Electronic Colloquium on Computational Complexity, Report No. 149 (2005)

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 st-connectivity, which is the standard complete problem for NL. *Grid graphs are a useful tool in this regard, since*

- *reachability on grid graphs is logspace-equivalent to reachability in general planar digraphs, and*
- *reachability on certain classes of grid graphs gives 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.

In addition to explicating the structure of L*, another of our goals is to expand the class of digraphs for which connectivity can be solved in logspace, by building on the work of Jakoby* et al. *[11], who showed that reachability in series-parallel digraphs is solvable in* L.

Our main results are:

- *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 if 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 [5].*
- *Series-Parallel digraphs are a special case of single-source-single-sink planar 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 have long played a fundamental role in complexity theory. The general st-connectivity problem in directed graphs is the standard complete problem for NL, while the st-connectivity problems for directed graphs of outdegree 1 [9, 7] and undirected graphs [13] are complete for L. It follows from [3] that reachability in directed graphs of width $O(1)$ (or even width five, with outdegree 1) is complete for NC^1.

Grid graphs are a special class of planar graphs whose vertices are located on grid points, and whose vertices are adjacent only to their immediate horizontal or vertical neighbors. Barrington *et al.* showed [4] that st-connectivity in width k (directed or undirected) graphs is complete for depth k AC^0 under first-

[*]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: barring@cs.umass.edu.

[‡]Chennai Mathematical Institute, Chennai, India. e-mail: tanmoych1985@gmail.com.

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

[¶]Department of Computer Science, Rutgers, the State University of NJ. Supported in part by NSF Grant CCF-0514155. email: samroy@paul.rutgers.edu.

ISSN 1433-8092

order projections. In this paper we study grid graphs without any width restrictions. The general grid-graph reachability problem (GGR) is equivalent to the st-connectivity problem in directed planar graphs (and graphs of genus one) under logspace reducibility [1]. The best upper bound known for GGR is NL, although a slightly better upper bound is known for so-called "layered" grid graphs (LGGR): LGGR $\in \mathsf{UL} \cap \mathsf{coUL}$ [1].

Our focus in this paper is primarily on classes of grid graphs whose reachability problem is solvable in logspace. Reachability in *undirected* grid graphs (UGGR) was studied by Blum and Kozen [5]; they showed that UGGR is solvable in logspace (which was superseded a quarter-century later by the work of Reingold [13]). Buss has studied UGGR in connection with tautologies arising from the game of HEX [6] (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 if 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 AC^0 reductions (and even under first-order projections). We show that these problems are hard for NC^1, and thus this gives a cluster of natural problems that are candidates for having complexity intermediate between NC^1 and L, since even the general GGR problem is not known to be hard for L under AC^0 reductions.

Jakoby, Liskiewicz, and Reischuk showed that reachability in series-parallel digraphs is solvable in logspace [11], thus solving the reachability question for an important subclass of planar directed 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 [11].

- *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 5.

The rest of the paper is organized as follows. In Section 2 we introduce the various grid graph problems that we will be considering, and present reductions showing how these problems relate to each other. In Section 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. Our hardness results are presented in Section 4. Our logspace algorithms for SSPD and SMPD are presented in Section 5. We conclude with open questions in Section 6.

2 Versions of the GGR Problem

We begin 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.

2.1 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 [10], which exploits the close connections between AC^0 and first-order logic. In particular,

AC^0-Turing reducibility ($\leq_T^{\mathrm{AC}^0}$) 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 [10]. For this reason, we sometimes refer to $\leq_T^{\mathrm{AC}^0}$ reductions as FO reductions. The class of problems $\leq_T^{\mathrm{AC}^0}$ reducible to A is sometimes denoted as $\mathsf{FO}+A$.

Immerman also gives good motivation for studying a restricted form of $\leq_m^{\mathrm{AC}^0}$ reductions called *first-order projections* ($\leq_{\mathrm{proj}}^{\mathrm{FO}}$). 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.

2.2 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 tuples (G, s, t) where G is a directed grid graph, s and t are vertices on the **boundary** of G, and there is a path from s to t in G.
- The problem LGGR is the set of tuples (G, s, t) where G is a **layered** directed grid graph, having **only east and south edges**, and there is a path from s to t.
- The problem 1GGR is the set of tuples (G, s, t) where G is a directed grid graph of **out-degree at most 1** and there is a path from s to t.
- The problem 11GGR is the set of tuples (G, s, t) where G is a directed grid graph of **in-degree and out-degree at most 1** and there is a path from s to t.

It is obvious that 11GGR is a special case of 1GGR. LGGR is a special case of GGR-B because given a layered graph with vertices s and t, we can clearly restrict our attention to the rectangle with s at its northwest corner and t at its southeast corner – if there is no such rectangle, there can be no path from s to t. The local and global restrictions are orthogonal, so that the three global conditions (general, boundary, and layered) and three local conditions (general, out-degree 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 4 that it is hard for the class TC^0.

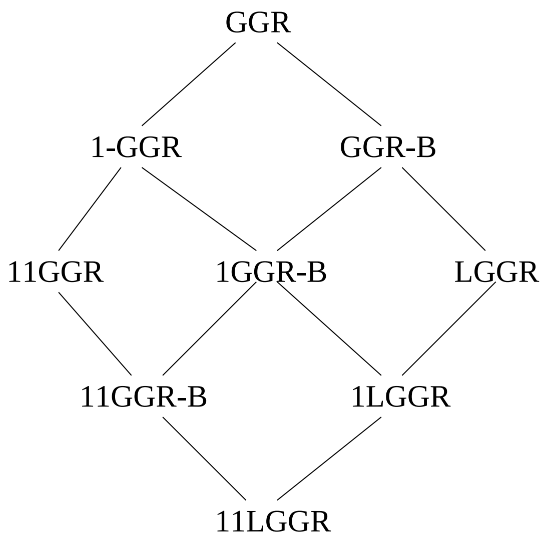

Figure 1. Nine GGR problems.

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.

2.3 Undirected GGR

One of the most natural local restrictions on a graph is **undirectedness**. Long before Reingold[13] showed that the undirected reachability problem is in L, Blum and Kozen [5] 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 2.1. *The problems UGGR, UGGR-B, 1GGR, 1GGR-B, 11GGR, and 11GGR-B are all equivalent under first-order projections.*

Proof. (of Theorem 2.1) We will show that 1GGR$\leq^{\mathrm{FO}}_{\mathrm{proj}}$UGGR$\leq^{\mathrm{FO}}_{\mathrm{proj}}$ UGGR-B $\leq^{\mathrm{FO}}_{\mathrm{proj}}$ 11GGR-B $\leq^{\mathrm{FO}}_{\mathrm{proj}}$1GGR, appealing to Section 3 for the second reduction and observing that the last reduction is trivial.

Lemma 2.2. 1GGR$\leq^{\mathrm{FO}}_{\mathrm{proj}}$UGGR

Proof. The well-known general reduction from out-degree one reachability to undirected reachability works without modification for grid graphs. Given an out-degree 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. □

Lemma 2.3. UGGR-B$\leq^{\mathrm{FO}}_{\mathrm{proj}}$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 in-degree and out-degree 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 $(u + 1/3, v + 1/3)$.

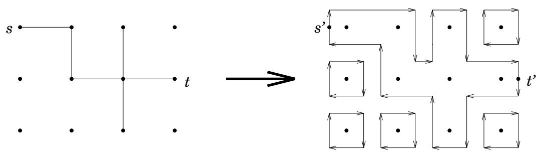

Figure 2. An undirected grid graph and its in-1-out-1 graph.

- 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 $(u - 1/3, v + 1/3)$.
- 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 in-degree and out-degree 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 2 shows the result of this construction on a small undirected graph. □

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

2.4 Five Problems

The results of the preceding section and of Section 3 reduce our nine problems to five. If we close each un-

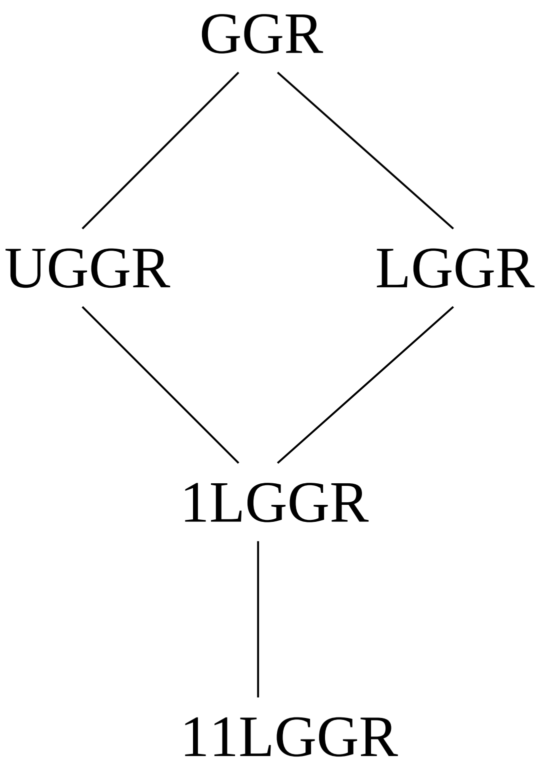

Figure 3. The five surviving GGR problems.

der first-order reductions, we get a hierarchy of complexity classes within NL and (as we shall see in Section 4) above TC^0. Since each problem has a number of interesting alternate formulations, we spend some time looking at each in turn:

2.4.1 GGR

The general GGR problem is in NL, of course, but we have no better upper bound. As shown by Allender *et al.*[1], it is equivalent under *logspace* reductions to the general planar reachability problem. (Our argument in Section 3 that GGR and GGR-B are equivalent is in fact a simplification of the argument there that general planar reachability reduces to its boundary special case.)

There is an easy (first-order projection) reduction from GGR-B to its complement, grid-graph non-reachability with s and t on the boundary. This is because 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 [4].) The reachability problem for general graphs reduces to its complement by the Immerman-Szelepcsenyi theorem [8, 15], but this much simpler reduction provides some weak evidence that GGR is not complete for NL.

2.4.2 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[6], 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 2.4. *A language is in the class* $\mathsf{FO} +$ UGGR *iff it projection-reduces to* UGGR.

Proof. (of Proposition 2.4) We show that the set of languages that projection-reduce to UGGR-B, and hence (by Section 3) to UGGR, is closed under $\leq_T^{\mathrm{AC}^0}$ reductions. We give an inductive argument on the depth of the circuits computing the $\leq_T^{\mathrm{AC}^0}$ 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

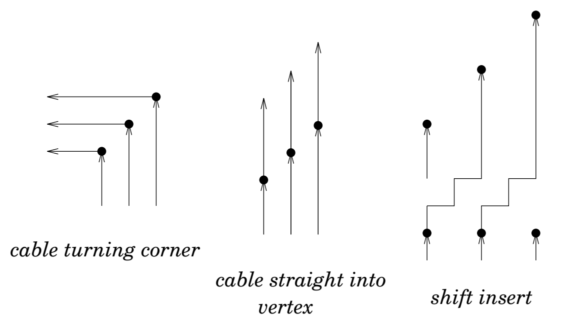

Figure 4. The Construction of Proposition 2.5

details are straightforward to fill in; by simple padding we may assume that all of the graphs G_w are the same size. □

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

Proposition 2.5. *If G is a grid graph of in-degree and out-degree each at most one, the following predicate projection-reduces to* UGGR*: DIST(s, t, k) $\leftrightarrow$ the path out of s reaches t in exactly k steps.*

Proof. (of Proposition 2.5) We first note that we can determine in $\mathsf{FO} + \text{UGGR}$ whether a given vertex, or a given edge, is on the path from s to t. Then we can define a graph $H_{s,t,k}$ where each vertex along the path is replaced by $k + 1$ copies, in a diagonal line from northeast to southwest, and 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 4.) Note that this graph H also has in-degree and out-degree at most 1. Then 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 $\mathsf{FO}+\text{UGGR}$, and thus by Proposition 2.4 we can define H as a first-order projection of G. □

In Section 5 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 in-degree and out-degree 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 2.6. *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* $\mathsf{FO} + \text{UGGR}$*: start time of a vertex, finish time of a vertex, depth of a vertex, and whether one vertex is an ancestor of another.* □

2.4.3 LGGR

The most 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. Allender *et al.*[1], following the method of Reinhardt and Allender [14], show that LGGR is in the class UL – it is the language of a nondeterministic logspace machine that never has more than one accepting run on the same input. But it is known [14] that the non-uniform versions of UL and NL are the same, and it is entirely plausible that the classes themselves are the same.

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 is closed under complementation. 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. Limaye *et al.* [12] give a nice survey of the various versions of this problem along with some new results.

2.4.4 1LGGR

The 1LGGR problem has a useful alternate form:

Lemma 2.7. 1LGGR *is equivalent to the reachability problem on directed grid graphs that have some east edges, all possible south edges, and no north or west edges.*

Proof. (of Lemma 2.7) We first reduce this new problem to 1LGGR. 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 an out-degree one layered grid graph H as follows. 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 H has out-degree 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, we first assume that G is a layered grid graph with out-degree *exactly* one, except at the boundary. 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.

It remains to show that 1LGGR, where each vertex has degree *at most* one, reduces to the special case treated above. Let G be a layered grid graph with out-degree at most one. We make a new graph H from G by making two copies v_1 and v_2 of each vertex v, with v_1 situated to the northeast of v_2. We replace each edge (u, v) of G by a pair of parallel edges (u_1, v_1) and (u_2, v_2), preserving the out-degree of one. If v is a sink in G, we connect both v_1 and v_2 to w_2, where w is the vertex to the east of v. Then there is a path from s to t in G iff there is a path from s_1 to t_1 in H, and H has out-degree exactly one except at the boundary. □

The language of problems projection-reducible to 1LGGR is closed under complement. 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. As with UGGR-B, we can use series and parallel connection to show that any language first-order reducible to 1LGGR is projection-reducible to it.

As we will see in Section 4, 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 $\mathsf{L} = \mathsf{NC}^1$ is the question of whether sequential computations using only log space can be parallelized to a certain extent.

Here is a problem that looks inherently 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 $\in \mathsf{NC}^1$, then this code can be parallelized in some way that is not readily apparent.

2.4.5 11LGGR

The easiest problem in our hierarchy, 11LGGR, has an interesting alternate formulation. By expanding each layer into up to n layers (where n is the width of a layer), we can projection-reduce a 11LGGR problem to one where the edges in each layer divide into two consecutive intervals, one of south edges and one of east edges, with a source or a sink between these two intervals. We can recast this latter problem in terms of the following data structure: a varying number n of items are in locations from 0 to $n-1$, and our operations are $insert(i)$, which places a new element in location i and moves the higher-numbered elements up, and $delete(i)$, which removes the element in location i and moves the higher-numbered elements down. Determining whether a sequence of such inserts and deletes removes a particular element with its last $delete$ is complete for the class of problems first-order reducible to 11LGGR.

2.5 Acyclicity and Single-Source

We cannot in general tell whether a given directed grid graph is **acyclic**, because this problem is hard for GGR-B and thus (by Section 3) for GGR. But in Section 5 we will present algorithms for two special cases of general acyclic GGR, and in these two cases we can decide whether the graph is acyclic in $\mathsf{FO}+$ UGGR. These are the **single-source** problem SMGGR and the **single source, single-sink** problem SSGGR. (In each case we will assume that the source occurs on the boundary of the grid graph.) Even the latter problem is non-trivial in our hierarchy:

Lemma 2.8. 1LGGR$\leq_{\text{proj}}^{\text{FO}}$SSGGR

Proof. (of Lemma 2.8) Appealing to Lemma 2.7, 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. Note that the source is on the boundary. ☐

Since most of our arguments in Section 5 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 $\mathsf{FO}+$ UGGR in the case of grid graphs.

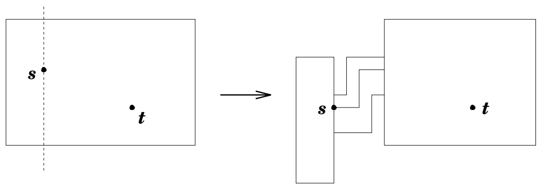

Figure 5. Putting s and t on the same row.

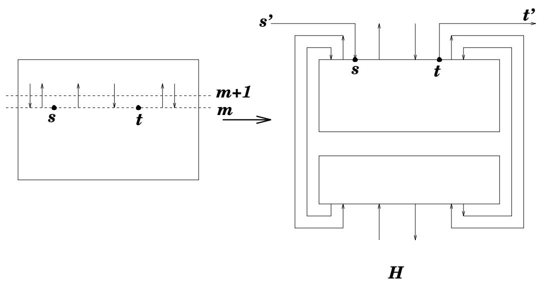

Figure 6. The basic gadget H

3 The Boundary Construction

In this section we show that each of the problems GGR, UGGR, and 1GGR reduces via first-order projections to the special case where s and t are on the external boundary. For simplicity, we first consider GGR.

Theorem 3.1. GGR$\leq_{\text{proj}}^{\text{FO}}$GGR-B.

Proof. (of Theorem 3.1) Let G be a grid graph. Without loss of generality, s and t appear on the same horizontal row of G; call this row m. (If this is not true, then add some paths to effect a vertical shift of part of the grid, as illustrated in Figure 5.) We may also 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, and 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 the two copies of v, going around the closest side bound-

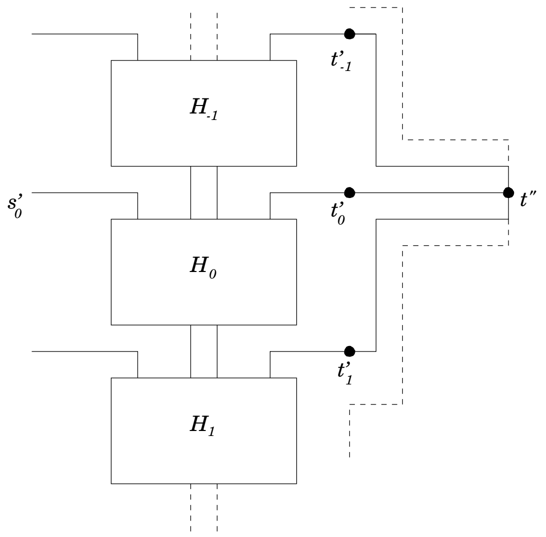

Figure 7. Connecting multiple copies of H

ary, and directed the same way as the edge that passes through v in G, as illustrated in Figure 6. Also as illustrated in Figure 6, 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 7.) A simple inductive argument shows that there is a path from s to t in G 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. □

Corollary 3.2. UGGR$\leq^{\text{FO}}_{\text{proj}}$UGGR-B *and* 1GGR$\leq^{\text{FO}}_{\text{proj}}$1GGR-B

Proof. If G has outdegree one, then the graph H' also has outdegree one. If G 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''. □

4 Lower Bounds

4.1 A TC^0 Lower Bound For 11LGGR

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

Theorem 4.1. *The problem* 11LGGR *is complete for* TC^0 *under first-order reductions.*

Proof. (of Theorem 4.1) 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 each diagonal, the vertices with $i+j = k$ for each k, have edges all in the same direction. 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 in-degree and out-degree 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. □

We can define a special case of 11LGGR that is *complete* for TC^0. Suppose that the in-degree and out-degree 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 11LGGR. 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 11LGGR problems falling within the special case, the special case is complete for TC^0.

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

We now show that except for the minimal problem 11LGGR, 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.* [11] 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 4.2. *The problem 1LGGR is hard for the class* NC^1 *under first-order projections.*

Proof. (of Theorem 4.2) 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 8 illustrates the construction.

As we observed in Section 2.3, 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 complete for NC^1 under such projections.

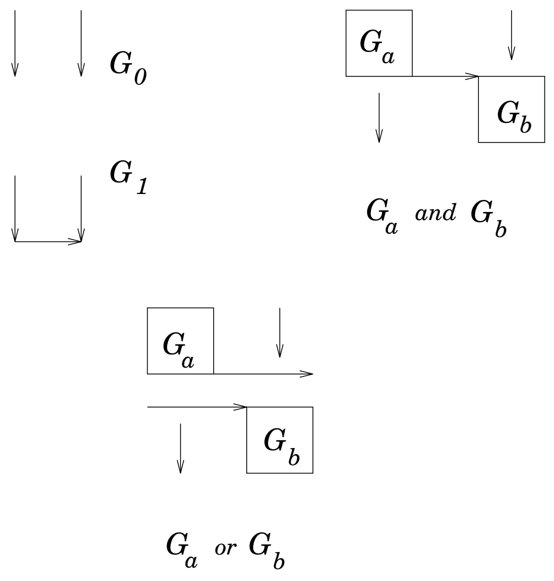

Figure 8. The construction of G_ϕ. All south edges are present.

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 of side a and b respectively, then G_ϕ is a square graph of side $a + b$ with G_α in its northwest corner and G_β in its southwest 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 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.

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. ☐

5 Acyclic Single-Source Graphs

Definition 5.1. *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 1.
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 [16]).*
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) the first 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 w.r.t. the spanning tree G can be computed by a* L*-transducer.*

It is easy to see that forward edges in T can be deleted without affecting the reachability predicate. (An 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, we assume from now on that there are no forward edges. We classify edges w.r.t. the spanning tree obtained above as follows:

Definition 5.2. *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.*

Since we may consider any face to be the external face of the embedding, we assume without loss of generality that s is on the external face. Thus no jump edges go "over the top" of the graph, around s.

We observe the following:

Observation 1. *Any subgraph of an* SMPD *that does not contain any jump edges, 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 formed by local edges does not strictly contain any vertex. ☐

Definition 5.3. *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.*

5.1 Reachability in SSPDs

Theorem 5.4. SSPD *reachability is in* L.

Proof. (of Theorem 5.4) We first state a lemma regarding the set of vertices reachable from a fixed vertex in a given SSPD.

Lemma 5.5. *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.*

Thus given vertices u and v, in order to determine whether there is a directed path from u to v, we just 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 u. (For example, we could take the reverse of the left-most path from u to s in the SSPD formed by reversing all edges in the given SSPD.) □

Proof. (of Lemma 5.5)

To see that each such vertex y is indeed reachable from x, we note that the subgraph in this region is itself a SSPD, and then appeal to Fact 1.

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 B 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 the vertices lying either strictly within R or on its right boundary, contradicting the choice of w. □

Corollary 5.6. *The problem* SSGGR *is in* $\mathsf{FO} +$ 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 1 as a first-order projection on G, and then by Theorem 2.6 we can compute all the predicates necessary to define the depth-first search of this tree in $\mathsf{FO} +$ UGGR. The argument of Theorem 5.4 refers only to reachability in graphs of out-degree one, which are computable in $\mathsf{FO} +$ UGGR by Lemma 2.2. □

5.2 Reachability in SMPDs

Theorem 5.7. SMPD *reachability is in* L.

Proof. (of Theorem 5.7) We defer to later the question of how to recognize if 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 if there is a path from u to v.

We may restrict attention to the special case where s and u are both on the external face, and where u appears on the rightmost path of the spanning tree T. (To see this, we first note that if we are given an arbitrary SMPD G, we can build a spanning tree as discussed above, and thus we can find a directed path from s to u. Now we use the argument presented in Section 2 of [1], where it is shown how to embed two vertices on the external face by first "cutting along" a path between the vertices to create a new face, and then "inverting" the graph so that this new face becomes the external face. In the special case where we have a *directed* path from s to u and G is a DAG, this construction has the property that *no new sources are created* and *no path from u to v is lost*. Thus we have created a graph with s and u on the external face (and in fact there are two directed paths from s to u along the external face). We create our spanning tree T so that the edges appearing in the directed path along the right side of the external face are all included in T, and now we have guaranteed that that u appears on the right-most path of T.)

It is convenient also to add a new vertex w that is the leftmost child of s in the tree, along with a jump edge from the rightmost child of u to w. This clearly creates no new paths from u to v (but it does provide a reachable jump edge to the far left of the graph, which simplifies some of our notation).

It is easy in logspace to see if v is a descendant of u (in which case there is a path, since T is a directed tree) and thus we assume for now that v is not a descendant of u, and thus that it is to the left of u in T. Given any vertex x, T partitions the vertices into the set of ancestors of x, the descendants of x, and the vertices to the *right* and *left* of x. The adjectives "right" and "left" give partial orders on the set of vertices (where two vertices on the same path in T are neither to the right nor to the left of each other). Let us call a local edge (x, y) *useless* if x is to the right of v and (x, y) is directed to the right, or if x is to the left of v and (x, y) is directed to the left.

Fact 2. *If there is a path from u to v, then there is a path that uses no useless edges.*

Proof. Assume that we have a path from u to a useless edge (x, y) and then to v, where x is to the right of v. Either this path intersects the tree path from s to y, or it doesn't. If it does, then we can clearly construct a path from u to y, and then to v, that avoids (x, y). Otherwise, y is in the closed region bounded by the tree paths from s to x and to u, along with the path from u to x. Any path from y to v must cross the boundary of this region, which would create a directed cycle, contrary to the fact that G is a DAG.

Now assume that we have a path from u to v via a useless edge (x, y), where x is to the left of v. Either this path intersects the tree path from s to v or it doesn't. In the former case, we clearly do not need the edge (x, y). In the latter case, v is in the bounded region enclosed by the tree paths from s to x and u, along with the path from u to x. Since y is to the left of x (by the definition of uselessness), any path from y to v must cross the boundary of this region, again creating a cycle. ☐

In logspace we can detect and remove useless edges; we therefore assume that G has no useless edges. Note also 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.

We need to define some basic search routines.

Definition 5.8. *Given an* SMPD *G and a vertex x, let* **ReachLocal(x)** *be the set of vertices reachable from x using only tree edges and local edges.*

Lemma 5.9. *The predicate $y \in$* **ReachLocal(x)** *is in* L.

Proof. (of Lemma 5.9) Consider the induced subgraph $G'(x)$ on the vertices in **ReachLocal(x)**. Since there are no jump edges, all the sinks in $G'(x)$ lie on the external face (by appealing to Observation 1). Construct a new graph $G''(x)$ by adding a sink to $G'(x)$ along with an edge from each old sink to this new sink. Clearly $G''(x)$ is an SSPD and we are done by an application of Theorem 5.4. ☐

An immediate consequence, which we record for future reference, is the following:

Corollary 5.10. *Given vertex x, the vertices in* **ReachLocal(x)** *with the least finishing time and maximum discovery time (relative to the original spanning tree of the graph) can be found in* L. *Let's call these vertices* **ReachLeft(x)** *and* **ReachRight(x)** *respectively.*

Our basic strategy is as follows. Start at u (on the right side of the graph) and w (on the left side of the graph) and do local searches. The goal vertex v is thus "squeezed" between some areas where we were able to do some searching. We will make use of the procedures **LeftwardSearch** and **RightwardSearch** to make limited use of jump edges to further restrict the area where v can try to hide. When these procedures no longer admit any progress, then we make stronger use of jump edges that "tunnel" from one side of the graph, below v, over to the other side, to take even more hiding room away from v. Below, we define these procedures more precisely, and then we show that the algorithm works.

The procedure **LeftwardSearch** starts at a given vertex and does a local search, updating **Limright** to mark the right boundary of the area where v can still be hiding. Then it looks for a jump edge that stays on the right side of v and advances as little as possible beyond **Limright**, and repeats the process until no more progress can be made.

```
LeftwardSearch($z$)
    while true
        do
            Enumerate ReachLocal($z$).
            Limright $\leftarrow$ ReachLeft($z$)
            $S \leftarrow \{(x, y) : (x, y)$ is a jump edge with
                    $x$ to the right of Limright and
                    $y$ to the left of Limright and
                            to the right of $v\}$
            if $S$ is not empty
              then pick $(x, y) \in S$ such that
                    $y$ is the furthest right
                    (i.e., as close as possible to Limright),
                    breaking ties by picking $y$
                    as close to the root $s$ as possible,
                    $z \leftarrow y$
              else  return
```

RightwardSearch is defined symmetrically. The procedure **Tunnel** looks for jump edges in S_r (jump edges that tunnel from the right side of the graph, below v, to the area just right of **Limleft**) or in a similarly-defined set S_l. (It is easy to see that at least one of S_l and S_r will always be empty, by planarity.)

```
Tunnel()
    $S_r \leftarrow \{(x, y) : (x, y)$ is a jump edge with
            $x$ to the right of Limright and
            $y$ to the left of $v$ and
                    to the right of Limleft.
    $S_l \leftarrow \{(x, y) : (x, y)$ is a jump edge with
            $x$ to the left of Limleft and
            $y$ to the right of $v$ and
                    to the left of Limright.$\}$
    if $S_r \cup S_l$ is empty,
       then Direction $\leftarrow$ Nil

    if $S_r$ is not empty
       then Direction $\leftarrow$ Right
            Pick $(x, y)$ in $S_r$ with
            $y$ as far left as possible
            (i.e., as close as possible to Limleft,
            breaking ties by picking the
            vertex closer to the root)
            Target $\leftarrow y$

    if $S_l$ is not empty,
       then Direction $\leftarrow$ Left
            Pick $(x, y)$ in $S_l$ with
            $y$ as far right as possible
            (i.e., as close as possible to Limright,
            breaking ties by picking the
            vertex closer to the root)
            Target $\leftarrow y$
```

We now present an algorithm to enumerate vertices that are reachable from u. The vertex v is reachable from u if and only if it ever shows up in the enumeration.

```
begin
    LeftwardSearch($u$)
    RightwardSearch($w$)
    Repeat
        Tunnel
        If Direction $=$ Left then
            LeftwardSearch(Target)
        If Direction $=$ Right then
            RightwardSearch(Target)
    until Direction $=$ Nil
end
```

In order to argue that the algorithm is correct, we will establish the following invariant condition: Each time **Limright** or **Limleft** is updated, if z is to the right of **Limright** or to the left of **Limleft**, then there is a path from u to z iff z has been enumerated, and any jump edge that is ever in one of the sets S, S_l, S_r is reachable from u.

Limright is updated only by **LeftwardSearch**, and it always occurs immediately after execution of **ReachLocal(**z**)** as the first step of an instantiation of **LeftwardSearch**(z). The first time this happens is for $z = u$, and in this case **Limright** is set to **ReachLeft(**u**)**. It is easy to see that all vertices to the right of **ReachLeft(**u**)** are enumerated in **ReachLocal(**u**)** and all are reachable; this establishes the basis of our induction for **Limright**, and an even easier argument establishes the basis for **Limleft**. Also, this directly implies that the first time the set S is considered in **LeftwardSearch**, or **RightwardSearch**, all of the relevant jump edges are reachable from u. Similarly, if **Tunnel** is called before **Limright** or **Limleft** is updated again, then we immediately have that the same is true for all jump edges in S_l and S_r (and in this case, S_l is empty).

For the inductive step, consider first the case where **Limright** is updated after executing another round of the loop in **LeftwardSearch**. Thus we have just enumerated **ReachLocal(**y**)** for a jump edge (x, y). By the inductive hypothesis, all of these enumerated vertices are reachable from u, since the jump edge is reachable. Thus if the inductive step were to fail, there must be some vertex z' to the right of **ReachLeft(**y**)**, that has not been enumerated but is reachable. By the inductive hypothesis, it must be to the right of **ReachLocal(**y**)** and to the left of the old value of **Limright**. Consider the first edge on the path from u to z' that is to the left of the old value of **Limright** and to the right of **ReachLocal(**y**)**. This edge cannot be a local edge or tree edge (because the predecessor is enumerated by hypothesis, and the enumeration follows such edges). Thus it must be a jump edge. But by the way that we select jump edges, it would have been chosen, instead of (x, y). Thus z' cannot exist.

It remains only to consider the case where **Limright** is updated after executing **Tunnel**. Thus we have just enumerated **ReachLocal(**y**)** for a jump edge (x, y) where x is to the left of v. By hypothesis, x is reachable, and thus all of the enumerated vertices are reachable from u. Thus as in the previous case, if the inductive step were to fail, there must be some vertex z' to the right of **ReachLocal(**y**)** and to the left of the old value of **Limright** that has not been enumerated but is reachable. Consider the first edge on the path from u to z' that is to the left of the old value of **Limright** and to the right of **ReachLocal(**y**)**. This edge cannot be a local edge or a tree edge; thus it must be a jump edge. But **Tunnel** would not have been called if there had been such a jump edge coming from the right, and if this jump edge were to come from the left, then it would have been chosen, instead of (x, y). Thus z' cannot exist.

A similar argument holds for **Limleft**. It remains only to show that the jump edges in S, S_l, and S_r are reachable. By induction hypothesis, the jump edges that start to the right of the old value of **Limright** are reachable, as are any jump edges that start from **ReachLocal(**y**)** for the vertex y that was selected when **Limright** was updated most recently. Let e be the jump edge (x, y) that was selected when this update happened. If the inductive hypothesis were to fail, there would have to be a jump edge e' departing between the old value of **Limright** and **ReachLocal(**y**)**. If e is directed from right to left, then it encloses the region where e' would begin, which means that e' would not be in S, S_l, or S_r. Thus we must have e directed from left to right. But then e' would have been selected during the previous execution of **LeftwardSearch**, which is contrary to the choice of e'.

We have now established the invariant condition. To see that this implies correctness, assume that v is is reachable from u but is not enumerated. Consider the first edge $e = (x, y)$ on this path from u such that y is not enumerated by the time that the procedure halts. By the invariant condition, y cannot be to the right of the final value of **Limright** or to the left of the final value of **Limleft**, whereas x is to the right of **Limright** or to the left of **Limleft**. Clearly, e cannot be a local edge or tree edge, and thus it is a jump edge. However, if such a jump edge had existed, then the procedure would not have stopped at the given values of **Limleft** and **Limright**. $\square$

5.3 Recognition of SSPDs

We prove:

Theorem 5.11. *Recognition of* SSPDs *can be done in* L.

In order to prove this, we use the following:

Lemma 5.12. *Given a planar graph G such that only its external face (and no other face) is a directed cycle, there has to be a source or a sink somewhere inside the graph G.*

Proof. (of Lemma 5.12) We argue by contradiction. Let G be a graph with no sources or sinks, with a given planar embedding such that its external face is a directed cycle and no other face is a cycle. Then 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, so 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 $v_1, v_2, \cdots, v_k$ 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 $v_1, v_2, \cdots, v_k$ meet 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. □

Proof. (of Theorem 5.11) 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?

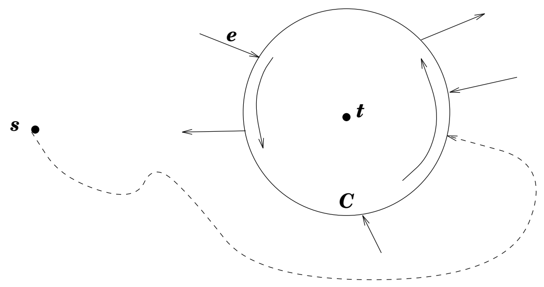

Figure 9. t **inside cycle** C, s **outside**

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}?

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

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

Clearly if G is a SSPD, then by Fact 1, 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. By Lemma 5.12, 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 9.

Consider all the edges from outside C that are incoming to some vertex on C (for instance, edge e in Figure 9). Suppose the cycle C were as directed as in Figure 9, then in the step 4 where all 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 there is no directed path from s to t in G_{left}, because, similarly to [1], we now have a *geometric cut* consisting of only the remaining outgoing edges from C to the outside of C - or a simple argument - if there is a path from s to t now, that path intersects C at some place, and it can only be directed towards C. But we deleted all of these incoming edges in constructing G_{left}. Thus, we end up with an indegree-1 graph in which there is no path from s to t.

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 5.13. *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* SSGGR*) is in* $\mathsf{FO} + \mathrm{UGGR}$*.*

Proof. We need only examine the five steps in the proof of Theorem 5.11. 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 *in-degree* one, which are easily converted to 1GGR questions on that graph's reversal. □

5.4 Recognition of SMPDs

Theorem 5.14. *Recognition of* SMPD*s can be done in* L.

Proof. (of Theorem 5.14) We perform the following tests:

1. We first check if the given graph G is planar, and if so, find a planar embedding of G [2].

2. Check if 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 if 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 if H is a directed tree. If not, return "false".

   Suppose H is a directed tree - H clearly inherits its embedding from G. We assume that we are given a dfs 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 **Finish(a)** $>$ **Finish(b)**).
   - **Type II** edges are those going left-to-right (i.e. cross edges (a, b) where **Finish(a)** $<$ **Finish(b)**).

4. Now, we check if G with the underlying spanning tree H has any back edge. If so, we have clearly found a cycle, so G is not a 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 either of G' and G'' are SMPDs (because any cycle in the tree 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 if 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 if 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 a 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 a SMPD. For this purpose, we introduce the following terminology

Definition 5.15. *A (directed) cycle is* minimal *if the set of cross edges contained in it is minimal w.r.t. inclusion.*

A directed chord in a cycle all of whose edges are tree edges, will be called a tree chord.

It is easy to see the following:

Lemma 5.16. *A cycle is not minimal if it has a tree chord.*

We use the above lemma to prove:

Lemma 5.17. *Any minimal cycle either contains exactly one edge of Type I or contains exactly one edge of Type II.*

Proof. (of Lemma 5.17) 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 5.16.

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 10. 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. Likewise 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.

□

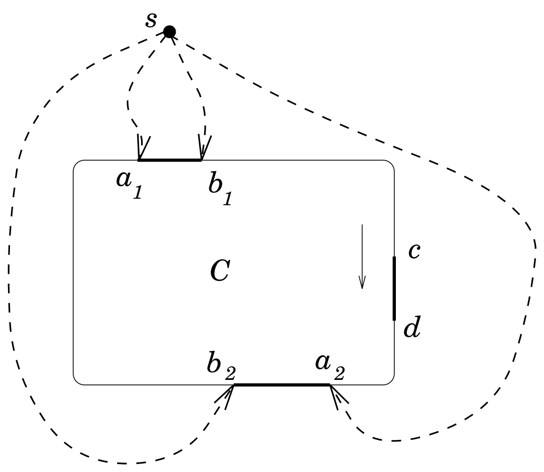

Figure 10. Tree-paths to edges around C

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 5.17, and we discover such a minimal cycle in Test 5. □

5.5 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 essential for being able to perform the above tasks in L? We show in the following that we can solve reachability questions, even when the graph has a few cycles, in L.

Consider the class $\mathcal{G}$ of graphs that are planar, have a single source and a single sink, and 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 5.18. *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. (of Theorem 5.18) Given a planar graph G with a unique source s and sink t, and no facial cycles, by appealing to Lemma 5.12, we know that any possible cycle has to contain at least a source or a sink inside. Now a potential cycle C cannot contain both s and t inside, because then the outside of the cycle C violates Lemma 5.12. So the only possibility is that C

separates s from t, *i.e.* without loss of generality, we can assume that t lies inside C and s outside.

Now we proceed to reduce reachability questions in G to a reachability question in a 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 [1], by cutting along the path between s and t. As in [1], 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 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 a 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. □

Theorem 5.19. *Reachability questions in outerplanar digraphs can be solved in* L*.*

Note that outerplanar digraphs, even DAGs, are *not* series-parallel digraphs as considered by [11]. 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 a SMPD.

In the language of book embeddings (see [17] 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. (of Theorem 5.19) 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 11).

Here, the graph G is not acyclic. The instance to the reachability question is (G, u, v) and we are to find if 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}.

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

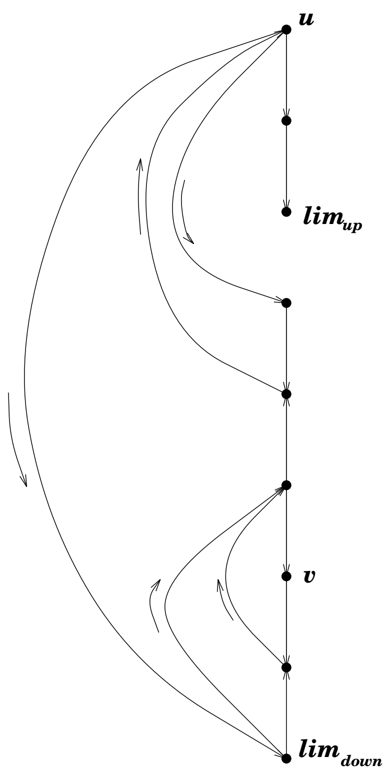

Figure 11. A 1-page embedding

as follows:

1. Initialize the markers as $lim_{up} = u, lim_{down} = \infty$.

2. Go down from lim_{up} as far as one can using only ordinary edges. Go up from lim_{down} as far as one 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 any more and we haven't discovered v as yet, 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. For this, we use 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. $\square$

6 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 12.

Are these problems and classes interesting? We argue, particularly in Section 2.4, 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

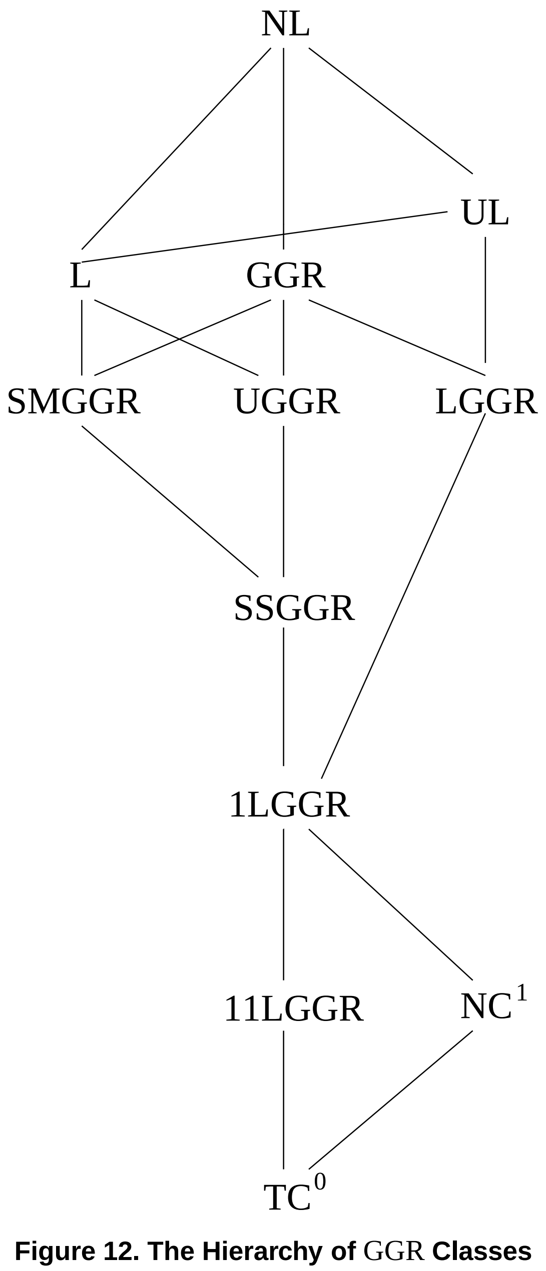

Figure 12. The Hierarchy of GGR Classes

the SMGGR problem reducible to UGGR? The proof of Theorem 5.7, 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 $\mathsf{FO} + \mathrm{UGGR}$ – 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.* ([11]) 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. We note that reachability in an acyclic graph with constantly many sources is in L by an easy extension of our methods. Also, while we can easily show that reachability questions in SSPDs reduce to the complement, we are not able to show the same for SMPDs.

Acknowledgments

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.

References

[1] 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. Springer, 2005. to appear.

[2] Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189:117–134, 2004.

[3] David 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.

[4] David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven 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.

[5] Manuel Blum and Dexter 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.

[6] S. Buss. Polynomial-sise frege and resolution proofs of st-connectivity and Hex tautologies. submitted for publication, manuscript available from `http://www.math.ucsd.edu/ sbuss`., 2005.

[7] Kousha Etessami. Counting quantifi ers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, Jun 1997.

[8] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[9] Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16(4):760–778, 1987.

[10] Neil Immerman. *Descriptive Complexity*. Springer Graduate Texts in Computer Science, 1998.

[11] A. Jakoby, M. Liskiewicz, and R. Reischuk. Space effi - cient algorithms for series-parallel graphs. In *18th International Symposium on Theoretical Aspects of Computer Science (STACS)*, number 2010 in Lecture Notes in Computer Science, pages 339–352. Springer, 2001. To appear in J. Algorithms.

[12] 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. Springer, 2006. to appear.

[13] O. Reingold. Undirected st-connectivity in log-space. In *Proceedings 37th Symposium on Foundations of Computer Science*, pages 376–385. IEEE Computer Society Press, 2005.

[14] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal of Computing*, 29:1118–1131, 2000.

[15] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[16] 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.

[17] M. Yannakakis. Four pages are necessary and sufficient for planar graphs. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 104–108, New York, NY, USA, 1986. ACM Press.