . Let's call these vertices **LeftReach**(u) and **RightReach**(u) respectively. Ties are broken by choosing the vertex that is closer to the root s .

Observation 44 For any vertex $x \neq u$, there is no jump edge in the region enclosed by the following three paths:

- the tree path from s to x ,
- the tree path from s to **FarthestReach**(x), and
- any path p of tree and local edges from x to **FarthestReach**(x).

A similar statement holds for $x = u$, where we replace **FarthestReach**(x) by either **LeftReach**(u) or **RightReach**(u).

(The observation follows by noting that any such jump edge, by definition, properly encloses a vertex in the region enclosed by the undirected cycle it forms with the tree paths from s to its endpoints. Such a vertex would also be enclosed in the cycle formed by some local edge on p , in contradiction to the fact that no such vertex can exist for a local edge.)

Our basic strategy is as follows. We shall maintain an explored region, within which all vertices that are reachable through existing edges have been enumerated. The “explored region” is marked by two boundary vertices on the left and right sides of the axis, which we will store as **Limleft** and **Limright**, respectively.

Definition 45 All vertices that lie between **Limright** and u , and between **Limleft** and u , and the descendants of **Limright** and **Limleft**, constitute the explored region. Here is another way to visualize the explored region. Consider the tree paths p_ℓ and p_r that lead from s through **Limleft** and **Limright**, respectively, and then continue on to the descendants of **Limleft** and **Limright** that are closest to v . The paths p_ℓ and p_r partition G into three parts:

1. vertices in $p_\ell \cup p_r$,
2. vertices above $p_\ell \cup p_r$ (i.e., closer to u),
3. vertices below $p_\ell \cup p_r$ (i.e., closer to v).

The explored region consists of the vertices in the second block of this partition, together with **Limleft** and **Limright** and their descendants.

The procedure stops with a positive answer if v is enumerated, or stops with a negative answer if neither **Limleft** nor **Limright** progress in an iteration.

The algorithm starts with a local search from u . **Limleft** and **Limright** are updated to **LeftReach**(u) and **RightReach**(u). We then look for a jump edge (x, y) such that x lies in the explored region and y lies outside it, and such that y is closest to the explored region. A local search is performed from y to expand the explored region, and **Limleft** or **Limright** is updated, to mark the new border of the explored region.

```

SMPDReach( $u, v$ )
  Enumerate ReachLocal( $u$ )
  if  $v$  is enumerated
    then return true
  Limleft  $\leftarrow$  LeftReach( $u$ )
  Limright  $\leftarrow$  RightReach( $u$ )
   $z \leftarrow u$ 
  while true
    do
       $S \leftarrow \{(x, y) : (x, y) \text{ is a jump edge with}$ 
         $x \text{ in the explored region and}$ 
         $y \text{ in the unexplored region}\}$ 
      if  $S$  is not empty
        then pick  $(x, y) \in S$  such that
           $y$  is closest to  $u$  on either the left side
          or the right side of the axis.
          (i.e., as close as possible to Limleft or Limright),
          breaking ties by picking  $y$ 
          as close to the root  $s$  as possible
           $z \leftarrow y$ 
        else return false
      Enumerate ReachLocal( $z$ )
      if  $v$  is enumerated
        then return true
      if  $z$  is on the left side of the axis
        then Limleft  $\leftarrow$  FarthestReach( $z$ )
        else Limright  $\leftarrow$  FarthestReach( $z$ )

```

It is clear that the algorithm can be implemented in logspace.

Note that, since v is a leaf on the external face, and since all useless edges have been deleted, **Limleft** will always be to the left of v (unless it is equal to v), and **Limright** will always be to the right of v (unless it is equal to v). Thus, if v ever enters the explored region, v will be enumerated.

In order to argue that the algorithm is correct, we will establish the following invariant condition: Each time **Limright** or **Limleft** is updated,

- (1) all vertices that are enumerated are reachable from u ,
- (2) if there is any path from u to v , then there is a path from u to v that does not visit any vertex in the explored region that has not been enumerated,
- (3) all jump edges that go from the explored region to the unexplored region (i.e. edges in S) begin in an enumerated vertex, and
- (4) all jump edges that go from the unexplored region into the explored region end in an enumerated vertex.

The proof of the invariant relies strongly on the absence of forward edges and useless edges.

We must first establish that the invariant holds the first time that **Limright** and **Limleft** are updated. **Limright** and **Limleft** are first updated to **LeftReach**(u) and **RightReach**(u). By definition, all vertices enumerated by **ReachLocal**(u) are reachable from u ; this establishes (1). Suppose there exists a vertex z in the explored region that has not been enumerated, although there is a path from u through z to v . Let p be a path from u through z to v . Since any descendent of an enumerated vertex is enumerated, such a vertex z must lie properly inside the region R enclosed by the paths $p_\ell(s, \mathbf{Limleft})$, $p_r(s, \mathbf{Limright})$, and the paths from u to **Limleft** and **Limright**. (Recall the definitions of p_ℓ and p_r from Definition 45.) Consider the first edge (x, y) on this path where y lies inside the region R . (Clearly, x must lie on the boundary of R .) By Observation 44, the edge (x, y) cannot be a jump edge. If x lies on the path from u to **Limleft** or **Limright**, then (x, y) must be a tree edge or a local edge, which means that y would have been enumerated (contrary to hypothesis). Thus x must lie on $p_\ell(s, \mathbf{Limleft})$, or $p_r(s, \mathbf{Limright})$. Assume without loss of generality that it lies on p_ℓ . The path from z to v must exit region R somewhere (since v lies outside this region). If the path crosses the boundary of R at some point y' along the paths from u to **Limleft** or **Limright**, then there is a path from u to y' that follows only enumerated vertices, and hence (2) is satisfied. On the other

hand, if this path crosses the boundary of R at a point y' on p_ℓ , then there is a path from x to y' using the tree edges on p_ℓ that does not visit any of the vertices in R , and again (2) is satisfied. The remaining case is that the path crosses the boundary of R at a point y' on p_r . If x lies to the left of the vertical axis, then the path from x to p_r must use useless edges, in contradiction to the fact that all useless edges have been removed. Thus the vertex x must lie on the vertical axis. Since it lies outside the explored region, this means that x is an ancestor of u . But this also leads to a contradiction, since this means that there is a path from u to an ancestor of u , implying that there there is a directed cycle, although G is acyclic. We conclude that (2) holds.

For part (3) of the invariant, it suffices to observe that, by Observation 44, the only jump edges from the explored region start at the descendants of vertices that are enumerated (and thus the start vertices of these jump edges have also been enumerated). For part (4) of the invariant, note that, by planarity, any jump edge from the unexplored region into the explored region must also land at vertices that are enumerated in **ReachLocal**(u).

For the inductive step, consider the case where **Limleft** is updated after **ReachLocal**(z), and z is reached through a jump edge (x, z) . Let the old value of **Limleft** be l . Then x lies in the explored region bounded between **Limright** and l . The invariant implies that (x, z) is reachable from u , and hence all vertices enumerated by **ReachLocal**(z) are reachable from u . This establishes (1).

To establish part (3), let us now see that any jump edge in the new set S is reachable from u . No jump edge that begins in the region enclosed by the cycle formed by (x, z) and the tree edges connecting x and z to s can go to the unexplored region (by planarity), and thus no such edge can be in S . The only other jump edges that are added to S must start at vertices in **ReachLocal**(z) (since by Observation 44, there are no jump edges from any other vertex that is added to the explored region at this step). Since, as we have just observed, all vertices in **ReachLocal**(z) are reachable from u , this shows that all jump edges in S are reachable from u . Similar analysis of the jump edges that go from the unexplored region into the explored region shows that such jump edges must land at enumerated vertices, which establishes (4).

Let us now show that (2) holds. It suffices to consider paths from u to v that visit some vertex z' in the explored region, between **Limleft** and l , which did not appear in the enumeration of **ReachLocal**(z). Let p be a path from u to z' that visits the least number of vertices that have not yet been enumerated. Consider the first edge (x', y') in p that reaches an unenumerated vertex. The edge (x', y') must be a jump edge (or else y' would have been enumerated), and x' is either in the region bounded by **Limright** and l , or x' is enumerated in **ReachLocal**(z). In the first case, since (x, z) was the closest jump edge in S , this vertex must lie in the unexplored region. The same conclusion follows in the second case, due to the absence of useless edges. Thus p goes out of the explored region.

Since z' lies in the explored region, p must re-enter the explored region. If it re-enters the explored region via a jump edge, then by part (3) of the invariant (which we have already established), it lands at an enumerated vertex y'' . Thus (by (1), which we have already established) there is a path from u to y'' that stays entirely inside the explored region, contrary to our selection of p as the path to z' that visits the least number of vertices from the unexplored region. Thus we can conclude that p re-enters the explored region via some edge (x'', y'') that is either a tree edge or a local edge. But this means that the vertex x'' must lie on $p_\ell(s, \mathbf{Limleft})$, or $p_r(s, \mathbf{Limright})$. Assume without loss of generality that it lies on p_ℓ . As in the basis case, we know that x'' cannot be an ancestor of u (because this would yield a directed cycle), and thus x'' must be to the left of the vertical axis. The path from x'' to v cannot exit the explored area along p_r , since this would imply the existence of useless edges. If the path leaves the explored area via a jump edge, then by (3) the start vertex of this jump edge is an enumerated vertex, and thus we could have visited fewer vertices from the unexplored region. Thus the path from x'' to v must leave the explored area along p_ℓ (in which case we could have followed tree edges from x'' to this point on p_ℓ and not visited any unenumerated vertex from the explored region. This establishes (2).

A similar argument applies when **Limright** is updated, and thus the proof of the invariant is complete. From the invariant, it follows that if v is enumerated, then it is reachable from u . On the other hand, suppose v is reachable from u , but it was not enumerated. By our observations made before establishing the basis case of the invariant, v must lie in the unexplored region if it is not enumerated. Consider a path p from u to v . Let (x, y) be the edge in p that goes from the explored region to the unexplored region, as defined by the final iteration of the algorithm. This edge would have belonged to S in the final iteration, so S would not have been empty, and the algorithm would not have terminated in this iteration, a contradiction. This completes the proof of correctness of the algorithm.

□

We remark that L is the best upper bound that we have on the complexity of this problem. We do not know of a first-order reduction to UGGR.

9.3 Two Sources

In an earlier version of this work [ABC⁺06], we claimed that reachability in acyclic planar graphs with constantly many sources could be shown to be solvable in L by an easy extension of the techniques used to prove our other results. Unfortunately, this claim proved over-optimistic; we currently see only how to present logspace algorithms for reachability in planar acyclic graphs with two sources, and the extension for $O(1)$ sources remains an open problem. In this section, we sketch our algorithm for graphs with two sources.

The first step is to find an undirected path between the two sources, and then (as in Section 3) cut along this path and invert the graph, putting the two original sources on the external face. Note that, by cutting along the path, we may have created additional sources, but all of them are now on the external face.

Next (again following the approach used in Section 3) sew together $O(n)$ copies of this graph along the copies of the path that was cut apart in step 1. There is a path from u to v in the original graph if and only if there is a path from u to one of the copies of v in this new graph. In this new graph, all of the sources are still on the external face.

By adding one new vertex s , along with edges from s to the sources on the external face, we obtain an SMPD, and it suffices to solve the reachability question on this graph.

9.4 Recognition of SSPDs

We prove:

Theorem 46 *Recognition of SSPDs can be done in L.*

In order to prove this, we use the following:

Lemma 47 *In any planar graph with a single source s and sink t and no facial cycles (that is, no cycles around a face of the planar embedding), any directed cycle separates s and t . (That is, s and t cannot both be embedded in the interior (or exterior) of any directed cycle.)*

Proof: We give a proof by contradiction. Assume that there is a directed cycle D (not a facial cycle) that does not separate s and t . Assume without loss of generality that s and t are both embedded on the exterior of D . By deleting all of the vertices that are embedded outside of D , we obtain a planar graph G with no sources or sinks, such that only its external face (and no other face) is a directed cycle. We will show that this leads to a contradiction.

G has a smallest cycle C that encloses no other cycle in its interior. We consider the cycle C and its interior. Since by assumption, C is not a face of G , there are vertices in its interior; (note that if this is not the case, then C has a chord, which gives rise to a smaller directed cycle, contrary to our choice of C). Thus there has to be some edge leading from some vertex v_1 on C to one such interior vertex v_2 (or an edge from an interior vertex v_2 to a vertex v_1 on C - the reasoning for this case is similar). Given that no vertex in G is a source or a sink, we have at least one outgoing edge from v_2 . Follow that to a third vertex v_3 , and repeat the process of choosing an arbitrary outgoing edge and following that edge. Clearly, this process can end in one of two ways. Either the sequence of vertices v_1, v_2, \dots, v_k satisfy that $v_i = v_j$ for some i, j , in which case we have a smaller cycle than C lying inside C , or the sequence of vertices v_1, v_2, \dots, v_k meets C again (i.e. v_k lies on C), in which case we have again a proper cycle lying inside C contrary to the minimality of C . \square

Proof: (of Theorem 46) In the following, we are given a planar graph G along with an embedding on the plane. We perform the following tests:

1. Does G have a single source s and a single sink t ?
2. Does every face of G have a single (local) source and a single (local) sink?
3. Is G bimodal at every vertex?
4. For every vertex v of graph G , consider all the incoming edges. Delete all incoming edges at v except for the *leftmost* incoming edge (pick any arbitrary incoming edge at the sink node). Call the residual graph G_{left} . Is there a path from s to t in G_{left} ?
5. For every vertex v of G , consider all the incoming edges. Delete all incoming edges at v except for the *rightmost* incoming edge (with a similar proviso for t). Call the residual graph G_{right} . Is there a path from s to t in G_{right} ?

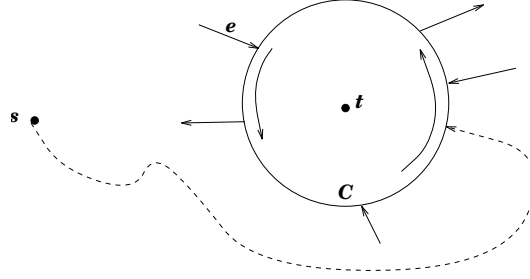


Figure 15. t inside cycle C , s outside

If all of the tests above are answered affirmatively, we claim that G is indeed an SSPD.

Observe that G_{left} and G_{right} are indegree-1 digraphs for any G .

Clearly if G is an SSPD, then by Fact 31, we know that G passes all the above tests (in this case, G_{left} and G_{right} are both trees).

So suppose G passes all the above tests, and yet has a directed cycle C . We show that this leads to a contradiction.

By Lemma 47, we only have to consider the case where the sink t lies inside C while the source s lies outside C (i.e., C separates s from t). See Figure 15.

Consider all the edges from outside C that are incoming to some vertex on C (for instance, edge e in Figure 15). Suppose the cycle C were as directed as in Figure 15, then in Step 4 where all but the leftmost incoming edges are deleted, all such incoming edges to C get deleted. So, in G_{left} among all the edges between C and the outside of C , we only have the outgoing edges from C (it is of course possible that some of the edges on C also get deleted in this process).

But now it is clear that G fails test 4, contrary to our assumption. To see this, assume that there is a directed path from s to t in G_{left} . That path intersects C at some place, and it can only be directed from the exterior of C towards C . But we deleted all of these incoming edges in constructing G_{left} – so no such path can exist.

Since we are not sure *a priori* what direction the edges on C might have, we have to include *both* tests 4 and 5. In one of these tests, the edges incoming to C from the outside will get deleted and disconnect t from s .

So, if G has a directed cycle, then there is no path from s to t in either G_{left} or G_{right} .

Thus, we have recognized SSPDs in L.

□

Corollary 48 *Let G be a single-source, single-sink directed grid graph. The problem of determining whether G has a cycle (and hence whether G provides an instance of SSGR) is in FO + UGGR.*

Proof: We need only examine the five steps in the proof of Theorem 46. The first and third are simple first-order questions. The second requires traversing the boundary of a face of the embedding to count the local sources and sinks, which is a 1GGR and hence a UGGR question. The fourth and fifth are reachability questions in a graph of *indegree* one, which are easily converted to 1GGR questions on that graph's reversal. □

9.5 Recognition of SMPDs

Theorem 49 *Recognition of SMPDs can be done in L.*

Proof: We perform the following tests:

1. We first check whether the given graph G is planar, and if so, find a planar embedding of G [AM04].
2. Check whether the digraph G has a single source. If not, return “false”.

Henceforth we can assume that G has a single source s . We first transform the given embedding so that s lies on the external face. We now need to check whether G has a cycle.

3. We construct a subgraph H of G as follows: for every vertex that is not the source, retain a single, arbitrarily chosen, incoming edge to the vertex and delete all other edges. Check whether H is a directed tree. If not, return “false”.

Suppose H is a directed tree - H clearly inherits its embedding from G . Compute a depth-first-search numbering of H . We refer to the non-tree edges in G (with respect to the tree H) as **cross edges**. In this embedding of G , the cross edges can be classified into two types:

- **Type I** edges are those going right-to-left (i.e. a cross edge (a, b) is Type I if $\mathbf{Finish}(a) > \mathbf{Finish}(b)$).
- **Type II** edges are those going left-to-right (i.e. cross edges (a, b) where $\mathbf{Finish}(a) < \mathbf{Finish}(b)$).

4. Now, we check whether G with the underlying spanning tree H has any back edge. If so, we have clearly found a cycle, so G is not an SMPD. Otherwise, delete all forward edges from H .

Create two graphs G' and G'' : in G' remove all edges from G of Type I, (but retaining all edges of Type II), and in G'' , remove all edges of Type II. We observe that both of G' and G'' are SMPDs (because any cycle in G has to use edges of both types - also we are not creating any more sources, but removing all edges of a specific type can potentially create more sinks). Thus, we can solve reachability questions in G' (or G'') in L.

5. Choose a cross edge (a, b) . If (a, b) is a Type I edge, then query G' to find whether there is a path from b to a . If there is such a path, return “false”. Likewise, if (a, b) is a Type II edge, then query G'' to find whether there is a path from b to a . Again, if there is such a path, return “false”.

It is easy to see that if G is an SMPD, then it passes all of the above tests. This is because G in such a case will neither have a back edge nor any cycle. We thus need to prove that if G passes all the tests above, it is an SMPD. For this purpose, we introduce the following terminology

Definition 50 *A (directed) cycle is minimal if the set of cross edges contained in it is minimal among all cycles with respect to inclusion.*

A directed path of tree edges that begins and ends on a directed cycle (and that does not otherwise intersect the cycle), will be called a tree chord.

It is easy to see the following:

Lemma 51 *A cycle is not minimal if it has a tree chord.*

We use the above lemma to prove:

Lemma 52 *Any minimal cycle either contains exactly one edge of Type I or contains exactly one edge of Type II.*

Proof: Consider a minimal cycle C in G . Clearly, C must contain at least one edge each of both Types I and II.

Consider any vertex v on C . The tree-path from the source s (remembering that s lies on the outer face) to v cannot intersect C : if it did, then that would be a tree chord, contradicting the minimality of C by Lemma 51.

So we can assume that for all vertices v on C , the tree-path to v does not intersect the interior of C .

Since cycle C has edges of both Type I and Type II, let us consider two edges: (a_1, b_1) of Type II, and (a_2, b_2) of Type I. Given the constraint that the tree-paths cannot intersect the interior of C , together with the constraints that the tree-path to a_1 is to the left of the tree-path to b_1 (because edge (a_1, b_1) is of Type II) and the tree-path to a_2 is to the right of the tree-path to b_2 (because edge (a_2, b_2) is of Type I), the situation is as in Figure 16. The dotted paths from s to the vertices on C are the tree-paths.

But now we see that, under the constraint of planarity, any edge (c, d) lying on C between b_1 and a_2 has to be such that the tree-path to c lies to the left of the tree-path to d . So any cross edge lying between b_1 and a_2 has to be of Type II. The same holds for any cross edge lying between b_2 and a_1 .

The symmetric case where the edge (a_1, b_1) is of Type I and (a_2, b_2) of Type II is handled similarly.

Thus we have proven that any minimal cycle can contain exactly one edge of Type I or exactly one edge of Type II.

□

Hence if there is a cycle in G , then there is a minimal cycle that contains exactly one edge of Type I or Type II by Lemma 52, and we discover such a minimal cycle in Test 5. We have thus proved Theorem 49. □

Clearly, this algorithm gives an alternative logspace algorithm for recognition of SSPDs. Also, it yields an algorithm to recognize if a graph is a MMPD with two sources. (Namely, given G , first check that G has two sources, and then apply the

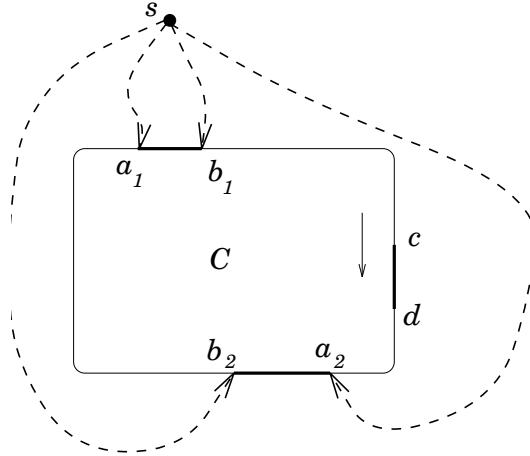


Figure 16. Tree-paths to edges around C

construction of Section 9.3 to create a new graph G' . It is easy to verify that G' is a SMPD if and only if G is a MMPD with two sources.

In contrast to Corollary 48, we do not know how to adapt this proof to determine whether a single-source grid graph has a cycle (and hence whether it provides an instance of SMGGR) in the class $\text{FO} + \text{UGGR}$. This is because the algorithm presented above appeals to the SMGGR recognition algorithm of Theorem 38, and we do not know how to carry out this algorithm in $\text{FO} + \text{UGGR}$.

9.6 Planar digraphs with a few cycles

In the above, we have considered the reachability and recognition questions for different classes of DAGs. We may now ask: is the acyclicity condition essential for being able to perform the above tasks in L ? Here we show that we *can* solve some reachability questions, even when the graph has a few cycles, in L .

Consider the class \mathcal{G} of planar graphs that have a single source and a single sink and are embedded in the plane so that they have no *facial* cycles (no faces that form directed cycles). Note that the recognition problem for graphs of the class \mathcal{G} is easily in L . We prove:

Theorem 53 *Reachability questions in graphs from the class \mathcal{G} can be solved in L .*

Observe that any SSPD belongs to the class \mathcal{G} . Also note that a graph $G \in \mathcal{G}$ is not necessarily bimodal.

Proof: Given an embedded planar graph G with a unique source s and sink t , and no facial cycles, Lemma 47 tells us that any cycle in the graph separates s and t .

Now we proceed to reduce reachability questions in G to a reachability question in an SMPD.

We can find a path (not necessarily a directed path) from s to t in L . Now we apply the cut-and-paste method from Section 3 by cutting along the path between s and t . As in Section 3, after cutting along the path from s to t and inverting the graph inside out to get a graph G' , we paste n copies of G' along the path from s to t to get a graph G'' which preserves the connectivity of G (in the sense that there is a path from u to v in G if and only if there is a path from one of the copies of u to one of the copies of v in G'') and has s and t on the outer face. However, in this process, because the path from s to t is not a directed path, we have introduced some more sources and sinks on the outer face. Now we can add a single source vertex and connect it to all the sources in G'' to get a graph G''' . One can verify that G''' is an SMPD, since it still satisfies the properties of \mathcal{G} , but now s and t are on the external face, and thus there can be no directed cycles. (That is, any cycle in the original graph is destroyed when we cut along the undirected path). Hence reachability in G''' (and thus in G) can be solved in L . \square

Theorem 54 *Reachability questions in outerplanar digraphs can be solved in L .*

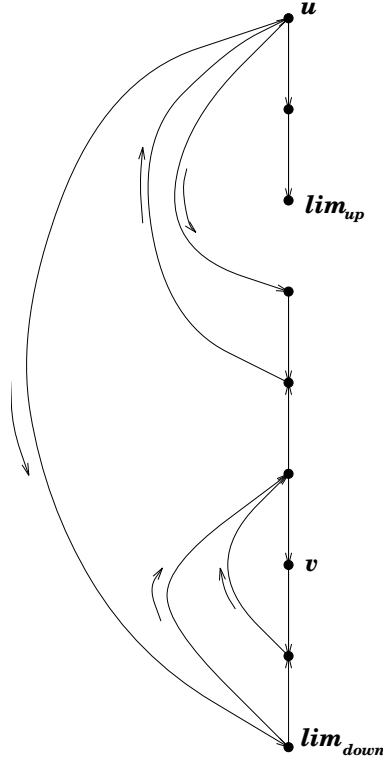


Figure 17. A 1-page embedding

Note that outerplanar digraphs, even DAGs, are *not* series-parallel digraphs as considered by [JLR06]. The result above is trivial for outerplanar DAGs, since all the sources and sinks lie on the same face, and we can reduce this case to an SMPD.

In the language of book embeddings (see [Yan89] for instance), outerplanar graphs are exactly the ones that have 1-page embeddings: in short, all the vertices are laid out on the spine of the book, and all the edges are on a single page.

Proof: Suppose we have a 1-page embedding of outerplanar graph G given to us (here, the vertices are all on the spine as in Figure 17).

Here, the graph G is not acyclic. The instance to the reachability question is (G, u, v) and we are to find whether v is reachable from u . We can assume that u is the topmost vertex on the spine of the embedding.

We keep two markers lim_{up}, lim_{down} , ranging over the set of vertices unioned with $\{\infty\}$.

Call the edges on the spine *ordinary* edges and the edges not on the spine *jump* edges. The algorithm is as follows:

1. Initialize the markers as $lim_{up} = u, lim_{down} = \infty$.
2. Go down from lim_{up} as far as you can using only ordinary edges. Go up from lim_{down} as far as you can using only ordinary edges. Call the region between u and lim_{up} and lim_{down} and ∞ on the spine the *explored* region E .
3. Consider all jump edges between the explored region E and the unexplored region. The unexplored region is thereby an “interval” on the spine of the embedding. Consider the jump edges j_1, j_2 (if any) that land on vertices closest to the target vertex v on the spine, from either side (from above or below).
4. Let $j_1 = (a, b)$ be the jump edge landing on a vertex closest to the target v from below (if any). Update $lim_{down} = b$. Similarly, let $j_2 = (c, d)$ be the jump edge landing on a vertex closest to the target v from above (if any). Update $lim_{up} = d$.
5. Go to Step 2.
6. If v is discovered at some step then return “true”. If at some step neither lim_{up} nor lim_{down} can be changed and v has not yet been discovered, then return “false”.

In order to prove that the above procedure is correct, we need to show: if v is reached by our algorithm, then v is indeed reachable from u . This follows by an easy induction on lim_{up}, lim_{down} . Specifically, we have to convince ourselves that vertices lim_{up}, lim_{down} are always reachable from u . This follows via an easy induction, using the 1-page embedding of the graph.

On the other hand, if v is not reached by the algorithm, that means that the algorithm stopped at a stage when it could change neither lim_{up} nor lim_{down} any more. Clearly, in a run of the algorithm, on the spine, lim_{up} always stays above v (or is equal to v), and likewise, lim_{down} always stays below v (or equals v). Hence, when the algorithm stops there is no jump edge from the explored region to the interval on the spine between lim_{up}, lim_{down} (and also lim_{up}, lim_{down} cannot be extended any further using ordinary edges). But this means v is not reachable from u .

(We remark that Raghunath Tewari has pointed out that an alternate proof is possible, by noticing that the dual of an outerplanar graph is a tree, and searching for a separating cut in the dual graph. A similar observation was also made much earlier by Papakostas [Pap95].) \square

10 Conclusions and Open Problems

Any problem defines the complexity class of those problems reducible to it. There is a general phenomenon whereby interesting problems, such as general reachability, define interesting classes, such as NL. The GGR problem and its subproblems as outlined here define a hierarchy of new classes, whose relations to each other and to the standard classes between TC^0 and NL are shown in Figure 18. (We include the fact that $GGR \in UL$ [BTV07].)

Are these problems and classes interesting? We argue, particularly in Section 7.4, that many of them have interesting alternate formulations, sometimes not appearing to involve graphs at all. The computational actions of searching on a grid, of searching in a maze, of following a laid-out path on a grid, and so forth strike us as fundamental ones, well worth studying.

The natural next questions concerning this hierarchy are whether any of the upper and lower bounds can be improved, or whether additional containment relations exist among the new classes. In particular, is the SMGGR problem reducible to UGGR? The proof of Theorem 38, like the proofs for SSPD's, seems to mostly involve following a laid-out path on a grid, but we do not yet see how to formulate it solely in terms of this. The question also remains as to whether we can detect cycles in a general single-source graph in $FO + UGGR$ – the algorithm presented here relies on SMPD reachability but this may not be necessary.

Our logspace algorithm for SMPD reachability expands the class of graphs for which Jakoby *et al.* ([JLR06]) provided logspace reachability algorithms – but our results are not completely extensions of theirs. They proved that *counting* the number of paths between two vertices of a series-parallel digraph can be done in logspace. We have no new upper or lower bounds for the counting problem in the classes of graphs that we study. Another shortcoming of our reachability algorithms is that they provide no clue about how to find a *shortest* path, and we have no lower bounds showing that finding a shortest path is harder than the reachability problem.

It is entirely plausible that reachability in planar graphs, like planarity testing itself, is in L. Our work here fits into a general program of expanding the classes of planar graphs for which we have logspace reachability tests. A natural intermediate goal on the way to general planar graphs is *acyclic* planar graphs, which would be called MMPD in our notation. Also, while we can easily show that reachability questions in SSPDs reduce to non-reachability, we are not able to show the same for SMPDs.

We close by noting that the results and techniques introduced in this work have subsequently proved useful in classifying the complexity of other problems [CD06, DKLM07]. Recently, logspace algorithms for reachability in some other classes have been presented by Jakoby and Tantau [JT07].

Acknowledgments

Earlier versions of this work appeared in conference proceedings as [ADR05] and [ABC⁺06]; we thank the reviewers for those conferences and the referee of this journal version for their detailed and insightful comments, and for correcting some of the errors that appeared in earlier versions. We thank Andreas Jakoby and Peter Bro Miltersen for their helpful comments on an earlier draft. We thank Raghunath Tewari for recalling our attention to the question of whether we could extend our reachability results for SMPDs to graphs with a constant number of sources, and for his comments about reachability in outerplanar graphs. We also thank DIMACS REU participants Tomáš Vyskočil and Jan Kynčl for suggesting that it should be possible to simplify our original proof of reachability in SMPDs; the argument in this version of the paper reflects their

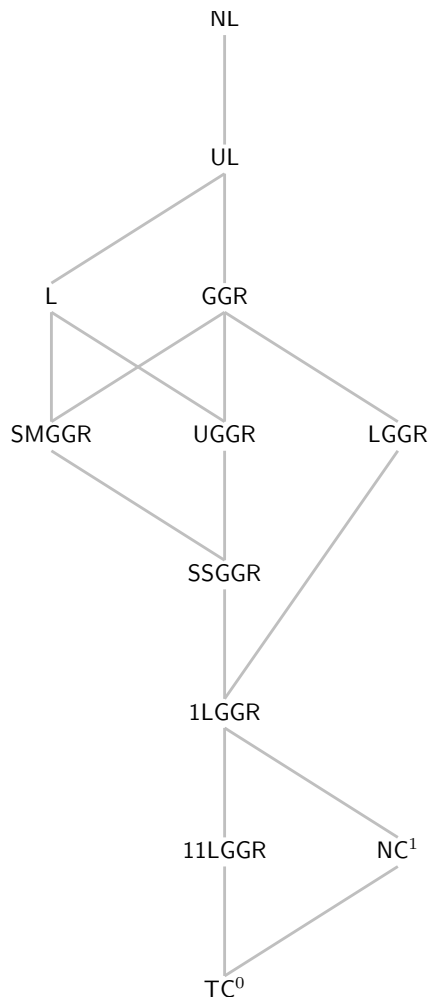


Figure 18. The Hierarchy of GGR Classes

suggestions. The first and fifth authors acknowledge the support of NSF Grant CCF-0514155. The second author gratefully acknowledges helpful discussions over the years on this topic with Sam Buss, Steve Cook, Bill Hesse, Pierre McKenzie, and Charlie Rackoff. The third and the fourth authors would like to acknowledge interesting and illuminating discussions with Meena Mahajan and K. V. Subrahmanyam on planar reachability and related topics. We acknowledge many people for sharing with us their thoughts about what was already known about this problem, including Til Tantau, Omer Reingold, Paul Beame, Jeff Edmonds, Anna Gal, Vladimir Trifonov, and K.V. Subrahmanyam.

References

- [ABC⁺06] E. Allender, D. A. M. Barrington, T. Chakraborty, S. Datta, and S. Roy. Grid graph reachability problems. In *IEEE Conference on Computational Complexity*, pages 299–313, 2006.
- [ADR05] E. Allender, S. Datta, and S. Roy. The directed planar reachability problem. In *Proc. 25th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, number 1373 in Lecture Notes in Computer Science, pages 238–249. Springer, 2005.
- [AM04] E. Allender and M. Mahajan. The complexity of planarity testing. *Information and Computation*, 189:117–134, 2004.
- [ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.
- [Bar89] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 . *Journal of Computer and System Sciences*, 38:150–164, 1989.
- [BH91] S. R. Buss and L. Hay. On truth-table reducibility to SAT. *Inf. Comput.*, 91(1):86–102, 1991.
- [BK78] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, 1978.
- [BLMS98] D. A. M. Barrington, C.-J. Lu, P. B. Miltersen, and S. Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, number 1373 in Lecture Notes in Computer Science, pages 73–83. Springer, 1998.
- [BST93] H. Buhrman, E. Spaan, and L. Torenvliet. The relative power of logspace and polynomial time reductions. *Computational Complexity*, 3:231–244, 1993.
- [BTV07] C. Bourke, R. Tewari, and N.V. Vinodchandran. Directed planar reachability is in unambiguous logspace. In *IEEE Conference on Computational Complexity*, pages 217–221, 2007.
- [Bus06] S. Buss. Polynomial-size Frege and resolution proofs of st-connectivity and Hex tautologies. *Theor. Comput. Sci.*, 357:35–52, 2006.
- [CD06] T. Chakraborty and S. Datta. One-input-face MPCVP is hard for L, but in LogDCFL. In *Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, number 4337 in Lecture Notes in Computer Science, pages 57–68, 2006.
- [CM87] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987.
- [DKLM07] S. Datta, R. Kulkarni, N. Limaye, and M. Mahajan. Planarity, determinants, permanents, and (unique) matchings. In *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, (CSR 2007)*, number 4649 in Lecture Notes in Computer Science, pages 115–126, 2007.
- [EIT⁺92] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. R. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *J. Algorithms*, 13(1):33–54, March 1992. (Corrigendum in Vol 15, 1993.).

- [Ete97] K. Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, Jun 1997.
- [GT87] J. Gross and T. Tucker. *Topological Graph Theory*. John Wiley and Sons, New York, first edition, 1987.
- [HKRS97] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.
- [HRS93] M. Halldórsson, J. Radhakrishnan, and K. V. Subrahmanyam. Directed vs. undirected monotone contact networks for threshold functions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 604–613, 1993.
- [Hus95] T. Husfeldt. Fully dynamic transitive closure in plane dags with one source and one sink. In *Proc. 3rd ESA*, volume 955 of *Lecture Notes in Computer Science*, pages 199–212, 1995.
- [Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16(4):760–778, 1987.
- [Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.
- [Imm98] Neil Immerman. *Descriptive Complexity*. Springer Graduate Texts in Computer Science, 1998.
- [JLR06] A. Jakoby, M. Liskiewicz, and R. Reischuk. Space efficient algorithms for directed series-parallel graphs. *Journal of Algorithms*, 60:85–114, 2006.
- [JT06] A. Jakoby and T. Tantau. personal communication, 2006.
- [JT07] A. Jakoby and T. Tantau. Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, pages 216–227, 2007.
- [Lan97] K.-J. Lange. An unambiguous class possessing a complete set. In *14th International Symposium on Theoretical Aspects of Computer Science (STACS)*, number 1200 in *Lecture Notes in Computer Science*, pages 339–350. Springer, 1997.
- [LL76] R. Ladner and N. Lynch. Relativization of questions about log space reducibility. *Mathematical Systems Theory*, 10:19–32, 1976.
- [LMS06] N. Limaye, M. Mahajan, and J. Sarma M. N. Evaluating monotone circuits on cylinders, planes, and torii. In *Proc. 23rd Symposium on Theoretical Aspects of Computing (STACS)*, *Lecture Notes in Computer Science*, pages 660–671. Springer, 2006.
- [MT01] B. Mohar and C. Thomassen. *Graphs on Surfaces*. John Hopkins University Press, Baltimore, first edition, 2001.
- [MV00] M. Mahajan and K. R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs. In *ACM Symposium on Theory of Computing (STOC)*, pages 351–357, 2000.
- [NTS95] N. Nisan and A. Ta-Shma. Symmetric logspace is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995.
- [Pap95] A. Papakostas. Upward planarity testing of outerplanar dags. In *Graph Drawing (GD94)*, number 894 in *Lecture Notes in Computer Science*, pages 298–306, 1995.
- [RA00] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal of Computing*, 29:1118–1131, 2000.
- [Rei05] O. Reingold. Undirected ST-connectivity in log-space. In *Proceedings of the 37th Symposium on Foundations of Computer Science*, pages 376–385. IEEE Computer Society Press, 2005.

- [Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.
- [Tho90] C. Thomassen. Embeddings of graphs with no short noncontractible cycles. *J. Comb. Theory Ser. B*, 48(2):155–177, 1990.
- [TW08] T. Thierauf and F. Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008. to appear, also ECC report TR07-068.
- [Yan89] M. Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, 1989.
- [Yan91] H. Yang. An NC algorithm for the general planar monotone circuit value problem. In *SPDP: 3rd IEEE Symposium on Parallel and Distributed Processing*. ACM Special Interest Group on Computer Architecture (SIGARCH), and IEEE Computer Society, 1991.