On the Parallel Complexity of Structural CSP Decomposition Methods

Zoltán Miklós

ABSTRACT. Constraint satisfaction problems are NP-complete in general, but they can be solved in polynomial time and also in parallel if their associated hypergraph is acyclic. It is often important for applications to solve also non-acyclic problems, thus finding larger tractable classes of CSPs is of high interest. Many structural decomposition methods have been suggested, which enable that the efficient techniques for acyclic hypergraphs can be applied to larger problem classes as well. For some of these classes, e.g. for bounded treewidth and bounded hypertreewidth also the recognition problem is in LogCFL, thus it is highly parallelizable. In this paper we show that all other tractable classes also posses this desirable property. In particular, we show, that recognizing hypergraphs with bounded biconnected width is in LogCFL. We also prove that recognizing hypergraphs with bounded cycle cutset width is feasible in L.

1 Introduction

Many important problems in artificial intelligence, database theory and operations research can be formulated as constraint satisfaction problems. A constraint (S_i, R_i) consists of a constraint scope, i.e. a set of variables S_i and a constraint relation R_i containing the allowed combinations of values of the variables. A constraint satisfaction problem (CSP) is a set of constraints $(S_1, R_1), \ldots, (S_r, R_r)$, whose variables may overlap. A solution to a CSP is an assignment of the variables, such that they simultaneously satisfy all of the relations. Deciding whether there exists such an assignment is often referred as solving the CSP.

We can associate a hypergraph H to a CSP in the following way: the vertices of H correspond to the variables of the CSP while the hyperedges correspond to the constraints, such that whenever a variable occurs in a constraint relation, the corresponding vertex is contained in the corresponding hyperedge.

Solving constraint satisfaction problems is NP-complete in general, but CSPs whose associated hypergraph is acyclic can be solved in polynomial time and also in parallel. It is often important for applications to solve also non-acyclic problems, thus finding larger tractable classes of CSPs is of high interest. Many structural decomposition methods have been suggested, which enable that the efficient techniques for acyclic hypergraphs can be applied to larger problem classes as well. Solving a CSP with a decomposition involves the following steps.

1. Recognizing structures of k-bounded width,
2. constructing a decomposition tree (of width k),
3. solving the CSP using the decomposition tree.

If a decomposition tree for a CSP is given, then the well-known algorithm by Yannakakis [28] can be used to solve the problem in polynomial time. Parallel algorithms have been suggested by Gottlob et al. in [11] for an equivalent problem, for the evaluation problem of Boolean conjunctive queries. In [11], the authors also show that solving CSPs with a given decomposition tree is complete for LogCFL.

In this paper we concentrate on Step 1, on the problem of recognizing bounded width structures: since efficient and parallel algorithms exists for solving CSP if the decomposition tree is given, we study the question, how to recognize such structures. For the bounded width classes we study here, tractable algorithms are already known. Our contribution is to show containment in complexity classes, for which effective parallel algorithms exists.

For some of the bounded-width classes, e.g. for bounded treewidth (independently, by Wanke [27] and by Lautermann [18]) and bounded hypertreewidth (Gottlob et al. [12]), the recognition problem is known to be in the low complexity class LogCFL. Bounded width structures defined by subedge-based decompositions with a logspace computable subedge function can also be recognized in LogCFL, see [13]. Thus, also bounded component hypertreewidth hypergraphs can be recognized in LogCFL.

We study other known tractable classes and demonstrate how the proof techniques of [12] can be applied to these classes as well. In particular, we study hypergraphs with bounded biconnected cut width and show that their recognition problem is in LogCFL. For bounded cycle cutset and cycle hypercutset width we give an even better upper bound, we show that they can be recognized in logspace. Bounded hinge width and bounded spread cut width can also be tested in LogCFL.

The paper is organized as follows. In Section 2 we give some definitions and we recall some relevant results from complexity theory. In Section 3, we study biconnected components, in Section 4 we analyze the generalized hypertreewidth recognition problem of bounded dimension hypergraphs. In

Section 5 we show that testing bounded cycle cutset-width is in L. In Section 6 we outline the proofs for hinge decomposition and spread cut decomposition, while Section 7 concludes the paper.

2 Preliminaries

2.1 CSP decompositions

In [10], Gottlob et al. compare tractable CSP decomposition methods and study which of the decomposition concepts capture larger classes of hypergraphs. Cohen et al. [4] give a unified theory for structural CSP decompositions, they show, that all of the decomposition methods follow the same definition scheme, they only differ in one condition, characteristic for the particular method. The Figure 1 is adapted from [10], and also reflects he latest development in the field.

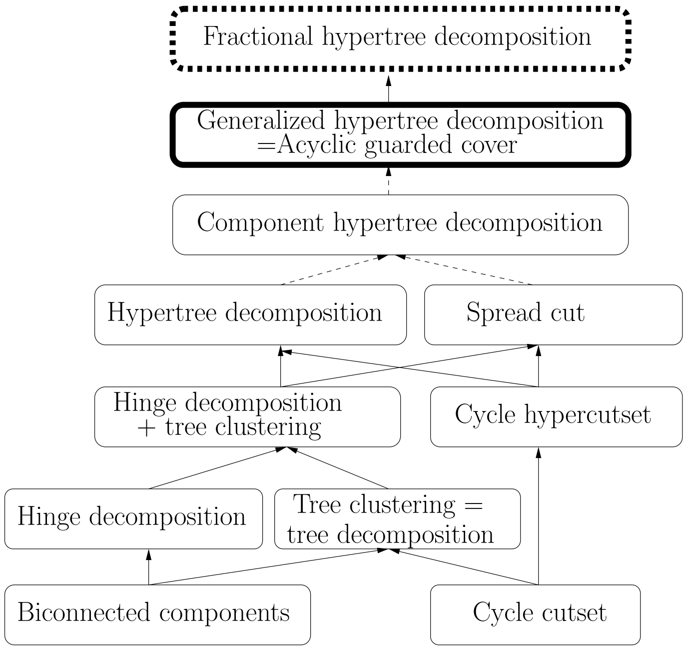

Figure 1. Hierarchy of decomposition methods.

We recall that an arrow on the figure has the following meaning: if there is an arrow from decomposition method A to method B, then there exists structures with bounded width defined by B, but unbounded width defined by A. Arrows with dotted line have a different semantic: if there is an arrow form A to B with a dotted line then the width of hypergraphs defined by the method B is less or equal than the width defined by the method A. By the results in [2], hypergraphs with bounded generalized hypertreewidth cannot have unbounded hypertreewidth:

THEOREM 1. *([2])*
For a hypergraph H, $ghw(H) \leq hw(H) \leq 3ghw(H) + 1$.

Generalized hypertree decomposition is depicted with a bold line on the figure, because –as it was shown recently in [13]– testing whether a hypergraph has generalized hypertreewidth at most 3 is NP-complete. Fractional hypertree decompositions were introduced by Grohe and Marx [15]. It is an open problem whether recognizing bounded fractional hypertreewidth structures is tractable, therefore we depicted the class with a dotted borderline. Given a fractional hypertree decomposition of a CSP, it can be solved in polynomial time. The polynomial time algorithm for constraint solving defined in [15] however is different from the algorithms in [28] and [11], and it is not known to be parallelizable.

2.2 LOGCFL

LogCFL is the class of decision problems logspace reducible to a context-free language. There is a number of problems, including some very natural problems, which are complete for LogCFL, e.g. Greibach's hardest context-free language [14], the problem of evaluating a Boolean acyclic conjunctive query over a relational database [11], computing pure Nash equilibria of certain games [8], evaluating a positive core XPath query over an XML document [9] and the uniform membership problem for nondeterministic tree automata [19].

The relationship between LogCFL and other well-known complexity classes is summarized as follows:

$$\mathrm{AC}^0 \subsetneq \mathrm{NC}^1 \subseteq \mathrm{L} = \mathrm{SL} \subseteq \mathrm{NL} \subseteq \mathrm{LogCFL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2 \subseteq \mathrm{P} \subseteq \mathrm{NP}$$

Here, L is logspace, SL is symmetric logspace, NL is nondeterministic logspace, P is polynomial time, NP is nondeterministic polynomial time. AC^i is the class of languages recognized by a logspace-uniform circuit family of Boolean circuits of depth $O(log^i n)$, while NC^i denotes the class of languages recognized by a logspace-uniform circuit family of Boolean circuits of depth $O(log^i n)$ having bounded fan-in. For an overview of this concepts, see e.g. [11].

Since LogCFL $\subseteq$ AC^1 $\subseteq$ NC^2, the problems in LogCFL are highly parallelizable. Figure 2 summarizes the computational models characterizing the class LogCFL, compared with polynomial time. In this models, LogCFL is a basic computational resource restricted class, which also underlines its central place among complexity classes.

PTIME	LogCFL
Alternating Turing machine using logarithmic space [22]	Alternating Turing machine using logarithmic space *with a polynomial size witness tree* [24]
Nondeterministic Turing machine using logarithmic space with an auxiliary pushdown [17]	Nondeterministic Turing machine using logarithmic space with an auxiliary pushdown *halting in polynomial time* [25]
Logspace-uniform family of unbounded Boolean circuits [22]	Logspace-uniform family of *semi-unbounded* Boolean circuits *of logarithmic depth* [26]

Figure 2. Computational models: PTIME vs. LogCFL

2.3 On logspace computations

The following breakthrough result of Reingold simplifies our investigations.

THEOREM 2. *([23])* $L = SL$

We also use the following property of logspace computations.

THEOREM 3. *(Nisan, Ta-Shma [21])* $L^{SL} = SL^{SL} = SL$

ConnectedComponentsEqual is the following decision problem:
Input: Two undirected graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$
Question: Is the number of connected components of G_1 equal to the number of connected components G_2?

THEOREM 4. *([3, 21] and Theorem 2)*
ConnectedComponentsEqual *is complete for L.*

A direct consequence of the above Theorem is that we can decide in logarithmic space whether a graph has exactly 2 connected components, by using a "dummy" graph containing only two isolated vertices.

3 Biconnected components and LOGCFL

Biconnected components were introduced in [7].[1] In a hypergraph $H = (V, E)$ a vertex v is separating, if removing v from H, the number of connected components of H increases. A biconnected component C is a set of vertices which has no separating vertex.

[1] In [7], the concept was defined for graphs, whereas here we concentrate on hypergraphs.

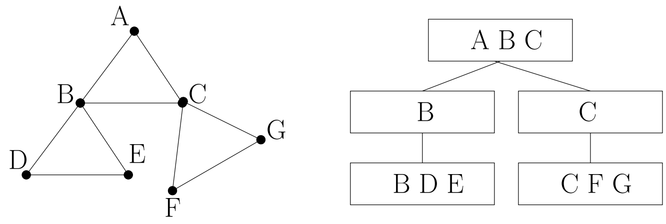

Figure 3. Biconnected components decomposition.

Biconnected decomposition $\langle T, \chi \rangle$ of H is a tree with a labeling function that associates either a biconnected component of H or a singleton vertex to the nodes of T, such that there is an edge btw. two nodes p and q in T, if $\chi(p)$ is a biconnected component, and $\chi(q)$ is a separating vertex contained in $\chi(p)$. Without loss of generality we can assume that the root node of a biconnected decomposition is corresponds to a biconnected component.

On Figure 4 we give a high level description of the algorithm `k-biconnected` that decides whether a hypergraph has k-bounded biconnected components width. The algorithm can be effectively implemented on an alternating Turing machine, as we show later (Theorem 7). The following lemmas show that the algorithm `k-biconnected` indeed recognizes hypergraphs with biconnected width k. Note, that the description of the algorithm refers to two different types of components: to biconnected components (a set of vertices without separating vertex) and to connected components of $H[V \setminus S]$ (the connected components of the induced subgraph on vertices $V \setminus S$).

LEMMA 5. *For any hypergraph H, such that biconnected-width of H is at most k, `k-biconnected` accepts H.*

Proof. Let $\langle T, \chi \rangle$ a biconnected decomposition of H. We show, that there exists an accepting computation tree τ for `k-biconnected`. The accepting computation and the tree τ can be constructed using the decomposition $\langle T, \chi \rangle$ as follows.

For the initial call of $k - biconn(V, \emptyset, \emptyset)$, we choose the set S in Step 1 as $\chi(root(T))$. For a call $k - biconn(C_R, R, W)$, where R corresponds to a biconnected component at some vertex r of T, and s is a descendant node of r such that the only node on the unique path from r to s in T is p that corresponds to a vertex P separating the biconnected components R and S, where S is the biconnected component corresponding to s, then we choose S in Step 1. and we choose the vertex P as the parameter in the procedure call at step 4.

We use induction to show that in this way we defined an accepting computation.

ALTERNATING ALGORITHM `k-biconnected`
Input: hypergraph H, (non-empty)
Result: Accept, if the biconnected width of H is at most k, **Reject** otherwise.
Procedure $k - biconn(C_R : ConnectedComponent, R : SetOfVertices, W : Vertex)$
begin
1) **Guess** a set S of size at most k
2) **Check** whether S is a biconnected component and whether $W \in S$
3) **If** the check above fails **Then Halt** and **Reject; Else**
Let $\mathcal{C} = \{(C, Z) \mid C$ is a connected component of $H[V \setminus S]$, Z is a separating vertex such that $Z \in S$ and $C \subseteq C_R\}$
4) **If, for each** $(C, Z) \in \mathcal{C}$, $k - biconn(C, S, Z)$
then Accept else Reject
end;
begin $(*Main*)$
Accept if $k - biconn(V, \emptyset, \emptyset)$
end.

Figure 4. Recognizing k-bounded biconnected component width hypergraphs

Basis: If the tree contains only a root node, then clearly, we define an accepting computation.
Induction step: Assume that the computation we defined above reaches the vertex r. Let s be a node of T such that s is a descendant node of r such that the only node on the unique path from r to s in T is p that corresponds to a vertex P, separating the biconnected components R and S. By the choice of S in Step 1, and by the choice of the parameters at the procedure call 4, the check in Step 2 does not fail, as S is a biconnected component of size at most k with $P \in S$ as a separating vertex. Thus, the computation also reaches the vertex s. Therefore, by induction, the defined computation is accepting. ■

LEMMA 6. *For any hypergraph H, such that* `k-biconnected` *accepts H, the biconnected-width of H is at most k.*

Proof. Let $\langle F, \rho \rangle$ be a witness tree of an accepting computation of `k-biconnected` with input H. We can construct a biconnected decomposition $\langle T, \chi \rangle$ of with k as follows.

For the root node of F we define a node of T and we label it with $\rho(root(F))$. This node is the root of T. For each node r in F we add a node p_r with label $\chi(p_r) = \rho(r)$ in T. For each node r of F, let $children(r)$ denote the child nodes of r in F. For each vertex in $\{v \mid v \in \rho(r) \cap \rho(s)$, where $s \in children(r)\}$ we define a vertex p_v with label $\chi(p_v) = \{v\}$ in T, and add as a child node of p_r in T. If s is a child node of r and $v \in \rho(r) \cap \rho(s)$, then p_s is a child node of p_v in T.

We prove that the tree constructed in this way indeed a biconnected decomposition of width at most k. We have to show the following.

1. If s is a child node of r in F, then the set $\rho(r) \cap \rho(s)$ contains exactly one vertex Z, which is a separating vertex in H and $Z \in \rho(r)$ and $Z \in \rho(r)$.
2. T is a tree graph.
3. $\langle T, \chi \rangle$ is a biconnected decomposition of width at most k.

1\. Since $\langle F, \rho \rangle$ be a witness tree of an accepting computation, both $\rho(r)$ and $\rho(s)$ are biconnected components of H and there is a separating vertex $Z \in \rho(r)$, which is also in $\rho(s)$, since the test in Step 2 does not fail. Indirectly, if for two different vertices Z_1 and Z_2, the containment $\{Z_1, Z_2\} \subseteq \rho(r) \cap \rho(r)$ holds, then neither of them is a separating vertex. Contradiction.

2\. We constructed T in a way that each node has exactly one parent node, therefore it is a tree.

3\. The tree T is labeled with biconnected components and singleton vertices. By the construction, if $\chi(p)$ is a biconnected component and q is a child of p then $\chi(q)$ is a separating vertex contained in $\chi(p)$. The width of the decomposition is clearly at most k. Thus the tree is indeed a biconnected decomposition of H. ■

THEOREM 7. *Deciding whether a hypergraph H has biconnected width at most k is feasible in* LogCFL.

Proof. The theorem follows from Ruzzo's characterization of LogCFL [24], and from the Lemmas 5, 6, since `k-biconnected` can be implemented on a logspace alternating Turing machine with polynomially bounded tree size in the following way (see also Gottlob et al. [12], Lemma 5.15.) We can represent biconnected components of size at most k in logarithmic space. For a biconnected component S ($\leq k$), also the connected components of $V \setminus S$ can be represented in logspace: we need to store the set S, which is of size k and one vertex in the component. For each vertex of H, we can test in logarithmic space whether a vertex is in a connected component. Before the procedure call at Step 4. we have to test whether a vertex is a separating vertex in H. This test can also be performed by a logspace algorithm, by Theorem 4. ■

4 Bounded dimension hypergraphs

A hypergraph has bounded dimension, if the size of its hyperedges bounded by a constant d. The following result is in contrast with the NP-completeness result of testing generalized hypertreewidth [13]. The theorem is easy to prove, nevertheless it shows an important parametrization of the generalized hypertreewidth test, which not only makes the problem tractable, but also parallelizable.

THEOREM 8. *Let $H = (V, E)$ be a bounded dimension hypergraph H, i.e. $d = max\{|e| \mid e \in E\}$. Testing whether H has bounded generalized hypertreewidth, i.e. $ghw(H) \leq k$ is in* LogCFL.

Proof. Let us given a hypergraph H. Let H^* denote the hypergraph obtained from H by adding all possible subedges of H, i.e. $H^* = H \cup \{e' \mid \exists e \in E, e' \subseteq e\}$. It has been observed by Adler [1], that $ghw(H) = hw(H^*)$. Since H has bounded dimension, H^* can be computed in logspace (because each edge has 2^d –a constant number of– subedges), and testing whether $hw(H^*) \leq k$ is feasible in LogCFL, see [12]. ■

5 Cycle cutset and cycle hypercutset in L

Cycle cutset [6] (a.k.a feedback vertex set) of a hypergraph $H = (V, E)$ is a set of vertices, such that the induced subhypergraph on vertices $V \setminus S$ is acyclic. The cycle cutset width of H is 0 if H is acyclic, otherwise the minimal size over all of its cycle cutsets.

THEOREM 9. *Deciding whether a hypergraph $H = (V, E)$ has cycle cutset width at most k is complete for L under NC^1 reductions.*

Before we prove this result we show, recognizing acyclic hypergraphs is also complete in logspace logspace.

THEOREM 10. *Deciding whether a hypergraph is acyclic is complete in L under NC^1 reductions.*

Proof. Gottlob et al. [11] have shown that testing hypergraph acyclicity is in SL, therefore, by theorem 2 it is in L. The hardness follows form the fact, that testing graph acyclicity is is hard for deterministic logspace [5], under NC^1 reductions. An acyclic graph is at the same time an acyclic hypergraph. ■

Proof. (of Theorem 9) *Containment.* A logspace algorithm can go through all subsets of V of size k and test whether the removal of the subset makes the hypergraph acyclic. Hypergraph acyclicity can be tested in logspace oracle, by Theorem 10. Thus, by Theorems 2 and 3, testing bounded cycle cutset width of a hypergraph is feasible in logspace.

Hardness. Testing, whether a hypergraph H is acyclic, i.e. whether H has cycle cutset width 0, is hard for L, see Theorem 10. ■

A simple modification of the concept cycle cutset is a cycle hypercutset, where we remove hyperedges, instead vertices. A cycle hypercutset of a hypergraph $H = (V, E)$ is a set of edges S, such that the hypergraph induced by the vertices of $V \setminus vertices(S)$ is acyclic. The cycle hypercutset width of a hypergraph is the minimum cardinality over all of its possible cycle hypercutsets.

THEOREM 11. *Deciding whether a hypergraph $H = (V, E)$ has cycle hypercutset width at most k is complete for L under NC^1 reductions.*
Proof. Analogous to the proof of Theorem 9. ■

6 Hinge decomposition and spread cut decomposition

Using similar techniques as in Section 3, which are adaptations of the technique first applied in Gottlob et al. [12] to the particular problems, we can obtain similar results for hinge decompositions and spread cut decompositions. Here we only outline the results, the full proofs can be found in the full version of this paper and in [20].

Hinge decompositions were introduced in [16].

THEOREM 12. *Deciding whether a hypergraph H has hinge width at most k is feasible in* LogCFL.
Proof. (sketch) The nodes of a hinge decomposition of H correspond to some of the hinges of H. Given a set S of at most k hyperedges, we can test in logarithmic space, whether S is a hinge of H. We can design a nondeterministic algorithm, running on an alternating Turing machine, using only logarithmic space and having a polynomial size witness tree. ■

Spread cut decompositions were introduced recently by Cohen et al. [4].

THEOREM 13. *Deciding whether a hypergraph H has spread cut width at most k is feasible in* LogCFL.
Proof. (sketch) We can relate a spread cut decomposition of H of width at most k to a hypertree decomposition of an another hypergraph, $H \cup \Delta_k$, where Δ_k is a set of subedges allowed by a spread cut decomposition. A generalized hypertree decomposition D in normal form is a spread cut decomposition of H, iff D is a hypertree decomposition of $H \cup \Delta_k$ satisfying some additional conditions: the components respect labels, there are no unbroken components, for definitions see [4]. The set Δ_k can be computed in logspace and the additional conditions can also be tested in logspace, thus an alternating logspace algorithm can be designed in a similar way, as in [12]. ■

7 Conclusion and future work

We studied upper bounds for the complexity of various recognition problems for tractable CSP decomposition methods and we have demonstrated that all of these problems are contained in low parallel complexity classes. In all of the cases we were able to design an nondeterministic recognition algorithm, which can be implemented on an alternating Turing machine using only logarithmic space. Furthermore, in each case, the ATM has only polynomial size witness tree. A more detailed discussion and the complete proofs can be found in the full version of this paper and in [20].

To our best knowledge, no better upper bounds are known for the recognition problems. We leave it as an open question whether for any of the LogCFL cases, also LogCFL-hardness holds.

We believe that there is a unified theory of the CSP decompositions, similar to [4], which also gives deeper explanations for containment of the recognition problems in LogCFL. This theory could rely on the connection between tractable CSP decompositions, and context-free hyperedge replacement grammars, as presented e.g. in [18], but this connection needs to be further explored.

Acknowledgements

The author was partially supported by the Wolfgang Pauli Institute, in the project "Foundations of Knowledge and Information Handling".

BIBLIOGRAPHY

[1] Isolde Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, 2004.

[2] Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree-width and related hypergraph invariants. In *Proceedings of the 3rd European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'05)*, volume AE of *DMTCS Proceedings Series*, pages 5–10, 2005.

[3] C. Alvarez and R. Greenlaw. A Compendium of Problems Complete for Symmetric Logarithmic Space. *Computational Complexity*, 9:73–95, 2000.

[4] David Cohen, Marc Gyssens, and Peter Jeavons. A unifying theory of structural decompostions for the constraint satisfaction problems. In *Complexity of Constraints*, number 06401 in Dagstuhl Seminar Proceedings, http://drops.dagstuhl.de/opus/volltexte/2006/801, 2006.

[5] Stephen A. Cook and Pierre McKenzie. Problems Complete for Deterministic Logarithmic Space. *Journal of Algorithms*, 8(5):385–394, 1987.

[6] Rina Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[7] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, 1985.

[8] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Pure Nash Equilibria: Hard and Easy Games. *Journal of Artificial Intelligence Research*, 24:357–406, 2005.

[9] Georg Gottlob, Christoph Koch, Reinhard Pichler, and Luc Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM*, 52(2):284–335, March 2005.

[10] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artifical Intelligence*, 124(2):243–282, 2000.

[11] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.

[12] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences (JCSS)*, 64(3):579–627, May 2002.

[13] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. In *Proceedings of the 26th ACM SIGMOD-SICACT-SIGART Symposium on Principles of Database Systems*, 2007.

[14] S. H. Greibach. The hardest context-free language. *SIAM Journal on Computing*, 2:304–310, 1973.

[15] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[16] Marc Gyssens, Peter G. Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, March 1994.

[17] O.H. Ibarra. Characterizations of some tape and time complexity classes of turing machines in terms of multihead and auxiliary stack automata. *J. Comput. System Sci.*, 5(2):88–117, 1971.

[18] Clemens Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27(5):399–421, 1990.

[19] Markus Lohrey. On the parallel complexity of tree automata. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes In Computer Science*, pages 201–215. Springer, 2001.

[20] Zoltán Miklós. *Understanding tractable constraints*. PhD thesis, University of Oxford, 2007.

[21] Noam Nisan and Amnon Ta-Shma. Symmetric Logspace is Closed Under Complement. *Chicago Journal of Theoretical Computer Science*, 1, 1995.

[22] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesledy, 1994.

[23] Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of computing*, pages 376–385, 2005.

[24] W. L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences (JCSS)*, 21(2):218–235, 1980.

[25] I. H. Sudborough. On the tape complexity of deterministic context-free laguages. *Journal of the ACM*, 25(3):405–414, 1978.

[26] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences (JCSS)*, 43(2):380–401, 1991.

[27] Egon Wanke. Bounded Tree-Width and LOGCFL. *Journal of Algorithms*, 16(3):470–491, 1994.

[28] M. Yannakakis. Algorithms for acyclic database schemes. In C. Zaniolo and C. Delobel, editors, *Proceedings of the International Coference on Very Large Data Bases (VLDB'81)*, pages 82–94, Cannes, France, 1981.

Zoltán Miklós
University of Oxford and
Technische Universität Wien
zoltan.miklos@comlab.ox.ac.uk