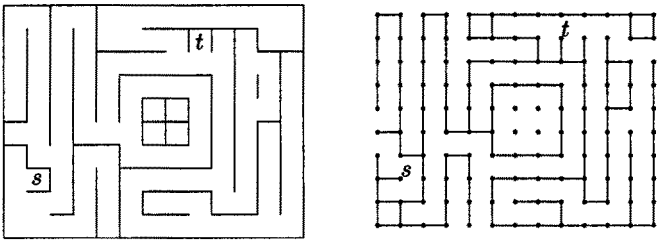
Searching Constant Width Mazes Captures the AC^0 Hierarchy

David A. Mix Barrington[1] Chi-Jen Lu[1] Peter Bro Miltersen[2*] Sven Skyum[2*]

[1] Computer Science Department, University of Massachusetts.
[2] BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus.

Abstract. We show that searching a width k maze is complete for Π_k, i.e., for the k'th level of the AC^0 hierarchy. Equivalently, st-connectivity for width k grid graphs is complete for Π_k. As an application, we show that there is a data structure solving dynamic st-connectivity for constant width grid graphs with time bound $O(\log \log n)$ per operation on a random access machine. The dynamic algorithm is derived from the parallel one in an indirect way using algebraic tools.

1 Introduction



(a) (b)

Fig. 1. (a) A maze and (b) the corresponding grid graph

Blum and Kozen [4] considered the problem of searching a *maze*. A maze is an object as depicted in Figure 1(a). Formally, we will define a maze of width m and length n as follows: Let $S_{n,m} = \{1, \ldots, n\} \times \{1, \ldots, m\}$. We call an element s of $S_{n,m}$ a *square* and identify s with the unit square with center s in the plane. A maze of width m and length n is a set M of line segments (*walls*) of length exactly 1, each separating two squares of $S_{n,m}$. Figure 1(a) depicts a maze of width 10 and length 13 (we consider the longer line segments as consisting of several atomic walls of length 1). A path in the maze between two squares s and t is a path inside the rectangle $[0, n] \times [0, m]$ connecting the centers of s and t and not intersecting any of the walls in M. The reader is invited to verify that there is a path between s and t in Figure 1(a). Blum and Kozen gave bounds on the power of systems of automata capable of searching a maze, i.e. capable of deciding whether a path between two given squares in the maze exists. In complexity theoretic terms, one of their results was that searching a maze is in deterministic logspace.

* Supported by the ESPRIT Long Term Research Programme of the EU under project number 20244 (ALCOM-IT).

In this paper we consider the complexity of searching a *constant width* maze, i.e., rather than letting both n and m be parameters, we fix m to a constant $k \geq 1$. Let MAZE_k be the problem which takes as input (a Boolean encoding of) a maze of width k, two squares s and t, and decides if there is a path from s to t.

We relate the complexity of MAZE_k in a strong way to the levels of the AC^0 hierarchy. Recall the following definitions: Non-uniform AC^0 is the class of languages recognizable by families of AND/OR/NOT-circuits of constant depth, polynomial size, and unbounded fan-in. Inside AC^0 we find the following hierarchy: Non-uniform Σ_k is the class of languages recognizable by circuits with k alternating levels of unbounded fan-in AND and OR gates, with the output an OR-gate and a "zeroth level" of input gates and their negations. Non-uniform Π_k is defined analogously, but with the output gate being an AND-gate. Following [1], we define a uniform version of the hiearchy as follows: Uniform Π_k (Σ_k) is the class of languages accepted by alternating Turing machines running in logarithmic time and making exactly k alternations, the first being universal (existential).

An appropriate class of reductions to use for the non-uniform classes in the AC^0 hierarchy is the class of (non-uniform) p-projections [15]. Similarly, an appropriate class of reductions to use for the uniform classes is the class of DLOGTIME-uniform projections (for a precise definition, see Section 2).

Our main result is:

Theorem 1. *For $k \geq 3$, MAZE_k is complete for non-uniform Π_k under p-projections and complete for uniform Π_k under DLOGTIME-uniform projections.*

This seems to be the first example of natural graph problems, complete for the levels of the AC^0 hierarchy.

There is a close correspondence between mazes and *grid graphs*, as defined by Itai *et al.* [11]. An $n \times k$ grid graph is an undirected graph G with vertex set $V_{n,k} = \{1, \ldots, n\} \times \{1, \ldots, k\}$ and with the property that if $\{(a, b), (c, d)\}$ is an edge in G, we have $|a - c| + |b - d| = 1$. The length of the grid graph is n and the width is k. A grid graph is shown in Figure 1(b). The st-connectivity problem USTCON_k for width k grid graphs is the following: Given a grid graph, and two vertices s and t, decide if s and t are connected in G. There is a trivial isomorphism between MAZE_k and USTCON_k: To get from a maze problem to a grid graph problem, simply make a vertex for each square of the maze, and put an edge between two vertices if and only if there is *not* a wall between the corresponding squares. We shall use the grid graph formulation in the main part of the paper.

A third setting for these problems is the following variant of bounded-width branching programs. An $n \times k$ *switching network* is an *undirected* labelled graph whose vertices form a rectangular array with k rows and n columns and whose edges are restricted to be between vertices in adjacent columns. (Switching networks are also called "contact schemes" — see the survey of Razborov [13] for further background.) Each edge is labelled by an input variable, its negation, or

the value 1, and the network accepts a given input string iff there is a path from a fixed vertex s to another fixed vertex t such that the label of each edge on the path evaluates to 1 on the input. It is not hard to show that the grid graph problem is closely related to *planar* switching networks as follows: USTCON$_k$ is complete, under p-projections, for the class of languages decidable by families of width-k, polynomial-size planar switching networks. This is because an $n \times k$ planar switching network can be simulated by a $kn \times k$ grid graph, and an $n \times k$ grid graph can be simulated by a $kn \times k$ planar switching network. We omit the details of these simulations in this version of the paper.

In our second result, we consider the following *dynamic* graph problem: Maintain, on a random access machine with word size $O(\log n)$, a data structure representing an $n \times k$ grid graph under insertions and deletions of edges and connectivity queries, i.e. queries asking whether there is a path between two vertices, given as input. For *non-constant* width $m \leq n$, Eppstein *et al* provide a solution to this problem with a time bound $O(\log n)$ per operation [6]. We show:

Theorem 2. *For any constant k, there is a solution to the dynamic connectivity problem for width k grid graphs with time complexity $O(\log \log n)$ per operation. On the other hand, no solution to the dynamic connectivity problem for width 2 grid graphs has time complexity $o(\log \log n / \log \log \log n)$.*

We derive the dynamic algorithm from the parallel one in an indirect way: We note that the *existence* of the parallel algorithm implies that a certain monoid, G_k, associated with the width-k problem is aperiodic by results of Barrington and Therien [2]. Combining this with a result of Thomas [16], we in fact show that G_k has dot-depth exactly k, providing a natural example of such a monoid which may be of independent interest. We then use results on dynamic word problems by Frandsen, Miltersen and Skyum [7] to derive the dynamic algorithm. Unfortunately, the algorithm obtained is rather impractical; the constant in the big-O is $2^{2^{O(k)}}$. The lower bound we get as a corollary to work of Beame and Fich [3], again by looking at the problem from an algebraic point of view.

2 The completeness result

An instance of the USTCON$_k$ problem consists of a grid graph, a vertex s and a vertex t. We represent the graph by a number of Boolean *edge indicator variables*, one for each edge position in the grid. We pack them in two binary relations, $E_h \subseteq \{1, \ldots, n-1\} \times \{1, \ldots, k\}$ representing the horizontal edges; $E_h(i,j)$ is true if and only if there is an edge between (i,j) and $(i+1,j)$, and $E_v \subseteq \{1, \ldots, n\} \times \{1, \ldots, k-1\}$; $E_v(i,j)$ is true if and only if there is an edge between (i,j) and $(i,j+1)$. The vertices s and t are represented by two indicator variables for each vertex v, one which is true iff $v = s$ and one which is true iff $v = t$. With the encoding made clear, we can now show the non-uniform part of Theorem 1.

Lemma 3. USTCON$_k$ *is Π_k, for $k \geq 3$. The constructed circuit is positive (monotone) in the edge variables.*

Proof. We first show that for all $k \geq 1$, the statement "there is a path from vertex s to vertex t in G" for *fixed boundary* vertices s and t can be computed by a positive Π_k circuit; that is, we do not let s and t be part of the input and we assume them to be on the boundary of the grid. Then, we generalize, first to the case of non-boundary vertices, and then to s and t being given as input.

Base, $k = 1$: There is a path between s and t if and only if, for all a, if a is an edge position between s and t, a is an edge. This is a Π_1 statement in the edge indicator variables, as desired.

Now suppose $k > 1$. Given a grid graph G on $V_{n,k}$, we define its *dual* G^* as follows: G^* has a vertex s^* for each square s of the grid and a vertex ∞ representing the region outside the grid. We put an edge between two vertices u^* and v^* of G^* if and only if the edge position separating u and v in G is *not* an edge. Thus, for every edge position e of G there is an edge position e^* of G^* and exactly one of G or G^* has an edge at that position. G and G^* can be simultaneously embedded in the plane. Note that $G^* - \{\infty\}$ is a grid graph on $V_{n-1,k-1}$.

Now, for any given vertices s and t of G, there is a path between s and t in G if and only if there is *not* a simple cycle in G^* so that if the cycle is drawn in the plane, s is on the outside of the cycle and t is on the inside of the cycle. Since s and t are border vertices, such a cycle must go through the vertex ∞. Let C be some cycle going through ∞ and let e_1^* and e_2^* be the two edges adjacent to ∞ on the cycle. C separates s and t if and only the edge positions e_1 and e_2 separate s and t in the following sense: If one tracks the border clockwise from s back to itself, one of e_1 and e_2 is found before hitting t and the other is found after.

Thus, there is a path from s to t if and only if for all border edge positions e_1 and e_2, such that e_1 and e_2 separate s and t, there is *not* a path in $G^* - \{\infty\}$ from u^* to v^*, where u is the square of G adjacent to e_1 and v is the square of G adjacent to e_2.

The statement "e_1 and e_2 separate s and t" is independent of the input. Since $G^* - \{\infty\}$ is a grid graph of width $k-1$ there is a positive Π_{k-1} circuit deciding whether a path between two fixed border vertices exists. Note that the inputs of this circuit are edge indicators for G^*, i.e. negations of edge indicators for G. Thus, using DeMorgan's law, checking whether *no* path between two fixed border vertices exists can be done by a positive Σ_k circuit in the primal edge indicator variables. We conclude that the validity of the entire statement can be checked by a positive Π_k circuit, as desired.

Now consider the more general problem, where s and t are not on the boundary, but still fixed. Assume without loss of generality that s is to the left of t or immediately above t. Split the graph into 3 parts — the part left of s, the part between s and t, and the part right of t. Compute the transitive closure for each component, restricted to the vertical border vertices. By the above, this can be done by $O(k^2)$ Π_k circuits, i.e. a constant number. The end result is now a monotone Boolean function of the computed information. Since the amount of information is constant, we can compute this function with a positive NC^0

circuit. Since Π_k is closed under positive finite Boolean combinations, the entire thing is Π_k.

Finally, consider the USTCON$_k$ problem with s and t being part of the input. Recall that they are given by two indicator variables for each vertex. For each value of s and t we can construct a gate $E_{s,t}$ which evaluates to 1 if and only if the inputs are s and t; this gate is just an AND of two indicator variables. For each possible value of (s,t), construct the Π_k circuit $C_{s,t}$ solving the problem for this value. Now, we adjust $C_{s,t}$ so that it outputs 1, if s or t do not match the actual input. We do this by giving each of the OR gates of the second layer from the top of $C_{s,t}$ one additional input, namely the negation of $E_{s,t}$. The end result is the AND of all these adjusted $C_{s,t}$ circuits. There is no penalty in depth if $k \geq 3$. Note that the final circuit is no longer positive, but the only negative literals are these $E_{s,t}$'s.

Lemma 4. *For $k \geq 1$, every problem in non-uniform Π_k reduces to* USTCON$_k$ *by a non-uniform p-projection.*

Proof. We will show the following stronger statement: For every $k \geq 1$, every problem in non-uniform Π_k reduces to USTCON$_k$ and every problem in non-uniform Σ_k reduces to USTCON$_{k+1}$ by p-projections. Furthermore, the value of the node s in the reduction is the bottommost left corner of the grid and the value of the node t in the reduction is the bottommost right corner of the grid.

Given a Π_k circuit of size s, we can construct a Π_k formula of size $s^{O(1)}$ computing the same function, so we can assume without loss of generality that we are given a function which can be computed by a Π_k formula. By the definition of p-projection [15], an alternative formulation of the statement is then this:

Given a Π_k formula C, we can construct a polynomial sized, width k grid graph $G(C)$ where some of the edges are labelled with input variables or their negations, the bottommost left corner of the grid is labelled s and the bottommost right corner of the grid is labelled t, so that, given an input vector $\mathbf{x}$, if we remove the edges labelled with variables assigned 0, there is a path form s to t in $G(C)$ if and only if $C(\mathbf{x})$ evaluates to true. Similarly, given a Σ_k circuit, we can construct a width $k+1$ grid graph with corresponding properties. We will construct this mapping G by recursion in k.

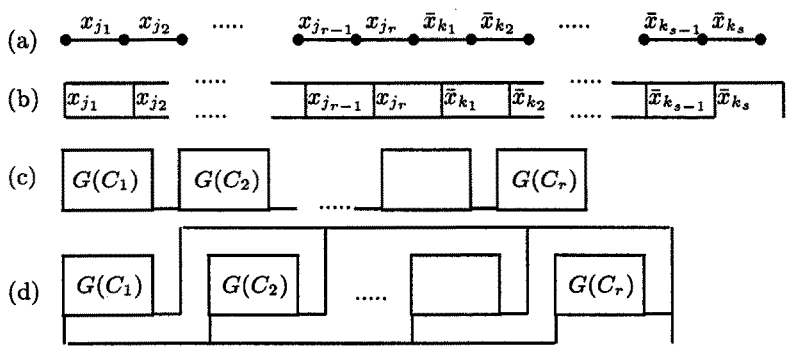


Fig. 2. The reduction

First suppose a Π_1 formula C is given. We can write C as $\bigwedge_{i=1}^{r} x_{j_i} \wedge \bigwedge_{i=1}^{s} \bar{x}_{k_i}$, where $x_{j_i}, i = 1 \ldots r$ and $x_{j_i}, i = 1 \ldots s$ are input variables. The corresponding width 1 grid graph $G(C)$ is shown in figure 2(a). Similarly, if a Σ_1 formula C is given, we write C as $\bigvee_{i=1}^{r} x_{j_i} \vee \bigvee_{i=1}^{s} \bar{x}_{k_i}$ and let $G(C)$ be the width 2 grid graph of Figure 2(b).

Now, let $k > 1$ and assume we have both the Σ_j and Π_j constructions for all $j < k$. Let a Π_k formula C be given. We can write it as $\bigwedge_{i=1}^{r} C_i$, where the C_i's are Σ_{k-1} formulae. Construct the width k graphs $G(C_i)$ corresponding to the C_i's and let $G(C)$ be the graph of Figure 2(c). Note that this graph also has width k, as desired. Finally, let a Σ_k formula C be given. Write it as $\bigvee_{i=1}^{r} C_i$, where the C_i's are Π_{k-1} formulae. Construct the width $k-1$ graphs $G(C_i)$ corresponding to the C_i's and let $G(C)$ be the width $k+1$ graph of Figure 2(d).

The correctness of the construction is easily checked.

Lemma 3 and Lemma 4 together gives us the non-uniform part of Theorem 1. We now make precise the content of the uniform part.

As in Barrington, Immerman, and Straubing [1], we define a *log-time* Turing machine to have a read-only input tape of length n, a constant number of read-write work tapes of total length $O(\log n)$, and a read-write input address tape of length $\log n$. On a given time step the machine has access to the bit of the input tape denoted by the contents of the address tape (or to the fact that there is no such bit, if the address tape holds too large a number). An *alternating* log-time machine has universal and existential states with the usual semantics. Furthermore, the alternating machine queries its input only once in a computation, in its last step. We now define uniform Π_k as the class of languages accepted by alternating log-time Turing machines, making exactly k alternations, the first being universal. It is shown in [1] that this hierarchy is in fact a uniform version of the AC^0 circuit hierarchy, where specific questions about the circuit family can be answered by a log-time Turing machine.

Recall that a family of p-projections can be viewed syntactically as a family of maps $\sigma_n : \{y_1, y_2, \ldots, y_{m(n)}\} \to \{0, 1, x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$, where $m(n)$ is bounded by a polynomial in n. A DLOGTIME-uniform projection is a family of projections σ_n, so that there is a log-time Turing machine which on input $\langle i, 1^n \rangle$ outputs the binary encoding of the values of $m(n)$ and $\sigma_n(y_i)$ on a specified work tape. It is easy to see that DLOGTIME-uniform projections are closed under composition and that uniform Π_k and Σ_k are closed under DLOGTIME-uniform projections. To obtain the uniform part of Theorem 1, we essentially now have to redo Lemma 3 and Lemma 4, checking that they are valid in the uniform setting. Due to lack of space, this is omitted from this version of the paper.

3 The width-k grid graph monoid

In this section we consider an algebraic interpretation of our results. We explore its consequences in Section 4, but we also consider it interesting in its own right.

We define the width k grid graph monoid G_k. It will be a submonoid of the following monoid M_k: The ground set of M_k is the set of equivalence relations

on $V_{2,k} = \{1,2\} \times \{1,\ldots,k\}$, or, equivalently, the set of transitively closed undirected graphs with vertex set $V_{2,k}$. Let G and H be members of M_k, viewed as graphs. We now define the composition of G and H. Let $U = \{1, \frac{3}{2}, 2\} \times \{1, ..., k\}$. Let R be the graph on U obtained by embedding G in U by the embedding $(1,y) \to (1,y), (2,y) \to (\frac{3}{2},y)$ and embedding H in U by the embedding $(1,y) \to (\frac{3}{2},y), (2,y) \to (2,y)$. Let R^* be the transitive closure of R. The composition $G \circ H$ is the restriction of R^* to $V_{2,k} = \{1,2\} \times \{1, ..., k\}$. G_k is now defined to be the submonoid of M_k generated by the transitive closures of the set of grid graphs on $V_{2,k}$.

A very intuitive way of viewing G_k is as follows. An element of G_k is a collection of plane blobs inside the $[1,2] \times [1,k]$ rectangle in the plane, where a blob is identified with the set of grid points it contains. In Figure 3, (a) are (b) are two such elements. To multiply two elements, we concatenate them and scale down the resulting picture by a factor of two on the x-axis. In Figure 3, (a) and (b) are concatenated to form (c) and then scaled down to (d). Finally, since two blobs are equivalent if they contain the same elements, we can make a nicer picture (e) which is equivalent to (d).

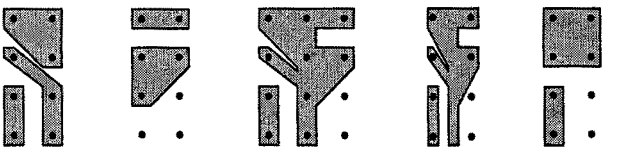


Fig. 3. Elements in G_k and their product

Note that every member of G_k can be described as a word $a_1 \circ a_2 \circ \ldots a_{f(k)}$ where the a_i's are closures of $2 \times k$ grid graphs for some function f (a trivial upper bound on $f(k)$ is $|G_k|$). Another way of viewing this: Every member of G_k can be described by the transitive closure of an $f(k) \times k$ grid graph, restricted to the vertical border vertices.

The size of G_k is $c_{2k} = \frac{1}{2k+1}\binom{4k}{2k}$, i.e. the $2k$'th Catalan number. We omit the proof of this in this version of the paper. Our earlier analysis of grid graphs can now be interpreted in terms of the algebraic structure of G_k. To use the vocabulary of formal language theory, we may think of a grid graph problem as a string where the individual letters are elements of G_k. The following result tells us something about the language of strings representing graphs with a particular connectivity property. It is star-free (meaning that it can be formed from one-letter languages by concatenation and boolean operations including complementation), and has dot-depth k (meaning that the optimal depth of nesting of concatenation operations is k). See, for example, [12] for further background on algebraic automata theory.

Lemma 5. *G_k is an aperiodic monoid with dot-depth exactly k.*

Proof. (sketch) We first show that the dot-depth is at most k. In our proof of Theorem 1, we essentially showed that connectivity in a width-k grid graph

could be expressed by a logical formula with k quantifiers in prenex normal form, and atomic predicates that either referenced individual edges (properties of individual "letters" in the input string) or compared two column numbers (positions of letters in the string). By a theorem of Thomas [16], any language so describable has a syntactic monoid that is aperiodic with dot-depth k. But this syntactic monoid is G_k itself, since G_k was designed to exactly capture this connectivity information.

If the dot-depth of G_k were less than k, we could derive a contradiction as follows. Consider any circuit C of depth k and size s. By our construction in Lemma 4, we can construct a word over G_k, of size polynomial in s, whose product determines the value of C. But if G_k has dot-depth $k-1$ or less, this product can be evaluated by a circuit of depth $k-1$ and size polynomial in s, using a construction of Barrington and Thérien [2]. Since C was arbitrary, we have collapsed two distinct levels of the AC^0 hierarchy, contradicting a theorem of Sipser [14].

In Section 4, we only use the aperiodicity of G_k, not its dot-depth. Still, G_k gives us an example of a natural (or at least easily visualizable) monoid which we know is aperiodic with dot depth exactly k — such examples are rare.

4 The dynamic grid graph connectivity problem

We consider the following *dynamic* graph problem: Maintain, on a random access machine with word size $O(\log n)$, a data structure representing an $n \times k$ grid graph under insertions and deletions of edges and connectivity queries, i.e. queries asking whether there is a path between two given vertices.

We are now ready to show Theorem 2 from the introduction. For the upper bound, we shall use a result of Frandsen, Miltersen, and Skyum [7]. Let S be a finite monoid. The dynamic range query problem for S is the problem of maintaining, on a random access machine with word size $O(\log n)$, a sequence $(a_1, a_2, \ldots, a_n) \in S^n$ under a change(i,b)-operation which changes a_i to $b \in S$, and an operation query(i,j) which returns $a_i \circ a_{i+1} \circ \cdots \circ a_{j-1} \circ a_j$. Frandsen, Miltersen, and Skyum show that if S is aperiodic, there is a solution to the dynamic range query problem for S with time bound $O(\log \log n)$ per operation. The constant in the big-O is $2^{O(|S|)}$.

We show that dynamic reachability reduces to dynamic range queries over G_k with a constant overhead. By Lemma 5, we have shown the upper bound of Theorem 2. Suppose we are to maintain a graph on $\{1,..,n\} \times \{1,...,k\}$. We maintain the product $a_1 \circ a_2 \circ ... \circ a_{n-1}$ where a_i is the element of G_k correponding to the subgraph in $\{i, i+1\} \times \{1,...,k\}$. A change in the graph corresponds to a single change of an a_i. If we are to answer if there is a path from (x_1, y_1) to (x_2, y_2) we do the following (assuming $x_1 < x_2$): We query the subproducts $a = a_1 \circ ... \circ a_{x_1 - 1}$, $b = a_{x_1} \circ ... \circ a_{x_2}$ and $c = a_{x_2+1} \circ ... \circ a_n$. Whether or not there is a path between (x_1, y_1) and (x_2, y_2) is completely determined by (a, b, c, y_1, y_2), and since these are all in a constant range, we can hardwire the answer for each possible value of the tuple into the algorithm.

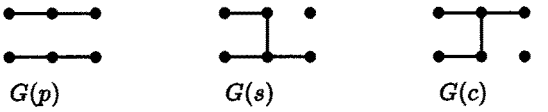


Fig. 4. Gadgets for reducing the dynamic prefix problem for L to dynamic connectivity

We now show the lower bound part of Theorem 2. We shall use a result by Beame and Fich [3]. A regular language L is called *indecisive* if and only if for all strings x, there exist z and z' such that $xz \in L$ and $xz' \notin L$. Given a language L over an alphabet Σ, the dynamic prefix problem for L is the problem of maintaining a string $x = x_1 \dots x_n \in \Sigma^n$ under a change(i, a) operation which changes x_i to a and a prefix(j) operation which answers the question "Is $x_1 x_2 \dots x_j \in L$?". Beame and Fich show that if L is indecisive, then, in any implementation of the dynamic prefix problem for L on a RAM with word size $O(\log n)$, if the change operation takes time at most $2^{(\log n)^{1-\Omega(1)}}$ in the worst case, then the query operation takes time at least $\Omega(\log \log n / \log \log \log n)$ in the worst case.

Now let L be the regular language $(c + s + p)^* sp^* + p^*$, i.e. the language over $\{c, s, p\}$, where $x \in L$ if and only if the last letter of x which is not a p is an s. This language is indecisive, so Beame and Fich's result apply. Now assume that the dynamic connectivity problem for width 2 grid graphs can be solved with time bound $o(\log n \log n / \log \log \log n)$ per operation. We will show that the dynamic prefix problem for L can also be solved with time bound $o(\log n \log n / \log \log \log n)$, a contradiction.

Given an instance $a_1 a_2 \cdots a_n$ of the prefix problem to maintain, we maintain a grid graph G, of width 2 and length $2n$, defined as follows. Consider G to be divided into n blocks of length 2. The i'th block of G is $G(a_i)$, where G is the mapping defined in Figure 4. Clearly, a change of a symbol in the maintained string corresponds to a constant number of insert and delete operations in the dynamic graph. Now, in order to determine if $a_1 a_2 \dots a_i \in L$, we remove all edges in the $i+1$'st block of G using the delete operation of the dynamic connectivity operation. Then we ask if there is a path from the bottom left vertex of G to the bottom right vertex of the i'th block of G. This is the case if and only if $a_1 a_2 \dots a_i \in L$. After getting the right answer, we restore the data structure by reinserting the edges of block i. This completes the reduction, and the proof of Theorem 2.

5 Generalization to directed graphs

In a directed grid graph, each edge present has one or both of the two possible orientations, and we consider finding directed paths from s to t. (Such graphs correspond to planar nondeterministic branching programs or planar "switching-and-rectifier networks" [13], except that a directed grid graph need not have horizontal arrows in only one direction. It follows from the analysis here, of course, that this additional ability is of no use in the case of constant width.)

All of our theorems about constant-width grid graphs hold for directed grid graphs. Of course, the lower bounds are trivial extensions, but we must revisit the upper bounds:

Lemma 6. STCON_k *is in uniform* Π_k*, with the constructed circuit being positive in the edge variables.*

Proof. (sketch) Given a directed grid graph G, we will define its dual G^* as follows. The possible edge positions of G^* are exactly as in the undirected case. Now let e^* be an edge position of G^*, we define which orientations of e^* are present in G^* as a function of the orientations of e present in G: An orientation of e^* is present if and only if the orientation, turned 90° degrees clockwise, is *not* an orientation of e.

Now it is easy to see that there is a directed path from s to t if and only if there is *not* a directed cycle in G^*, going clockwise around t, with s on the outside. The rest of the proof proceeds exactly as in the proof of Lemma 3 — the only operations needed on the column numbers are comparisons.

As before, we can define a monoid whose elements are now directed $2 \times k$ grid graphs, and show that this monoid is aperiodic with dot-depth exactly k. As a corollary, we also get the same upper bound on the directed dynamic grid graph connectivity as on undirected dynamic grid graph connectivity. (Interestingly, for general graphs, the directed version of the dynamic problem seems to be much harder than the undirected version).

6 Discussion and open problems

- As mentioned in the introduction, Blum and Kozen showed that the general problem, where the width is not fixed, is in L. By carrying out our construction in Section 4 for a width of $\log n$, we get that even this restricted version of the general problem is hard for NC^1. An obvious open question is to determine the complexity of the general problem precisely: Is it complete for NC^1, or for L, or does it have intermediate complexity?
- We can also consider the general version of the problem for directed grid graphs, which is still hard for NC^1 but which we only know to be in NL. Our notion of duality gives a simple positive reduction of this problem to its complement, suggesting (but of course not proving, as NL is in fact closed under complement, by a more complicated reduction) that it is not NL-complete. There are potentially interesting restrictions of this problem as well, where we prohibit edges in one or two of the four directions.
- A similar gap occurs in our understanding of the dynamic grid graph connectivity problem, if the width of the graph is non-constant. If the width is a free parameter m, with the restriction $2 \leq m \leq n$, the following is known: Eppstein *et al* [6] construct a data structure with a time bound of $O(\log n)$ per operation and Eppstein [5] shows a lower bound of $\Omega(\log m / \log\log m)$. This lower bound is improved by Husfeldt and Rauhe [8] to $\Omega(m)$, provided $m \leq \log n / \log\log n$. Our upper bound is $2^{2^{O(m)}} \log\log n$ and our lower bound is $\Omega(\log\log n / \log\log\log n)$, provided $m \geq 2$. Combining everything, we get a lower bound of $\Omega(\min(m, \log n / \log\log n) + \log\log n / \log\log\log n)$ and an upper bound of $O(\min(\log n, 2^{2^{O(m)}} \log\log n))$ with a big gap to close.

Acknowledgements

We would like to thank Paul Beame and Arny Rosenberg for help with historical references, Howard Straubing and Denis Therien for very helpful discussions about automata theory and monoids, and Sairam Subramanian for very helpful discussions about dynamic graph problems. This work was greatly facilitated by the March 1997 Dagstuhl workshop in Boolean Function Complexity, attended by the first and third authors.

References

1. D. A. M. Barrington, N. Immerman and H. Straubing. On uniformity within NC^1 *Journal of Computer and System Sciences*, 41(3):274–306.
2. D. A. M. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35(4):941–952, October 1988.
3. P. Beame and F. Fich. On searching sorted lists: A near-optimal lower bound. Manuscript, 1997.
4. M. Blum and D. Kozen. On the power of the compass (or why mazes are easier to search than graphs). In *19th Annual Symposium on the Foundations of Computer Science*, pages 132–142, October 1978.
5. D. Eppstein. Dynamic connectivity in digital images. Technical Report 96-13, Univ. of California, Irvine, Department of Information and Computer Science, 1996.
6. D. Eppstein, G. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Algorithms*, 13:33–54, 1992.
7. G. S. Frandsen, P. B. Miltersen, and S. Skyum. Dynamic word problems. *Journal of the ACM* 44:257–271, 1997.
8. T. Husfeldt and T. Rauhe. Hardness results for dynamic problems by extensions of Fredman annd Saks' chronogram method.. Manuscript, 1997.
9. N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.
10. N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116(1):103–116, January 1995.
11. A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.
12. J. E. Pin. *Varieties of Formal Languages.* New York: Plenum Press, 1986.
13. A. A. Razborov. Lower Bounds for deterministic and nondeterministic branching programs. In L. Budach, ed., *Fundamentals of Computation Theory, 8th International Conference: FCT '91.* Lecture Notes in Computer Science 529, 47–60. Berlin, Springer Verlag, 1991.
14. M. Sipser. Borel sets and circuit complexity. In *Proceedings, 15th ACM Symposium on the Theory of Computing*, 1983, 61–69.
15. S. Skyum and L. G. Valiant. A complexity theory based on Boolean algebra. *Journal of the ACM*, 32(2):484–502, April 1985.
16. W. Thomas. Classifying regular events in symbolic logic. *J. Comput. System Sci.* 25, 1982, 360–376.