Logspace Algorithms for Computing Shortest and Longest Paths in Series-Parallel Graphs

Andreas Jakoby* and Till Tantau

Inst. für Theoretische Informatik, Universität zu Lübeck, Germany
{jakoby,tantau}@tcs.uni-luebeck.de

Abstract. For many types of graphs, including directed acyclic graphs, undirected graphs, tournament graphs, and graphs with bounded independence number, the shortest path problem is NL-complete. The longest path problem is even NP-complete for many types of graphs, including undirected K_5-minor-free graphs and planar graphs. In the present paper we present logspace algorithms for computing shortest and longest paths in series-parallel graphs where the edges can be directed arbitrarily. The class of series-parallel graphs that we study can be characterized alternatively as the class of K_4-minor-free graphs and also as the class of graphs of tree-width 2. It is well-known that for graphs of bounded tree-width many intractable problems can be solved efficiently, but previous work was focused on finding algorithms with low parallel or sequential *time complexity*. In contrast, our results concern the *space complexity* of shortest and longest path problems. In particular, our results imply that for directed graphs of tree-width 2 these problems are L-complete.

Keywords: Series-parallel graphs, logspace algorithms, distance problem, longest path problem, bounded tree-width, K_4-minor-free graphs.

1 Introduction

Series-parallel graphs form an extensively-studied class of graphs that has applications both in theory and in practice. Different types of series-parallel graphs have been studied in the literature; in the present paper we study their most general form, namely series-parallel graphs with an arbitrary number of terminals and with edges having arbitrary directions. There are two well-known alternative characterization, see for instance [5,13], of this class of graphs: First, it is also the class of directed graphs of tree-width at most 2. Second, it is also the class of directed graphs whose underlying undirected graph is K_4-minor-free.

For this class of graphs we study the longest and the shortest path problems. We are given an element G of the class as input together with two nodes s and t and we are asked to output a path (which may consist only of distinct nodes) of minimal or maximal length from s to t in G. For general graphs, the shortest path problem is well-known to be NL-complete, while the longest path problem is

* Part of this work was done while visiting the University of Frankfurt and the University of Freiburg, Germany.

V. Arvind and S. Prasad (Eds.): FSTTCS 2007, LNCS 4855, pp. 216–227, 2007.

Table 1. The complexity of path problems for different graph classes. In this present paper we investigate series-parallel graphs, which are the same as directed graphs of tree-width 2, and prove the results shown in bold. By "open" we mean that no nontrivial upper bounds are known.

Graph class	*Reachability*	*Distance*	*Longest path*	*Number of paths*
Digraphs of tree-width 1	L-compl.	L-compl.	L-compl.	L-compl.
Digraphs of tree-width 2	**L-compl.**	**L-compl.**	**L-compl.**	**L-compl.**
Digraphs of tree-width k, $k \geq 3$	open	open	$\in \mathrm{AC}^1$	$\in \mathrm{AC}^2$
Planar digraphs	$\in$ UL	open	NP-compl.	#P-compl.
Tournament graphs	$\in \mathrm{AC}^0$	NL-compl.	open	open
Undirected graphs	L-compl.	NL-compl.	NP-compl.	#P-compl.
Acyclic digraphs	NL-compl.	NL-compl.	NL-compl.	#L-compl.
Digraphs	NL-compl.	NL-compl.	NP-compl.	#P-compl.

NP-complete even for planar graphs. The different characterizations of the class of series-parallel graphs yields different insights into the complexity of the longest and shortest path problems for this particular class. Results from the theory of bounded tree-width tell us that the shortest path problem lies in the class NL and that the longest path problem lies in AC^1. Unfortunately, since it is only known that $\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1$, this does not tell us whether these problems can be solved in deterministic logspace. Results from the theory of series-parallel graphs tell us that conceptually simpler problems, like the reachability problem for directed two-terminal series-parallel graphs, lie in L.

The main result of the present paper, Theorem 5, lowers the upper bound on the complexity of shortest and longest path problems in directed graphs of tree-width 2 to L. At the same time, this result extends the previous complexity bounds on the reachability problem in directed two-terminal series-parallel graphs to the shortest and longest path problems in general multiple-terminal series-parallel graphs. Table 1 shows how these results relate to the complexity of shortest and longest path problems in other kinds of graphs. As can be seen in the table, for many types of graphs the distance problem is still NL-complete, including undirected graphs [7,23], directed acyclic graphs, tournament graphs [21], and graphs with bounded independence number [21].

Recently, it has been shown that the reachability problem is in L even for single source multiple sink planar DAGs [1]. If we restrict ourselves to planar digraphs, it is only known that the reachability problem lies in unambiguous logspace (i.e. UL $\cap$ co-UL) [8].

Our formulation of the main result does not treat shortest and longest paths separately. Rather, we allow input graphs to be equipped with integer edge weights coded in unary (negative weights are indicated by a flag). We present a deterministic logspace algorithm with the following properties: On input of a directed graph with integer weights coded in unary and two nodes s and t, it either determines that the graph is not a multiple-terminal series-parallel graph or it determines that there is no path from s to t or it outputs a path

from s to t of maximum total edge weight. Setting all edge weights to 1 makes a maximum-weight path a longest path and setting all edge weights to -1 makes a maximum-weight path a shortest path.

Graphs of tree-width 2. The tree-width of a graph is a measure of how close the graph is to being a tree and graphs of tree-width 1 are, indeed, trees. For a graph G of tree-width k there must exist a tree T whose nodes are labeled with so-called bags, which are just sets of up to $k+1$ nodes of the graph G. For each edge of the graph at least one of the bags must contain both endpoints of the edge, and the set of all bags containing any given graph node must form a connected subtree of T.

Certain intractable graph problems become tractable if we restrict ourselves to graphs with small tree-width, see for instance [4,25], and the problem of constructing tree decompositions of small tree-width is a well-studied topic, see [3,6,20]. For graphs of bounded tree-width one can construct a tree decompositions of constant width in AC^1, as shown in [6], and using such a decomposition one can determine the distance and the longest path length between two nodes efficiently in parallel [10,11,17]: In detail, Chaudhuri and Zaroliagis [10,11] have presented sequential linear-time algorithms and an EREW-PRAM algorithm working in time $O(T(t,n)+\log n)$ for finding a shortest path, where $T(t,n)$ denotes the time for computing a tree-decomposition of digraphs of n nodes of tree-width t. In [6] Bodlaender and Hagerup presented an EREW-PRAM algorithm using $O(\log^2 n)$ time that generates a tree decomposition of constant width. They also show that all graph properties of a finite index can be decided by an $O(\log n \log^* n)$ time EREW-PRAM. While many problems, including Hamiltonicity and the reachability problem, are of finite index, distance and longest path problems are not. For example, the problem of deciding whether the distance between two given nodes is at most $n/2$ in a graph of size n does not have a finite index.

It is well known that parallel time complexity and space complexity are related: $\mathrm{NC}^1 \subseteq \mathrm{L}$ and all languages in L can be decided by an EREW-PRAM in time $O(\log n)$ with a polynomial number of processors. If we replace L by NL, we must replace EREW-PRAM by CRCW-PRAM. It is also known that $\mathrm{NL} \subseteq \mathrm{LOGCFL} = \mathrm{SAC}^1$. However, it is not known whether $O(\log n)$-time-bounded EREW-PRAMs can be simulated by $O(\log n)$-space-bounded DTM.

***K_4*-minor-free graphs.** Directed graphs of tree-width 2 can also be characterized as graphs whose underlying undirected graph does not contain the K_4 as a minor. This means we cannot obtain K_4 by forgetting the direction of the edges and then repeatedly contracting and deleting edges and deleting isolated nodes.

Defining classes of graphs by forbidden minors is a powerful tool in graph theory. For example, the undirected K_3-minor-free graphs are exactly the forests (every cycle in a graph can be contracted down to a K_3). Planar graphs can be characterized as the graphs that are both K_5- and $K_{3,3}$-minor-free. We prove results for graphs that are K_4-minor-free. A next major algorithmic step forward would be a logspace algorithm for the distance problem in graphs whose

underlying undirected graph is K_5-minor-free. Such an algorithm would settle the challenging question of whether there is a logspace algorithm for the distance problem in planar graphs.

Series-parallel graphs. The third characterization of the graphs studied in this paper is the class of *mixed multiple-terminal series-parallel graphs*. More restricted versions are studied in the literature and we make use of these restricted versions in our proofs: In the proof of the main result we establish the existence of logspace algorithms for computing maximum-weight paths in more and more general forms of series-parallel graphs.

The simplest form are directed two-terminal series-parallel graphs. They are defined inductively, starting with the graph that consists of a single directed edge whose endpoints are called source terminal and sink terminal. Graphs can be composed in two ways: A serial composition fuses the sink of one graph with the source of another, a parallel composition fuses the two sources and also the two sinks of two graphs. Multiple-terminal series-parallel graphs are formed by taking a set of two-terminal series-parallel graphs and repeatedly fusing a terminal node with some node in one of the graphs.

For series-parallel graphs we can consider different possibilities for the direction of edges. For *directed* series-parallel graphs, once we choose a source and a sink terminal, the direction of all edges is also implied. Our algorithms do not only work for directed series-parallel graphs and for undirected series-parallel graphs, but also for the graphs obtained by arbitrarily redirecting the edges of a series-parallel graph. To distinguish the resulting type of graphs from directed series-parallel graphs, we will call them *mixed* series-parallel graphs.

The space complexity of problems related to series-parallel graphs has been analyzed in [19], where logspace algorithms for the recognition problem and for the reachability problem for directed two-terminal series-parallel graphs are presented. Furthermore, in the paper the problem of *decomposing* series-parallel graphs is studied. In [18], Jakoby and Liśkiewicz focus on the recognition, the reachability, and the decomposition problem for undirected series-parallel graphs and show that these problems can be solved in deterministic logspace using an SL oracle for reachability, which shows that decompositions can be computed in logspace. However, since reachability in directed graphs is NL-complete rather than SL-complete, the techniques presented in [18,19] fail for the mixed multiple-terminal series-parallel graphs that we consider in the present paper.

The time complexity of the recognition problem for series-parallel graphs has also been investigated in detail. An optimal linear-time sequential algorithm for this problem has been developed by Valdes, Tarjan, and Lawler [24] and fast parallel algorithms have been published. He and Yesha have presented an EREW-PRAM algorithm working in time $O(\log^2 n)$ while using $n + m$ processors [15]. Eppstein has reduced the time bound by constructing an algorithm that takes only $O(\log n)$ steps on the stronger CRCW-PRAM model and requires $C(m, n)$ processors [14], where $C(m, n)$ denotes the number of processors necessary to compute the connected components of a graph in logarithmic time. Finally, the

EREW-PRAM algorithm by Bodlaender and Antwerpen-de Fluiter [5] mentioned earlier also solves this problem in time $O(\log n \log^* n)$ using $O(n+m)$ operations.

2 Basic Definitions

A *graph* is a pair $G = (V, E)$ consisting of a *node set* V and an *edge set* E. A graph G is called a *directed graph* (or *digraph* for short) if $E \subseteq V \times V$ is a *set of directed edges*, G is called an *undirected graph* if $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is a *set of undirected edges*, and G is called a *mixed graph* if $E \subseteq V \times V \cup \big\{\{u, v\} \mid u, v \in V, u \neq v\big\}$ is a *set of edges*, such that we do not have both $(u, v) \in E$ and $\{u, v\} \in E$ for any pair $u, v \in V$. A *weighted mixed graph* is a mixed graph (V, E) together with an *edge weight function* $w\colon E \to \mathbb{Z}$.

Given two nodes $u, v \in V$ of a mixed graph G, we write $u \to_G v$ if either $(u, v) \in E$ or $\{u, v\} \in E$. Given a mixed graph $G = (V, E)$, its *undirected underlying graph* $\mathrm{uug}(G)$ is obtained by replacing every directed edge by an undirected edge, that is, $\mathrm{uug}(G) = \big(V, \{\{u, v\} \mid u \to_G v, u \neq v\}\big)$.

A *path* in a graph G is a sequence $(v_0, \ldots, v_\ell)$ of *distinct* nodes such that $v_0 \to_G v_1 \to_G \cdots \to_G v_\ell$. The number ℓ is the *length* of the path. We write $v_0 \to_G^* v_\ell$ to indicate that there exists a path from v_0 to v_ℓ in G. Given a weighted mixed graph G and a path, the *weight* of the path is the sum of the weights of the edges along this path. Given two nodes $u, v \in V$ we write $\mathrm{m}_G(u, v)$ for the maximum weight of any path from u to v or $-\infty$ if there is no path between them. Note that if all weights are 1, then $\mathrm{m}_G(u, v)$ is the length of a longest path from u to v; and if all weights are -1 then $\mathrm{m}_G(u, v)$ is the negated distance from u to v. An undirected graph is *1-connected* if there is a path between any two nodes. An undirected graph is *k-connected* if we must remove at least k nodes (along with all pending edges) so that the resulting graph is no longer 1-connected.

We use the notation $\langle X \rangle$ to denote a standard binary encoding of the object X. For example, for a graph G let $\langle G \rangle$ denote the binary encoding of the adjacency matrix of G. When we code weighted mixed graphs, *the weights are always coded in unary.*

An *arithmetic tree* is a tree whose leaves are labeled with integers and whose inner nodes have two children and are labeled with functions that maps pairs of integers to integers, like addition, maximization, or multiplication. We will call such functions *binary operators*. For a set O of operators, an *O-tree* is an arithmetic tree in which only operators from O are used. For example, a $\{+, \times\}$-tree is, in essence, an arithmetic formula. Given an O-tree, we recursively assign integers to the inner nodes by applying the operator of a node to the values of the children. We call the integer assigned to an inner node its *value* and the integers assigned to the root is the *value of the tree*. Given a set O of operators, the *tree value problem* for O-trees is the problem of computing the value of O-trees. *The integers at the leaves are coded in binary or in unary; we always indicate the coding explicitly.*

2.1 Definition of Series-Parallel Graphs

We now define different types of series-parallel graphs, abbreviated s-p-graphs in the rest of the paper. We start with two-terminal s-p-graphs.

Definition 1. *We define* directed two-terminal s-p-graphs *inductively. Syntactically, they are triples* (G, a, b) *consisting of a directed graph* $G = (V, E)$*, a* source terminal $a \in V$*, and a* sink terminal $b \in V$*. The following graphs are directed two-terminal s-p-graphs:*

1. (G, a, b) *where* G *is a single directed edge from* a *to* b*, that is,* $V = \{a, b\}$ *and* $E = \{(a, b)\}$*, is a directed two-terminal s-p-graph.*
2. *Given two directed two-terminal s-p-graphs* (G_1, a, c) *and* (G_2, c, b)*, their* serial composition *is a directed two-terminal s-p-graph with the terminals* a *and* b*. It is obtained by taking the disjoint union of* G_1 *and* G_2 *and identifying the two copies of the node* c*.*
3. *Given two directed two-terminal s-p-graphs* (G_1, a, b) *and* (G_2, a, b)*, their* parallel composition *is a directed two-terminal s-p-graph, again with the terminals* a *and* b*. It is obtained by taking the disjoint union of* G_1 *and* G_2 *and identifying the two copies of* a *and also the two copies of* b*.*

Definition 2. *An* undirected two-terminal s-p-graph *is a triple* (G, a, b) *such that there exists a directed two-terminal s-p-graph* (G', a, b) *with* $G = \mathrm{uug}(G')$*.*

Definition 3. *A* mixed two-terminal s-p-graph *is a triple* (G, a, b)*, where* G *is a mixed graph, for which* $\big(\mathrm{uug}(G), a, b\big)$ *is an undirected two-terminal s-p-graph.*

The last definition can be rephrased as follows: Mixed two-terminal s-p-graphs are obtained from directed two-terminal s-p-graphs by arbitrarily redirecting some or all of the edges.

Definition 4. *We define* undirected multiple-terminal s-p-graphs *inductively. Syntactically, they are pairs* (G, ω) *where* $\omega \subseteq V$ *is the set of* terminals*. The following graphs are undirected multiple-terminal s-p-graphs:*

1. *For every undirected two-terminal s-p-graph* (G, a, b)*, the pair* $(G, \{a, b\})$ *is an undirected multiple-terminal s-p-graph.*
2. *Given two undirected multiple-terminal s-p-graphs* (G_1, ω_1) *and* (G_2, ω_2)*, their* tree composition *is also an undirected multiple-terminal s-p-graph. It is obtained by taking the disjoint union of* G_1 *and* G_2 *and identifying one terminal* $f \in \omega_2$ *with an arbitrary node of* G_1*. The terminal set of the tree composition is* $\omega_1 \cup (\omega_2 - \{f\})$ *and we call* f *a* fusion node*.*

Definition 5. *A* mixed multiple-terminal s-p-graph *is a pair* (G, ω)*, where* G *is a mixed graph and* $\big(\mathrm{uug}(G), \omega\big)$ *is an undirected multiple-terminal s-p-graph.*

2.2 Decomposition Trees

Decomposition trees reflect the building process of series-parallel graphs. A parallel composition results in a "parallel node" in the tree, a serial composition

yields a "serial node," and single edges correspond to leaves. Note that the decomposition tree of an s-p-graph is typically not unique.

Definition 6. *A* decomposition tree *of a mixed two-terminal s-p-graph* (G, a, b) *is defined as follows. Syntactically, it consists of a directed binary tree* T *("binary" meaning that inner nodes have exactly two children, a left and a right one), whose node set is the disjoint union of the three* type sets T_l, T_s, *and* T_p, *a terminal-pair information function* terminals$\colon T_\mathrm{l} \cup T_\mathrm{s} \cup T_\mathrm{p} \to V \times V$, *and an edge information function* edge$\colon T_\mathrm{l} \to E$. *The set* T_l *contains exactly the leaves of* T. *The elements of* T_s *are called* serial nodes, *the elements of* T_p *are called* parallel nodes.

Having fixed the syntax of decomposition trees, we next inductively describe which trees are decomposition trees. In all cases, $\mathrm{terminals}(r) = (a, b)$ *must hold for the root* r *of the tree.*

1. *If* G *consists of a single edge* e *between the two nodes* a *and* b, *then* T *consists of a single node* $r \in T_\mathrm{l}$ *and* $\mathrm{edge}(r) = e$. *Note that the edge* e *may point from* b *to* a *for arbitrary mixed two-terminal s-p-graphs, but will always point from* a *to* b *if* (G, a, b) *is a directed two-terminal s-p-graph.*
2. *If* G *is the parallel composition of two mixed two-terminal s-p-graphs* (G_1, a, b) *and* (G_2, a, b) *and if* T_1 *and* T_2 *are their tree decompositions, respectively, then* T *consists of a root node* r *whose children are the roots of* T_1 *and* T_2 *and* $r \in T_\mathrm{p}$.
3. *If* G *is a serial composition of two mixed two-terminal s-p-graphs* (G_1, a, c) *and* (G_2, c, b), *we do exactly the same as in the parallel case, only* $r \in T_\mathrm{s}$.

We now extend the definition of decomposition trees to encompass multiple-terminal s-p-graphs. We then have a fourth type of nodes: "tree nodes", corresponding to *tree compositions*.

Definition 7. *Let* (G, ω) *be a mixed multiple-terminal s-p-graph. Its* decomposition tree T *is defined similarly to the decomposition tree in Definition 6, but with the following addition: There is a fourth type set* T_t, *together with the fusion information function* fusion$\colon T_\mathrm{t} \to V$. *If* T_t *is not empty, its elements must form a connected component of* T *and it must contain the root. The tree* T *is defined recursively according to the same rules as in Definition 6 with the following addition:*

4. *If* G *is the tree composition of two mixed multiple-terminal s-p-graphs* (G_1, ω_1) *and* (G_2, ω_2) *and if* T_1 *and* T_2 *are their decomposition trees, respectively, then* T *consists of a root node* r *whose children are the roots of* T_1 *and* T_2. *We have* $r \in T_\mathrm{t}$ *and* $\mathrm{fusion}(r)$ *is the fusion node of the tree composition.*

2.3 Facts from the Literature Used in Our Proofs

We now list facts from the literature on s-p-graphs that will be used in our proofs.

Fact 1 ([19]). *There exists a logspace machine that on input of a directed graph G decides whether G is a directed two-terminal s-p-graph and, if this is the case, outputs a decomposition tree for it.*

Fact 2 ([19]). *There exists a logspace machine that on input of a directed two-terminal s-p-graph G and two nodes s and t decides whether there is a path from s to t.*

The following fact follows from the results in [18] and the fact that L = SL, see [22].

Fact 3 ([18]). *There exists a logspace machine that on input of an undirected graph G decides whether there is a terminal set ω such that (G, ω) is an undirected multiple-terminal s-p-graph and, if this is the case, outputs a decomposition tree T for it. Furthermore, every node n of T that is not an element of T_{t}, but whose parent is an element of T_{t}, has the following property: The undirected two-terminal s-p-graph described by the subtree of T rooted at n is 2-connected.*

The following fact is a conclusion of Lemma 8 and Theorem 6 from [18].

Fact 4. *There exists a logspace machine that on the input of an undirected 2-connected two-terminal s-p-graph (G, a, b) and a node $a' \in V$ computes a node $b' \in V$ such that (G, a', b') is an undirected two-terminal s-p-graph.*

Essentially, this fact states that we can "choose" the source terminal arbitrarily. But we cannot also choose the sink terminal arbitrarily at the same time.

3 Computing Maximum-Weight Paths in Logspace

In the present section we prove the central result of the paper, Theorem 5 below. Recall that weights are given in unary.

Theorem 5. *There is a logspace algorithm whose inputs are codes of weighted mixed graphs $G = (V, E)$ together with two nodes $s, t \in V$ and whose output is one of the following:*

1. *The algorithm determines that G is not a mixed multiple-terminal s-p-graph.*
2. *The algorithm determines that there is no path from s to t in G.*
3. *The algorithm outputs a path from s to t of maximal weight.*

The first step in the proof is an algorithm for computing a maximum-weight path in a weighted directed two-terminal s-p-graph from the source to a given node. Instead of writing down an explicit algorithm, we establish a series of reductions that ends with a problem that is known to be solvable in logspace.

The second step is an algorithm for computing maximum-weight paths between the terminals in weighted mixed two-terminal s-p-graphs. The main idea is to obtain a directed version of the mixed graph and to put a heavy penalty on all edges that "point in the wrong direction." We can then use the algorithm for weighted directed two-terminal s-p-graphs.

The third step is an algorithm for computing a maximum-weight path from the source a to an arbitrary node t in weighted mixed two-terminal s-p-graphs. A recursive algorithm is used to compute the maximum weight of a path from s to t by keeping track of smaller and smaller "intervals" (which are just subgraphs) that contain t and, at the same time, keeping track of the maximum weight of paths from a to the two "endpoints" of the intervals.

The fourth and last step is to consider weighted mixed multiple-terminal s-p-graphs G.

3.1 Terminal-to-Node Paths in Directed Two-Terminal S-P-Graphs

Theorem 6. *There exists a logspace machine that on input of any weighted directed two-terminal s-p-graph (G, a, b, w) and a node t outputs a maximum-weight path from a to t.*

Recall once more that weights are coded in unary. The algorithm internally uses an oracle M_{at} and the main task is to prove that M_{at} lies in the class L. The oracle is the decision version of the path construction problem:

$$M_{at} = \{\langle G, a, b, w, t, d\rangle \mid (G, a, b, w) \text{ is a weighted directed two-terminal s-p-graph in which there is a path from } a \text{ to } t \text{ of weight at least } d\}$$

To prove $M_{at} \in \mathrm{L}$, we establish a line of reductions. Note that the difficulty lies in computing the maximum weight of paths, not in checking whether the input graph is, indeed, a directed two-terminal s-p-graph, see Fact 1. The first reduction reduces M_{at} to M_{ab}, which is the restricted version of M_{at} where only inputs with $t = b$ are allowed. If we consider only the subset of nodes $V' = \{v \mid v \to_G^* t\}$ of the input graph G, we can show that:

Lemma 1. *M_{at} reduces to M_{ab} via a logspace many-one reduction.*

We next reduce M_{ab} to M_{ab}^+, which is the same problem, only all weights must be positive.

Lemma 2. *M_{ab} reduces to M_{ab}^+ via a logspace many-one reduction.*

Lemma 3. *M_{ab}^+ reduces to the tree value problem for $\{+, \max\}$-trees whose leaves are labeled with positive integers coded in unary via a single-query logspace reduction.*

The last step is to reduce the tree value problem for $\{+, \max\}$-trees whose leaves are labeled with positive integers coded in unary to the tree value problem for $\{+, \times\}$-trees, which is known to lie in logspace [9,2,12,16].

Lemma 4. *The tree value problem for $\{+, \max\}$-trees whose leaves are labeled with positive integers coded in unary reduces to the tree value problem for $\{+, \times\}$-trees whose leaves are labeled with integers coded in binary via a single-query logspace reduction.*

Using M_{at} as an oracle, we can construct a maximum-weight path node by node. This proves Theorem 6.

3.2 Terminal-to-Terminal in Mixed Two-Terminal S-P-Graphs

Theorem 7. *There exists a logspace machine that on input of any weighted mixed two-terminal s-p-graph (G, a, b, w) outputs a maximum-weight path from a to b or determines that no path exists.*

To prove the theorem, we introduce the notion of *green edges*, which are edges that "point in the right direction."

Definition 8. *Let (G, a, b) be a mixed two-terminal graph and let T be a decomposition tree for it. We color the edges of G according to the following rules: Let e be an edge of G and let n be the leaf node of T with $\text{edge}(n) = e$. Then, if $e = (x, y) \in V \times V$ but $\text{terminals}(n) = (y, x)$, we color the edge red; otherwise, namely when $e = (x, y)$ and $\text{terminals}(n) = (x, y)$ or when $e = \{x, y\}$ is undirected, we color it green.*

Let (G, a, b) be a mixed two-terminal s-p-graph and let T be a decomposition tree for G. Then every path from a to b uses only green edges. The key idea in proving Theorem 7 is to turn mixed s-p-graphs into directed s-p-graphs by redirecting all red edges while assigning large negative weights to them. We can then apply the algorithm from Theorem 6 to the resulting graph.

3.3 Terminal-to-Node Paths in Mixed Two-Terminal S-P-Graphs

Theorem 8. *There exists a logspace machine that on input of any weighted mixed two-terminal s-p-graph (G, a, b, w) and a node t outputs a maximum-weight path from a to t or determines that no such path exists.*

For the proof we introduce the notion of "intervals," which contain t and which get smaller and smaller. We will keep track of the maximum weights of paths from the source to the two endpoints of the intervals.

Definition 9. *Let (G, a, b) be a mixed two-terminal s-p-graph and let T be a decomposition tree. Given a node n of T, let (G_n, a_n, b_n) denote the mixed two-terminal s-p-graph that is described by the subtree of T rooted at n. We call (G_n, a_n, b_n) the* interval described by *n.*

For a node n of T we write $G - G_n$ for the graph obtained from G by deleting all edges of the graph G_n and the resulting isolated nodes. The weight record for *n is the tuple $\left(m_{a\to a_n}^{\neg via\ b_n}, m_{a\to a_n}^{via\ b_n}, m_{a\to b_n}^{\neg via\ a_n}, m_{a\to b_n}^{via\ a_n}\right)$ where $m_{a\to a_n}^{\neg via\ b_n}$ is the maximum weight of a path in $G - G_n$ from a to a_n that does not contain b_n, while $m_{a\to a_n}^{via\ b_n}$ is the maximum weight of a path in $G - G_n$ from a to a_n that does contain b_n. Similarly, $m_{a\to b_n}^{\neg via\ a_n}$ is the maximum weight of a path in $G - G_n$ from a to b_n that does not contain a_n, while $m_{a\to b_n}^{via\ a_n}$ is the maximum weight of a path in $G - G_n$ from a to b_n that does contain a_n.*

Lemma 5. *There exists a logspace machine that on input of any weighted mixed two-terminal s-p-graph (G, a, b, w) and a node t outputs $\mathrm{m}_G(a, t)$.*

Proof (Sketch of proof). To compute $\mathrm{m}_G(a,t)$, we generate the decomposition tree T of (G,a,b). Let r be the root of T and let $n_1, \ldots, n_k$ be the path in T that leads from $n_1 = r$ to a leaf n_k where one endpoint of edge(n_k) is t. We compute for successive $i = 1,\ldots,k$ the weight records for each n_i, using only the weight record of the previous n_{i-1} as a guide. □

To construct an path of maximum length we repeatedly apply the algorithm from Lemma 5 as a "guide" that tells us how we must extend the path as we descend. This proves Theorem 8.

3.4 Node-to-Node Paths in Mixed Multiple-Terminal S-P-Graphs

We first compute the tree decomposition of G. The tree decomposition allows us to identify components of G, each of which is a two-terminal s-p-graph, such that every path from s to t must go through a unique sequence of these components. Inside each component we can compute maximum-weight paths using the algorithms we obtained in the previous steps. Stringing together the paths yields the overall path. This proves Theorem 5.

4 Conclusion

In this paper we presented a logspace algorithm for computing paths of maximum weight in mixed multiple-terminal s-p-graphs. As mentioned in the introduction, little is known in comparison about the space complexity of the shortest and longest path problems for graphs with higher, but still constant tree-width. It is neither known whether one can solve the reachability problem for directed graphs of tree-width 3 in logspace nor whether the reachability problem for directed graphs of tree-width k is hard for the class NL for some constant $k \geq 3$.

On the positive side, a closer analysis of our approach shows that one can use the algorithm to count the number of self-avoiding paths in mixed multiple-terminal s-p-graphs using a logspace algorithm. Also, the existence of an efficient algorithm for computing longest paths implies further results like an efficient algorithm for computing topological sortings. Another application is the computation of s-t-enumerations.

References

1. Allender, E., Barrington, D.A.M., Chakraborty, T., Datta, S., Roy, S.: Grid graph reachability problems. In: 21th Annual IEEE Conference on Computational Complexity (CCC), pp. 299–313 (2006)
2. Ben-Or, M., Cleve, R.: Computing algebraic formulas using a constant number of registers. SIAM J. Comput. 21, 54–58 (1992)
3. Bodlaender, H.L.: NC-algorithms for graphs with small treewidth. In: 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 1–10.

4. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 1–14.
5. Bodlaender, H.L., de Fluiter, B.A.: Parallel algorithms for series parallel graphs and graphs with treewidth two. Algorithmica 29(4), 534–559 (2001)
6. Bodlaender, H.L., Hagerup, T.: Parallel algorithms with optimal speedup for bounded treewidth. SIAM J. Comput. 27, 1725–1746 (1998)
7. Borodin, A., Cook, S.A., Dymond, P.W., Ruzzo, W.L., Tompa, M.: Two applications of inductive counting for complementation problems. SIAM J. on Computing 18(3), 559–578 (1989)
8. Bourke, C., Tewari, R., Vinodchandran, N.V.: Directed planar reachability is in unambiguous log-space. In: 22th Annual IEEE Conference on Computational Complexity (CCC), pp. 217–221 (2007)
9. Buss, S., Cook, S., Gupta, A., Ramachandran, V.: An optimal parallel algorithm for formula evaluation. SIAM J. Comput. 21, 755–780 (1992)
10. Chaudhuri, S., Zaroliagis, C.D.: Shortest paths in digraphs of small treewidth. Part II: Optimal parallel algorithms. Theoretical Comput. Sci. 203, 205–223 (1998)
11. Chaudhuri, S., Zaroliagis, C.D.: Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. Algorithmica 27(3), 212–226 (2000)
12. Chiu, A., Davida, G., Litow, B.: Division in logspace-uniform NC^1. Theoretical Informatics and Applications 35, 259–275 (2001)
13. Duffin, R.: Topology of series-parallel networks. J. Math. Analysis and Applications 10, 303–318 (1965)
14. Eppstein, D.: Parallel recognition of series-parallel graphs. Inf. and Comp. 98, 41–55 (1992)
15. He, X., Yesha, Y.: Parallel recognition and decomposition of two terminal series parallel graphs. Inf. and Comp. 75, 15–38 (1987)
16. Hesse, W.: Division is in uniform TC^0. In: 28th International Colloquium on Automata, Languages and Programming (ICALP), pp. 104–114
17. Hohberg, W., Reischuk, R.: A framework to design algorithms for optimization problems on graphs. Technical Report ITI, Technical University Darmstadt (1990)
18. Jakoby, A., Liśkiewicz, M.: Paths problems in symmetric logarithmic space. In: 29th International Colloquium on Automata, Languages and Programming (ICALP), pp. 269–280.
19. Jakoby, A., Liśkiewicz, M., Reischuk, R.: Space efficient algorithms for series-parallel graphs. J. of Algorithms 60, 85–114 (2006)
20. Lagergren, J.: Efficient parallel algorithms for graphs of bounded tree-width. J. of Algorithms 20, 20–44 (1996)
21. Nickelsen, A., Tantau, T.: The complexity of finding paths in graphs with bounded independence number. SIAM J. Comput. 34(5), 1176–1195 (2005)
22. Reingold, O.: Undirected s-t-connectivity in log-space. In: 37th ACM Symposium on Theory of Computing (STOC), pp. 376–385 (2005)
23. Toda, S.: Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, Dept. Comp. Sci. and Inform. Math., Univ. Elect.-Comm., Chofu-shi, Tokyo 182, Japan (1991)
24. Valdes, J., Tarjan, R., Lawlers, E.: The recognition of series parallel digraphs. SIAM J. Comput. 11, 298–313 (1982)
25. Wanke, E.: Bounded tree-width and LOGCFL. J. of Algorithms 16, 470–491 (1994)