

Computing Shortest Paths in Series-Parallel Graphs in Logarithmic Space

Andreas Jakoby
Institut für Theoretische Informatik
Universität zu Lübeck
D-23538 Lübeck, Germany

Till Tantau
Institut für Theoretische Informatik
Universität zu Lübeck
D-23538 Lübeck, Germany

May 28, 2006

Abstract

Series-parallel graphs, which are built by repeatedly applying series or parallel composition operations to paths, play an important role in computer science as they model the flow of information in many types of programs. For directed series-parallel graphs, we study the problem of finding a shortest path between two given vertices. Our main result is that we can find such a path in logarithmic space, which shows that the distance problem for series-parallel graphs is L-complete. Previously, it was known that one can compute *some* path in logarithmic space; but for other graph types, like undirected graphs or tournament graphs, constructing *some* path between given vertices is possible in logarithmic space while constructing a *shortest* path is NL-complete.

1 Introduction

Problem statement. A well-studied subclass of graphs are *series-parallel graphs*, for which different definitions and characterizations can be found in the literature. We focus on the basic class, sometimes also called *two terminal series-parallel graphs*, which are the most important variant for applications in program analysis. Series-parallel graphs can be used to describe the information flow within a program that is based on sequential and parallel composition and help in deciding which parts of a program be parallelised and in generating schedules for a parallel execution.

In this paper we address the problem of determining the distance between two nodes in a directed two terminal series-parallel graph, which is, *a priori*, a more difficult problem than the reachability problem for this graph class. It is known that the reachability problem (the problem of telling whether there exists a path between two given vertices) and the distance problem (the problem of telling whether there exists a path between two given vertices of a given length) have different complexities for certain types of graphs. While both problems are NL-complete for general graphs, for undirected graphs the distance problem is NL-complete, but the reachability problem is L-complete [19]. For tournaments, the distance problem is also NL-complete, but the reachability problem is L-complete [18]. Finally, for forests both problems are L-complete.

Our contribution. The main result of the present paper is that the distance problem for directed two-terminal series-parallel is solvable in logarithmic space. We prove this by presenting a reduction of the distance problem to the problem of evaluating an arithmetic (+, min)-formula over unary given values. We show that the evaluation problem these formulas can be reduced

in logarithmic space to the problem of evaluating arithmetic $(+, \cdot)$ -formulas over binary given values. Using the results in [7, 4, 8, 12] one can show that such a formula can be evaluated in L.

The distance problem for directed two-terminal series-parallel graphs is equivalent to a number of other problems, whose complexity is also settled by our main result. These other problems include computing the longest path in a two terminal series-parallel graphs and computing the width of such a graph.

Because of the relation between L and parallel time complexity classes defined by the EREW PRAM model (see [17]) these new algorithms can be modified to solve these problems in logarithmic time on EREW PRAMs as well.

Related work. To determine whether a given graph G belongs to the class of series-parallel graphs is a basic problem in algorithmic graph theory. An optimal linear time sequential algorithm for this problem has been developed by Valdes, Tarjan, and Lawler [21] 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 [13]. Eppstein has reduced the time bound constructing an algorithm that takes only $O(\log n)$ steps on the stronger CRCW-PRAM model with concurrent instead of exclusive read and write, that requires $C(m, n)$ processors [11], where $C(m, n)$ denotes the number of processors necessary to compute the connected components of a graph in logarithmic time. Finally, Bodlaender and de Fluiter have presented an EREW-PRAM algorithm using $O(\log n \log^* n)$ time and $O(n + m)$ operations [5].

The space complexity of this problem has been analysed by Jakoby, Liśkiewicz and Reischuk. In [14] they presented logarithmic space algorithms for the recognition problem and for the reachability problem for directed two terminal series-parallel graphs. Furthermore they have studied the problem of *decomposing* a series-parallel graph. In [15] Jakoby and Liśkiewicz focused on the recognition, the reachability, and the decomposition problem of undirected series-parallel graphs and showed that this problems can be solved in deterministic logarithmic space using an SL oracle, which shows that decompositions can be computed in logarithmic space, also.

2 Preliminaries

2.1 Graphs, Graph Problems, Series-Parallel Graphs

The term *graph* will always refer to directed graphs, formalized as a pair (V, E) with $E \subseteq V \times V$. Instead of $(u, v) \in E$ we will also write $u \rightarrow v$. A *source* of a graph is a vertex with in-degree zero, a *sink* is a vertex with out-degree zero.

A *path* in a graph is a sequence (v_0, \dots, v_ℓ) of vertices such that $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_\ell$. The number ℓ is the *length* of the path. The *distance* $\text{dist}(u, v, G)$ of two vertices u, v is the length of the shortest path between them in G or infinity if there is no path between them. We write $u \rightarrow^* v$ if there is a path from u to v .

Given a class C of graphs, we define the following two computational problems:

$$\begin{aligned} C\text{-REACH} &:= \{(V, E, s, t) \mid (V, E) \in C, s \in V, t \in V, \text{there is a path from } s \text{ to } t \text{ in } (V, E)\}, \\ C\text{-DISTANCE} &:= \{(V, E, s, t, d) \mid (V, E) \in C, s \in V, t \in V, \\ &\quad \text{the distance of } s \text{ and } t \text{ in } (V, E) \text{ is at most } d\}. \end{aligned}$$

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 adjacency matrix of G .

The class of (directed) series-parallel graphs is defined recursively as follows: First, every (directed) path is a series-parallel graph. Second, given two series-parallel graphs G_1 and G_2

sources s_1 and s_2 and sinks t_1 and t_2 , their *serial composition*, obtained by taking the disjoint union of G_1 and G_2 and identifying t_1 and s_2 , is also a series-parallel graph. Third, again given two graphs series-parallel graphs G_1 and G_2 , their *parallel composition*, obtained by taking the disjoint union of G_1 and G_2 and identifying s_1 and s_2 and also t_1 and t_2 , is a series-parallel graph.

From [14] it is known that for any type of graph representations we can always compute a graph representation for a series-parallel graph that reflects the structure of the series-parallel composition operations in L. Hence, we can assume that the graph is given in an appropriate representation.

2.2 Trees, Decomposition Trees, Arithmetic Trees

A *tree* is a graph in which there is a unique path from every vertex to a special vertex called the *root* of the tree.

A *decomposition tree* is a tree whose inner nodes are labeled with the values p and s , which stand for *parallel composition* and *serial composition*. A decomposition tree *describes* the series-parallel graph that is constructed recursively as follows: To each leaf of the tree we assign a length-1 path. If we have assigned two series-parallel graphs to the children of an inner node labeled p , we assign the parallel composition of these graphs to the inner node. Similarly, to an inner node labeled s we assign the serial composition of the children. The series-parallel graph assigned to the root in this manner is the graph described by the tree. Note that different decomposition trees can describe the same graph.

An *arithmetic tree* is a tree whose leaves are labeled with integers and whose inner vertices have two children and are labeled with functions that map 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. A *unary O -tree* is similar to an O -tree, but the integers must be positive and given in unary. 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 integer 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 -tree. We may choose to simplify the problem by considering only unary O -trees.

3 A Logspace Algorithm for the Distance Problem

In the present section our aim is to prove the following theorem.

Theorem 3.1. SERIES-PARALLEL-DISTANCE \in L.

The theorem states that on input of a tuple $\langle V, E, s, t, d \rangle$, we can decide in logarithmic space whether a) the graph $G = (V, E)$ is a series parallel graph and b) whether there is a path from s to t in G of length at most d .

The hard part is computing the distance – it is known [14] that *checking* whether a graph is series-parallel can be done in logarithmic space and deciding *reachability* for series-parallel graphs can also be done in logarithmic space. It is even possible to compute a decomposition tree for a given series-parallel graph in logarithmic space:

Theorem 3.2 ([15]). *There exists a logspace machine that on input of a directed graph G determines that the graph is not a series-parallel graph or outputs a decomposition tree of the graph.*

Our proof of Theorem 3.1 proceeds in several steps.

1. We generalize the problem by considering weighted graphs.
2. We reduce the problem to the problem of finding a *longest* path.
3. We reduce the longest path problem to the tree value problem for unary $\{+, \max\}$ -trees.
4. We reduce the tree value problem for unary $\{+, \max\}$ -trees to the tree value problem for $\{+, \times\}$ -trees.

Before we proceed, let us make one easy observation: We may assume that the start vertex s is the source of the series-parallel graph (the single vertex of in-degree zero) and t is the sink of the series-parallel graph (the single vertex of out-degree zero). The reason is that, since reachability is decidable in logarithmic space for series-parallel graphs, we can filter out all vertices that do not lie on a path from s to t in (V, E) . More precisely, there is logspace reduction from SERIES-PARALLEL-DISTANCE to itself such that for all instances in the range of the reduction s is the source and t is the sink. This reduction works by checking, for each vertex v of its input graph, whether there is a path from s to v and a path from v to t . Only vertices passing this test are included in the output graph of the reduction. Clearly, the removed vertices play no role with respect to the question what paths exist between s and t . Because of these arguments, in the following we always assume that s is the source and t is the sink of the graphs.

3.1 Weighted Series-Parallel Graphs

We start with generalizing the problem slightly. We now consider inputs where we are given, in addition to the graph $G = (V, E)$, the vertices s and t , and the distance d , also a weight function $w: V \rightarrow \mathbb{Z}$ that assigns an integer to each edge of the series-parallel graph.

Given such a weighted graph, we define the weight of a path in the obvious manner: Given a path (v_0, \dots, v_ℓ) , its *weight* is the sum of the weights along the edges of the path. Note that a path may have a negative weight, but since series-parallel graphs are directed acyclic graphs we do not have to worry about cycles with a negative weight.

We can now consider a new problem: On input $\langle V, E, s, t, d, w \rangle$ we must now decide whether $G = (V, E)$ is a series-parallel graph in which there is a path of weight at most d . We require that the numbers of the weight function w are given in *unary* but we allow also *negative weights (also given in unary)*. So, the sign of the weight is stored explicitly and the absolute value of the weight is given in unary. Let us call this problem W_{\leq} (where W stands for *weighted*). We also introduce the problem W_{\leq}^+ where all weights are positive.

Clearly, we can reduce SERIES-PARALLEL-DISTANCE to W_{\leq} and even to W_{\leq}^+ using first-order projections. In the other direction, we can reduce W_{\leq}^+ to SERIES-PARALLEL-DISTANCE by replacing each edge of weight w by a path of length w . This can also be done efficiently since weights are given in unary. It is *not* clear how we could reduce W_{\leq}^+ to W_{\leq} , we come back to this problem in the next section.

Recall that our first objective is to reduce the shortest path problem for series-parallel graphs to the longest path problem for these graphs. As for the shortest path problem, there also exists a weighted version for the longest path problem. Let W_{\geq} be the same problem as W_{\leq} , only we now ask whether there exists a path from s to t of weight *at least* d . Let W_{\geq}^+ be the problem restricted to instances with positive weights. Then we can reduce the (ordinary) longest path problem for series-parallel graphs to W_{\geq}^+ and we can reduce W_{\geq}^+ to the longest path problem.

Our final observation for this section is that W_{\leq} and W_{\geq} are equivalent under first-order projections: The reduction (which works in both directions) simply maps an instance to the instance in which all weights are negated and in which the target distance d is negated. Then, clearly, there is a path from s to t of a weight w for the original weights if, and only if, there is a path from s to t of weight $-w$ for the negated weights. In particular, there is a path of weight at most d for the original weights if, and only if, there is a path of weight at least $-d$ for the negated weights.

3.2 Reduction to the Longest Path Problem

We now aim at reducing the shortest path problem for series-parallel graph to the longest path problem. How this reduction works is demonstrated in the next lemma. Prior to stating this lemma, let us quickly define the longest path problem formally.

$$\begin{aligned} \text{SERIES-PARALLEL-LONGEST-PATH} := \\ \{ \langle V, E, s, t, d \rangle \mid (V, E) \text{ is a series-parallel graph, } s \in V, t \in V, \\ \text{there is a path of length at least } d \text{ from } s \text{ to } t \text{ in } (V, E) \}. \end{aligned}$$

Lemma 3.3. $\text{SERIES-PARALLEL-DISTANCE} \leq_{\text{fop}} \text{SERIES-PARALLEL-LONGEST-PATH}$.

Proof. The idea is to use the weighted graphs introduced earlier. We know already that $\text{SERIES-PARALLEL-DISTANCE}$ reduces to W_{\leq} , which reduces to W_{\geq} in turn. We also know that we can reduce W_{\geq}^+ to $\text{SERIES-PARALLEL-LONGEST-PATH}$. The missing link in the chain of reductions is a reduction from W_{\geq} to W_{\geq}^+ . In the following we show how a logspace many-one reduction from W_{\geq} to W_{\geq}^+ works.

Let $\langle V, E, s, t, d, w \rangle$ be an input for the reduction. In order to obtain the desired input for W_{\geq}^+ , we will modify only the weight function w and the target value d . We will *not* modify the graph. The new weight function will be called w' and the new target value d' .

For an edge $e = (u, v) \in E$, let P_e be the set of vertices x such that $x \rightarrow^* u$. In other words, P_e contains the vertices from which we can still reach the start of the edge. Note that $s \in P_e$ since s is the source and every vertex is reachable from the source (recall that we consider only graphs in which s is the source). We next we define a subset $E_e \subseteq E$ of edges as follows: An edge $(x, y) \in E$ is in E_e if $x \in P_e$ and $y \notin P_e$. Note that $e \in E_e$. Note also that, given e , we can check in logarithmic space whether $(x, y) \in E_e$ holds as this check involves only reachability checks, which can be done in logarithmic space.

Let us now establish some basic properties of the sets P_e and E_e . For every edge e and every path from s to t , the vertices on the path start inside P_e and continue to stay inside P_e until, at some point, the first vertex leaves P_e (at the latest, the vertex t does). Once the sequence has left P_e , it stays outside P_e . This means that every path from s to t passes exactly one edge from E_e .

Consider what happens if we increase the weight of all edges in E_e by a constant c . Then the weight of every path from s to t is increased exactly by this constant since every path passes through exactly one edge of E_e . The idea, to be detailed in a moment, is to increase the weight of all edges in this manner so strongly that all weights become positive.

Let m be the minimum weight present in the input graph. If m is positive, nothing needs to be done, so assume that it is negative or zero. Let $k = -m + 1$. Our objective is to do the following for each edge $e \in E$: We increase the weight of all edges in E_e by k . Thus, each edge $e \in E$ causes a weight increase of some edges in the graph and some edges may be increased multiple times. Formally, the total new weight $w'(e')$, where e' is some edge from E ,

is $w(e') + k \cdot |\{e \mid e' \in E_e\}|$. Clearly this number can be computed in logarithmic space since checking whether $e' \in E_e$ holds can be done in logarithmic space.

Let $|E|$ be the total number of edges in the graph. We claim that our modification of the weights has increased the weight of every path from s to t exactly by $k|E|$. To see this, just note that every path passes through exactly one edge from each E_e and we increased the weight of each such edge by $k|E|$. But then for $d' = d + k|E|$, we have that $\langle V, E, s, t, d', w' \rangle \in W_{\geq}$ if, and only if, $\langle V, E, s, t, d', w' \rangle \in W_{\geq}$. Finally, note that all weights assigned by w' are positive since $e \in E_e$ and, thus, each weight is increased by at least k , which makes it positive. This observation allows us to conclude that $\langle V, E, s, t, d', w' \rangle \in W_{\geq}$ if, and only if, $\langle V, E, s, t, d', w' \rangle \in W_{\geq}^+$. \square

3.3 Reduction to Addition and Maximization Trees

The next step toward a proof of Theorem 3.1 is a reduction of the longest path problem for series-parallel graphs to the tree value problem for unary $\{+, \times\}$ -trees.

Lemma 3.4. *SERIES-PARALLEL-LONGEST-PATH reduces to the tree value problem for unary $\{+, \max\}$ -trees via a logspace many-one reduction.*

Proof. This reduction is quite simple. On input $\langle V, E, s, t, d \rangle$, where s is the source and t is the sink, we compute the decomposition tree of the graph (V, E) . Next, we turn the tree into a $\{+, \max\}$ -tree as follows: We label each leaf with the number 1. We change each p -label (recall that p stands for parallel) into a \max -label and change s -label into a $+$ -label.

We claim that the value of the resulting $\{+, \max\}$ -tree is the length of the longest path in the graph. This is easy to prove using structural induction: First, decomposition trees consisting only of a single leaf (which equals the root) describe the series-parallel graph consisting of a single edge between s and t . In such graphs the length of the longest (or shortest or just any) path from s to t is 1 and this is the value of the tree.

Second, given a decomposition tree with a p -label at the root, assume that the lengths of the longest paths in the two graphs G_1 and G_2 described by the trees rooted at the two children c_{left} and c_{right} equal the values of these children. Then the longest path from s to t in the graph resulting from the parallel composition of G_1 and G_2 is the maximum of the two longest paths in G_1 and G_2 . On the other hand, the value of the root is exactly the maximum of the values of the children.

Third, consider the situation where the root of the decomposition tree is labeled s . Let once more G_1 and G_2 be the graphs described by the root's children. A longest path in the serial composition of G_1 and G_2 is given by first going through G_1 via a longest path in G_1 and then going through G_2 via a longest path of G_2 . Then the total length is the sum of the length of the two longest paths. This shows that the value of the root of the tree, which is the sum of the values of the children is, indeed, exactly the length of the longest path through the whole graph. \square

3.4 Reduction to Addition and Multiplication Trees

The last step is to reduce the tree value problem for unary $\{+, \max\}$ -trees to the tree value problem for $\{+, \times\}$ -trees. Once we have established this reduction, we have proved Theorem 3.1 as it is known that the tree value problem for $\{+, \times\}$ -trees can be solved in logarithmic space [7, 4, 8, 12].

Lemma 3.5. *The tree value problem for unary $\{+, \max\}$ -trees reduces to the tree value problem for $\{+, \times\}$ -trees via single-query Turing-reduction that runs in logarithmic space (even in NC^1).*

Proof. Let T be an input $\{+, \max\}$ -tree and let m be the sum of all number of the tree's leaves plus one. We map T to an $\{+, \times\}$ -tree T' as follows: For a leaf with value v , we change the value to 2^{vm} . Note that this mapping is feasible as both v and m are "small" number: Their values are polynomial in the input length. For inner nodes, change the labels according to the following rule: A $+$ -label is replaced by a \times -label, a \max -label is replaced by a $+$ -label.

Let v be the value of T and let v' be the value of T' . We claim that $v = \lfloor (\log_2 v')/m \rfloor$. If we accept this claim for the moment, the reduction can the query T' and receive the value v' . Then v can easily be obtained from v' using the above formula.

To prove the formula, we prove the following using simultaneous structural induction on the trees T and T' (these trees have the same structure, only the labels differ): Let u be a value of a node n in T and let u' be the value of this node in T' . Let s be the sum of the values of the leaves of the subtree of T rooted at n . We claim that

$$2^{um} \leq u' \leq 2^{um+s}.$$

Note that if the claim holds for all nodes then it holds for the root in particular, where $s = m - 1$. This implies $2^{vm} \leq v' \leq 2^{vm+m}$ which implies $v \leq (\log_2 v')/m \leq v + v/m$. Since $v < m$ and, thus, $v/m < 1$, this implies $v = \lfloor (\log_2 v')/m \rfloor$ as claimed.

So, let us prove that $2^{um} \leq u' \leq 2^{um+s}$ holds for all nodes. First for every leaf this claim is correct as we set $u' = 2^{um}$. Second, consider a node that used to have a $+$ -label and now has a \times -label. Let u_1 and u_2 be the values in T of the children of the node and let u'_1 and u'_2 be the values in T' of these children. Let s_1 and s_2 be the sums of the leaves in the subtrees of T rooted at the different children. The induction hypothesis states that

$$\begin{aligned} 2^{u_1 m} &\leq u'_1 \leq 2^{u_1 m + s_1} \quad \text{and} \\ 2^{u_2 m} &\leq u'_2 \leq 2^{u_2 m + s_2}. \end{aligned}$$

We have $u = u_1 + u_2$ since the node had a $+$ -label in T and we have $u' = u'_1 \cdot u'_2$ since the node now has a \times -label in T' . We also have $s = s_1 + s_2$. We can calculate as follows:

$$\begin{aligned} 2^{um} &= 2^{u_1 m + u_2 m} = 2^{u_1 m} \cdot 2^{u_2 m} \\ &\leq u'_1 \cdot u'_2 = u' = u'_1 \cdot u'_2 \\ &\leq 2^{u_1 m + s_1} \cdot 2^{u_2 m + s_2} = 2^{(u_1 + u_2)m + (s_1 + s_2)} = 2^{um+s}. \end{aligned}$$

Thus, $2^{um} \leq u' \leq 2^{um+s}$.

Third and finally, consider a node that used to have a \max -label and now has a $+$ -label. Let u_1 and u_2 and u'_1 and u'_2 and s_1 and s_2 be as before. Once more, we may assume

$$\begin{aligned} 2^{u_1 m} &\leq u'_1 \leq 2^{u_1 m + s_1} \quad \text{and} \\ 2^{u_2 m} &\leq u'_2 \leq 2^{u_2 m + s_2}. \end{aligned}$$

We have $u = \max\{u_1, u_2\}$ and $u' = u'_1 + u'_2$. Since everything is symmetric, we may assume that u_1 is the larger of the two values u_1 and u_2 . We calculate:

$$\begin{aligned} 2^{um} &= 2^{\max\{u_1, u_2\}m} = 2^{u_1 m} \leq 2^{u_1 m} + 2^{u_2 m} \\ &\leq u'_1 + u'_2 = u' = u'_1 + u'_2 \\ &\leq 2^{u_1 m + s_1} + 2^{u_2 m + s_2} \leq 2^{u_1 m + \max\{s_1, s_2\}} + 2^{u_1 m + \max\{s_1, s_2\}} \\ &= 2^{u_1 m + \max\{s_1, s_2\} + 1} \leq 2^{u_1 m + s_1 + s_2} = 2^{um+s}. \end{aligned}$$

We used the fact $\max\{s_1, s_2\} + 1 \leq s_1 + s_2$, which holds since all leaf labels are positive. \square

References

- [1] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, C. Rackoff, *Random Walks, Universal Sequences and the Complexity of Maze Problems*, Proc. 20. FOCS, 1979, 218–223.
- [2] C. Álvarez, B. Jenner, *A Very Hard Log-space Counting Classes*, TCS 107, 1993, 3–30.
- [3] E. Allender, M. Mahajan, *The Complexity of Planarity Testing*, Proc. 17. STACS, LNCS 1770, 2000, 87–98.
- [4] M. Ben-Or, R. Cleve *Computing Algebraic Formulas Using a Constant Number of Registers*, SIAM J. Comput. 21, 1992, 54–58.
- [5] H. Bodlaender, B. de Fluiter, *Parallel Algorithms for Series Parallel Graphs*, Proc. 4. ESA, LNCS 1136, 1996, 277–289.
- [6] A. Brandstädt, V. Bang Le, J. Spinrad, *Graph Classes: A Survey*, SIAM 1999.
- [7] S. Buss, S. Cook, A. Gupta, V. Ramachandran, *An Optimal Parallel Algorithm for Formula Evaluation*, SIAM J. Comput. 21, 1992, 755–780.
- [8] A. Chiu, G. Davida, B. Litow, *Division in logspace-uniform NC^1* , Theoretical Informatics and Applications 35, 2001, 259–275.
- [9] S. Cook, P. McKenzie, *Problems Complete for Deterministic Logarithmic Space*, J. Algo. 8, 1987, 385–394.
- [10] R. Duffin, *Topology of Series-Parallel Networks*, J. Math. Analysis Appl. 10, 1965, 303–318.
- [11] D. Eppstein, *Parallel Recognition of Series-Parallel Graphs*, Inf. & Comp. 98, 1992, 41–55.
- [12] W. Hesse, *Division Is in Uniform TC^0* , Proc. 28. ICALP, Springer LNCS 2076, 2001, 104–114.
- [13] X. He, Y. Yesha, *Parallel Recognition and Decomposition of Two Terminal Series Parallel Graphs*, Inf. & Comp. 75, 1987, 15–38.
- [14] A. Jakoby, M. Liškiewicz, and R. Reischuk *Space Efficient Algorithms for Series-Parallel Graphs*, Proc. STACS 2001, 339–352.
- [15] A. Jakoby and M. Liškiewicz, *Paths Problems in Symmetric Logarithmic Space*, Proc. 29. ICALP, Springer LNCS 2380, 2002, 269–280.
- [16] B. Jenner, K.-J. Lange, P. McKenzie, *Tree Isomorphism and Some Other Complete Problems for Deterministic Logspace*, publication #1059, DIRO, Université de Montréal, 1997.
- [17] R. Karp, V. Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, in: J. van Leeuwen (Ed.): Handbook of Theoretical Computer Science, Volume A, 1990, 869–941.
- [18] A. Nickelsen, T. Tantau, *The Complexity of Finding Paths in Graphs with Bounded Independence Number*, SIAM J. Comput. 34, 2005, 1176–1195.
- [19] O. Reingold, *Undirected st -connectivity in log-space*, Proc. STOC 2005, ACM Press, 376–385.
- [20] R. Tamassia and J. S. Vitter, *Parallel Transitive Closure and Point Location in Planar Structures*, SIAM J. Comput. 20, 1991, 708–725.

- [21] J. Valdes, R. Tarjan, E. Lawlers, *The Recognition of Series Parallel Digraphs*, SIAM J. Comput. 11, 1982, 298–313.