Structural Decomposition Methods and What They are Good For

Markus Aschinger, Conrad Drescher, Georg Gottlob, Peter Jeavons, Evgenij Thorstensen

Structural Decomposition Methods and What They are Good For

Markus Aschinger[1], Conrad Drescher[1], Georg Gottlob[1,2], Peter Jeavons[1], and Evgenij Thorstensen[1]

1 Computing Laboratory, University of Oxford
2 Oxford Man Institute of Quantitative Finance
firstname.lastname@comlab.ox.ac.uk

Abstract

This paper reviews structural problem decomposition methods, such as tree and path decompositions. It is argued that these notions can be applied in two distinct ways: Either to show that a problem is efficiently solvable when a width parameter is fixed, or to prove that the unrestricted (or some width-parameter free) version of a problem is tractable by using a width-notion as a mathematical tool for directly solving the problem at hand. Examples are given for both cases. As a new showcase for the latter usage, we report some recent results on the Partner Units Problem, a form of configuration problem arising in an industrial context. We use the notion of a path decomposition to identify and solve a tractable class of instances of this problem with practical relevance.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.12

1 Introduction: Treewidth and Other Notions of Width

Tree decompositions [52, 6] and their variants and generalizations [44] constitute a significant success story of Theoretical Computer Science. In fact, tree decompositions and polynomial algorithms for bounded treewidth constitute one of the most effective weapons to attack NP hard problems, namely, by recognizing and efficiently solving large classes of tractable problem instances. Structural problem decompositions such as treewidth are thus closely related to fixed-parameter tractability [19, 37].

Tree and Path Decompositions. Formally, a *tree decomposition* of a graph $G = (V, E)$ is a pair $\mathcal{P} = (T, \chi)$ such that $T = (W, F)$ is a tree or forest, and where the function χ associates to every $w \in W$ a subset $B \subseteq V$ such that

(1) for every vertex v in V there is a vertex $w \in W$ with $v \in \chi(w)$;
(2) for every edge (v_1, v_2) in E there is a vertex $w \in W$ with $\{v_1, v_2\} \subseteq \chi(w)$; and
(3) for every vertex v in V the set $\{w \in W \mid v \in \chi(w)\}$ induces a subtree of T.

Condition (3) is called the connectedness condition. The subsets B associated with the vertices of W are called bags. The width of a tree decomposition is $\max_{w \in W}(|\chi(w)| - 1)$. The *treewidth* $tw(G)$ of G is the minimum width over all its tree decompositions.

A *path decomposition* of a graph is a tree decomposition where $T = (W, F)$ actually consists of a simple root-leaf path. The pathwidth $\mathrm{pw}(G)$ of a graph is the minimum width over all its path decompositions.

Several variants and generalizations of treewidth have been introduced, for an overview see [44]. For example, the notion of treewidth is easily generalized from graphs to finite structures. A *finite structure* $\mathcal{A}$ consists of a domain A, and relations $R_1, \ldots, R_k$ of arities $a_1, \ldots, a_k$, respectively. Each such relation R_i consists of a set of tuples $(r_1, \ldots, r_{a_i}) \in R$ where $r_1, ..., r_{a_i} \in A$. A graph $G = (V, E)$ corresponds to a finite structure whose domain is V, with a binary relation E encoding the edges. If G is undirected, then E contains both pairs (v, w) and (w, v) for each

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 12–28

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


edge $\{v, w\}$ of G. Undirected graphs are thus represented as special cases of arbitrary (possibly directed) graphs. The *Gaifman graph* of a structure $\mathcal{A}$, is the undirected graph $G(\mathcal{A})$ whose set of vertices is the domain A of $\mathcal{A}$ and where there is an edge $\{a, b\}$ iff $a, b \in A$ and $a \neq b$, and there exists a tuple in one of the relations of $\mathcal{A}$ in which a and b occur jointly. A *tree decomposition of the structure* $\mathcal{A}$ is a tree decomposition for the Gaifman graph, $G(\mathcal{A})$. The *treewidth* $tw(\mathcal{A})$ *of a structure* $\mathcal{A}$ is defined accordingly, i.e., $tw(\mathcal{A}) = tw(G(\mathcal{A}))$. Similarly, the pathwidth $pw(A)$ is defined as $pw(G(\mathcal{A}))$. The treewidth $tw(C)$ (pathwidth $pw(C)$) of a class C of finite structures is the maximum over all $tw(\mathcal{A})$ ($pw(\mathcal{A})$) for $\mathcal{A} \in C$. A tree decomposition of a hypergraph H is a tree decomposition of the *primal graph* $G(H)$ of H, which has the same vertices as H and an edge between each pair of vertices that jointly occur in a hyperedge of H.

It is NP hard to determine the treewidth of a structure $\mathcal{A}$. However, for each fixed k, checking whether $tw(\mathcal{A}) \leq k$, and if so, computing a tree decomposition for $\mathcal{A}$ of optimal width, is achievable in linear time [5], and was recently shown to be achievable in logarithmic space [20]. Even though the multiplicative constant factor of Bodlander's linear algorithm [5] is exponential in k, there are algorithms that find exact tree decompositions in reasonable time or good upper approximations in many cases of practical relevance, see for example [7, 8] and the references therein.

The treewidth of a graph or relational structure is an invariant that can be used as a parameter to define infinite classes of graphs (or structures) related to problem instances. Many NP hard problems of practical relevance can be solved in polynomial time on instances of bounded treewidth and some are actually fixed-parameter tractable with respect to treewidth. Given that bounded pathwidth implies bounded treewidth, these results hold a fortiori for bounded pathwidth. The notion of treewidth is at the base of strong meta-theorems such as Courcelle's Theorem [13], which states that any problem expressible in monadic second-order logic (MSO) over structures of bounded treewidth can be solved in linear time (or, by a recent result, in logarithmic space [20]). Many problems, e.g. the 3-colorability of graphs, are very easily expressed in terms of MSO, and thus Courcelle's theorem turns out to be a very effective tool for obtaining tractability results.

Hypergraph Decompositions The structure of a computational problem is sometimes better described by a hypergraph than by a graph. Therefore, various width-notions for hypergraphs have been defined and studied, and often these are more effective than simply considering the associated primal graph. In particular, the notion of *hypertree width*, which is based on *hypertree decompositions* [24, 31, 1], is an appropriate measure for the degree of acyclicity of a hypergraph. Bounded hypertree width generalizes the concept of α*-acyclic hypergraphs* developed by Fagin [21]. In essence, a hypertree decomposition of width k for a hypergraph H can be obtained from a suitable tree decomposition of $G(H)$ by covering each bag with at most k hyperedges of H. However, under this definition, which actually defines the concept of *generalized hypertree width*, $ghw(H)$, of a hypergraph H, it is unfortunately NP-hard to determine whether a hypergraph has width $\leq k$ for $k = 3$ [32]. Therefore, to define the actual concept of hypertree width, an additional condition is imposed that ensures the tractability of computing hypertree decompositions of low width (for details, see [24]). Bounded hypertree width is strictly more general than bounded treewidth because there exist families of hypergraphs with bounded hypertree width whose treewidth is unbounded.

A number of practically relevant problems become tractable for instances whose associated hypergraphs have low hypertree width. Examples are constraint satisfaction problems (CSPs) [16, 29], see Section 3.1 for a definition, and combinatorial auctions [26]. Very roughly, CSPs of bounded hypertree-width are *loosely constrained* and therefore tractable. CSPs may also become tractable because their associated hypergraphs are –in a precise sense– *tightly constrained*. Formally this was captured by the entropy-based measure of *fractional edge cover* [39].

CSPs whose associated hypergraphs are of bounded fractional edge cover number are tightly constrained and can be solved in polynomial time. Combining hypertree decomposition with fractional edge covers yields *fractional hypertree decompositions* [39], a decomposition method that is more general than both hypertree decompositions and fractional edge covers. While hypertree decompositions will be considered again in Section 3.1, this paper mainly deals with tree and path decompositions.

Structure of the Paper. The rest of the paper is structured as follows. In the next section, we present a taxonomy of the main uses of tree decomposition in Computer Science. We will distinguish between two main categories (categories 1 and 2) and four sub-categories (1.a, 1.b, 2.a, and 2.b) of applications of tree decompositions. In Section 3, we give examples of problems that fall into the categories 1.a, 1.b, and 2.a, respectively. In section 4, we describe a relevant version of the Partner Unit Problem (PUP) and report about our recent result [3] showing that the problem falls in category 2.b and is therefore tractable.

2 Taxonomy of Main Uses of Tree Decompositions

One may distinguish between two main usages of tree decompositions, each of which can be subdivided in turn into two sub-cases. The following taxonomy will be illustrated with concrete examples in the next sections. When we speak about the treewidth of a problem instance, we mean the treewidth of some graph associated with the instance. Obviously, for each concrete problem, one has to indicate what this graph is, and, whenever necessary, how it can be obtained from the instance.

1. *Proving a problem tractable for instances of bounded treewidth.* This is probably the main use of treewidth. The idea is that many practically relevant classes of inputs actually have bounded treewidth, and that for such classes, a polynomial algorithm can be designed. We distinguish between general tractability results and fixed-parameter tractability (FPT) results:
   - **1.a** *General tractability results.* We are able to show that the problem becomes tractable, but the best known polynomial algorithms are of complexity $\Omega(n^{f(k)})$, where $\lim_{k\to\infty} f(k) = \infty$. In many such cases it can actually be proven that the problem is *fixed-parameter intractable* with respect to the treewidth parameter k. A typical problem in this category, is the CSP problem, which we will discuss in more detail in the next section.
   - **1.b** *Fixed-parameter tractability results.* This is the case if there exists a function f such that the problem is solvable in time $f(k) \times n^{O(1)}$ on instances of treewidth $\leq k$. If the problem is actually solvable in time $f(k) \times O(n)$, then we speak about *fixed-parameter linearity*. In particular, fixed-parameter linearity results can be obtained whenever Courcelle's Theorem can be applied. To illustrate this, we will discus the MULTICUT problem in the next section.
2. *Using treewidth as a tool for proving a problem to be generally tractable (on all instances).* There are a number of cases where treewidth is used as a tool in general tractability proofs. Again, we may distinguish between two sub-cases:
   - **2.a** *Proving that a problem is tractable both for small and large treewidth.* For some problems having some associated implicit or explicit parameter c, it can be shown that there exists a function f such that the problem is tractable both for instances having treewidth $\leq f(c)$ and also for instances having treewidth $> f(c)$. The tractability proof for low treewidth is usually completely different from the tractability proof for high treewidth. As an example for this category, we will review the problem of checking whether a loop-free undirected graph has a cycle of length $0 \mod c$, where c is a fixed constant. This problem was shown to be tractable by Thomassen [53].

2.b *Proving that all yes-instances of a problem have bounded treewidth and that the problem is tractable for this reason.* Here, for some explicit or implicit constant parameter c associated with the problem, one is able to find a function t, such that all yes-instances of the problem must have treewidth $\leq t(c)$. Moreover, one shows that for instances of bounded treewidth the problem is polynomially solvable. Note that this is actually a very special case of Case 2.a. In fact, this means that the high-treewidth case is trivially tractable, because all instances of high treewidth are no-instances. As an example for this category, we will report in Section 4 about a new result related to the *Partner Unit Problem (*PUP*)*, a problem of industrial relevance [22].

3 Examples for Tractability Results Based on Bounded Treewidth

As announced, this section contains examples for the usages of bounded treewidth as described under categories 1.a, 1.b, and 2.a. We start with a very brief review of the Constraint Satisfaction Problem (CSP) in Section 3.1. We then describe the MULTICUT problem and illustrate how a nice generalization of Courcelle's Theorem can be used to show that MULTICUT is FPT on instances of bounded treewidth. Finally, in Section 3.3 we review Thomassen's famous result stating that it can be determined in polynomial time whether a graph has an even cycle, and more generally, whether a graph has a cycle of length $0 \mod c$ where c is a fixed constant. Our presentation of all these problems and results is necessarily very brief, but we include references to literature containing a full treatment and many more results.

3.1 The Constraint Satisfaction Problem

The efficient solution of Constraint Satisfaction Problems (CSPs) has for many years been an important goal of AI research and of related disciplines, in particular, Operations Research and Database Theory.

An instance of a *constraint satisfaction problem (CSP)* (also *constraint network*) is a triple $I = (Var, U, \mathcal{C})$, where Var is a finite set of variables, U is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_i is a pair (S_i, r_i), where S_i is a list of variables of length m_i called the *constraint scope*, and r_i is an m_i-ary relation over U, called the *constraint relation*. (The tuples of r_i indicate the allowed combinations of simultaneous values for the variables S_i). A *solution* to a CSP instance is a substitution $\theta : Var \longrightarrow U$, such that for each $1 \leq i \leq q$, $S_i\theta \in r_i$. The problem of deciding whether a CSP instance has any solution is called *constraint satisfiability (CS)*. Many well-known problems in Computer Science and Mathematics can be formulated as CSPs. For example, the famous problem of *graph three-colorability (3COL)*, i.e., deciding whether the vertices of a graph $G = \langle Vertices, Edges \rangle$ can be colored by three colors (say: red, green, blue) such that no edge links two vertices having the same color, can be formulated as a CSP as follows. The set Var contains a variable X_v for each vertex $v \in Vertices$. For each edge $e = \langle v, w \rangle \in Edges$, the set $\mathcal{C}$ contains a constraint $C_e = (S_e, r_e)$, where $S_e = \langle X_v, X_w \rangle$ and r_e is the relation $r_{\neq}$ consisting of all pairs of different colors, i.e., $r_{\neq} = \{\langle red, green \rangle, \langle red, blue \rangle, \langle green, red \rangle, \langle green, blue \rangle, \langle blue, red \rangle, \langle blue, green \rangle\}$.

It is well-known, and easy to see, that Constraint Satisfiability is an NP-complete problem. Membership in NP is obvious. NP-hardness follows immediately, e.g. from the NP hardness of 3COL. It is also well-known [4, 43, 15] that the CSP is equivalent to various database problems, e.g., to the problem of evaluating *Boolean conjunctive queries* over a relational database.

To any CSP instance $I = (Var, U, \mathcal{C})$, we associate a hypergraph $H(I) = (V, H)$, where $V = Var$, and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, where $var(S)$ denotes the set of variables in the scope S of the constraint C. The graph $G(I)$ associated with a CSP is the primal graph $G(H(I))$ of the hypergraph $H(I)$. Note that if all constraints of a CSP instance I are binary, then its associated hypergraph $H(I)$ is identical to its graph $G(I)$.

The following result (for treewidth) was implicit in work of Dechter and Pearl [17] based on Freuder (see [29] for clarifications). It was explicitly stated by Kolaitis and Vardi (Theorem 5.4 in [46]) and, by Chekuri and Rajaraman[12], who considered a slightly different graph associated to a CSP-instance I. The more general version for bounded hypertree width was proven in [24].

▶ **Theorem 1.** *CSPs of bounded treewidth and CSPs of bounded hypertree width are tractable.*

Since bounded treewidth implies bounded hypertree width [24, 29], it is sufficient to consider the proof for bounded hypertree width. A detailed exposition is given in the proof of Theorem 21 of [46]. Essentially it is shown that a CSP instance I of hypertree width k can be transformed in time $O(n^k)$ into an instance I^* of size n^k, where n is the size of I, whose associated hypergraph is acyclic. I^* can then be solved by using Yannakakis' well-known method for answering acyclic queries [54]. The total time for solving I is shown to be $n^{k+1} \log n$.

It is natural to ask whether we could achieve fixed-parameter tractability (FPT) by finding a better algorithm which would allow us to get rid of the constant k in the exponent of n, and thus to replace the runtime bound by some expression $f(k) \times n^c$ for a function f and a constant c independent of k. Unfortunately, this appears to be very unlikely. In fact, the problem of evaluating Boolean conjunctive queries, which is identical to the CSP problem, is easily shown to be fixed-parameter intractable (more precisely, $W[1]$-hard) with respect to either parameter, the query size or the number of variables [50], see also [38, 18]. It is therefore a fortiori FP-intractable with respect to the treewidth parameter, given that any k-variable CSP has treewidth at most $k-1$. This makes the CSP a prime example for a problem in category 1.a of our classification.

The precise complexity of solving CSPs (or answering conjunctive queries) of bounded treewidth, or hypertree width, is as follows [30, 24]:

▶ **Theorem 2.** *Deciding satisfiability for CSP instances of bounded treewidth or bounded hypertree width is* LOGCFL*-complete.*

Given that LOGCFL is a class of highly parallelizable problems included in the well-known very low classes AC_1 and NC_2, this shows that even though FP-intractable in case of bounded treewidth, the problem is efficiently parallelizable. A concrete parallel algorithm can be obtained by combining the transformation of the original CSP instance of bounded treewidth, I, into an acyclic CSP instance I' (see [29]) (which is an AC_0-reduction) with the DB-SHUNT algorithm described in [30].

By results of Grohe, Schwentick, and Segoufin [40], for CSPs of *bounded arity* (i.e., whose constraint scopes and relations are of bounded arity, but otherwise unrestricted), it was shown that bounded treewidth is actually the best possible tractability criterion: a class of CSP instances of bounded arity is tractable if and only if it has bounded treewidth. However, when the constraints can have unbounded arity, bounded treewidth is a sub-optimal tractability criterion and is dramatically outperformed by bounded hypertree width. But, as already alluded to in the introduction, there are yet more powerful decomposition methods, such as fractional hypertree decompositions [39]. Even bounded fractional hypertree width does not seem to be an optimal structural tractability criterion. An optimal criterion was very recently established by Marx in [48], however, this criterion imposes conditions not only on the constraint scopes, but also on the constraint relations, and is thus of a different, less "structural" type. Finally, let us observe that it may be useful to consider hypertree decompositions or their generalizations, rather than just tree decompositions, even in the case of bounded arities. In fact, in Theorem 16 of [33] it was shown that each CSP instance I having n variables is of hypertree width $\leq \lfloor n/2 \rfloor + 1$, while its treewidth may actually be $n-1$.

3.2 Network Multicut Problems

We now illustrate category 1.b of our taxonomy by a number of variants of the well-known Multicut problem.

Multicut problems and their complexity. Multicut problems are highly relevant to the field of network design. The smallest size of a multicut reveals the robustness and stability of a network with multicommodity flows. Given a set $H \subseteq V^2$ of pairs of *terminal* vertices of a graph $G = (V, E)$, several forms of Multicut have been considered and have given rise to complexity studies [9, 14, 23, 42, 41, 47, 49].

A large part of the literature deals with the Edge Multicut variant in which the solution is a set of edges $E' \subseteq E$ that, if removed, separate every terminal pair. Variants in which sets of vertices are removed, such as Unrestricted Vertex Multicut, wherein any vertices can appear in the solution set V', and Restricted Vertex Multicut, wherein no terminal vertices can appear in the solution set, have also been studied [9]. Formal definitions of these versions of Multicut are given below.

In 2006, Guo et al. [41] present an algorithm which solves all three of the variants of the problem in polynomial time given *two* constant parameters, the cardinality of the set of terminal pairs, $|H|$, and the treewidth ω of G.

In [28], Gottlob and Tien Lee introduced a new single parameter for which all variants of Multicut are FPT. This unique parameter is the treewidth ω^* of the input structure $\mathcal{A} = (V, E, H)$, which is equal to the treewidth of the graph $(V, E \cup H)$. These more recent FPT-results are proved by using a powerful extended version of Courcelle's theorem due to Arnborg, Lagergren, and Seese [2]. By formulating Multicut in such an extended MSO, the considered Multicut problems are shown to be FPT with respect to ω^*. Note that if the input graph has bounded treewidth, and if H has bounded cardinality, then, obviously, the entire input structure $\mathcal{A} = (V, E, H)$ also has bounded treewidth ω^*. However, ω^* can be bounded even in cases where G has bounded treewidth but H has unbounded cardinality. The FPT results for bounded ω^* are thus a strict generalization of the results by Guo et al. [41]. These findings demonstrate that powerful logical tools such as the extended version of Courcelle's master theorem by Arnborg, Lagergren, and Seese [2] can be applied to advance the state of the art in network multicut theory and can help identify parameters of interest in complexity analysis.

Formal definitions of multicut problems. We now define the various different versions of the MULTICUT problem mentioned earlier. The Edge Multicut problem is formally defined as follows:

▶ **Definition 3.** Edge Multicut (EMC)

Input: An undirected graph $G = (V, E)$, and $H \subseteq V \times V$ a collection of pairs of vertices.
Task: Find a minimal cardinality set $E' \subseteq E$ whose removal disconnects each pair in H.

The vertex variants of the problem were identified by Calinescu et al. [9] and they are defined as:

▶ **Definition 4.** Unrestricted Vertex Multicut (UVMC)

Input: An undirected graph $G = (V, E)$, and $H \subseteq V \times V$ a collection of pairs of vertices.
Task: Find a minimal cardinality set $V' \subseteq V$ whose removal disconnects each pair in H.

If H is a set of pairs of elements, we denote by H_0 the set of all elements occurring in some pair of H, i.e., $H_0 = \{x | \exists y : (x, y) \in H \vee (y, x) \in H\}$.

▶ **Definition 5.** Restricted Vertex Multicut (RVMC)

Input: An undirected graph $G = (V, E)$, and $H \subseteq V \times V$, a collection of pairs of vertices.
Task: Find a minimal cardinality subset $V' \subseteq V$ where $V' \cap H_0 = \emptyset$ and whose removal disconnects each pair of vertices in H.

It has been shown that all three forms of the problem are NP-complete for general graphs [14, 9] and remain hard even for graphs with bounded treewidth [23, 9].

A strong master theorem by Arnborg, Lagergren, and Seese. Before showing that the problems become FPT for inputs of bounded treewidth ω^*, we present a strong and most useful generalization of Courcelle's Theorem by Arnborg, Lagergren, and Seese [2]. Let us refer to the version of MSO where second-order quantification is restricted to *sets of domain elements* (e.g., sets of vertices of the input graph) as MSO_1. Note that Courcelle [13] and other authors considered an extended version of MSO called MSO_2 which extends MSO_1 by the possibility of quantifying over subsets of any relation of the input structure, e.g., sets of edges of an input graph. Thus, for example, if a relational symbol R is part of the problem signature, then a subformula $(\exists X \subseteq R)\varphi(X)$, expressing that there exists a subset X of the relation R such that $\varphi(X)$ holds for some formula φ, could be part of an MSO_2 formula. Courcelle's Theorem is actually valid for MSO_2 (and thus, in particular, for MSO_1). Arnborg, Lagergren and Seese [2] considered an important extension of MSO_2, called *extended MSO* that can be used to formulate optimization and counting problems. They proved that solving problems expressible in this form over input structures is FPT with respect to the treewidth of these input structures.

While in the original setting in [2], extended MSO properties were defined in a much more general context, it is sufficient for our purposes to state a drastically simplified definition and, accordingly, a simplified master theorem (Theorem 7). By *optimization* we here understand the search for a minimum or maximum solution according to some cardinality criteria. The solution itself is an "optimal" assignment of sets to second-order variables that freely occur in some MSO formula, such that the formula is satisfied over a given input structure. More precisely:

▶ **Definition 6** (simplified version of a definition in [2])**.** An optimization problem is an *extended MSO cardinality optimization problem* if it can be expressed in the following form. The input of the problem is a a structure $\mathcal{A} = (V, E, H)$, where V is a set (the universe of $\mathcal{A}$), and E and H are binary relations over elements of V. Let $\varphi(X)$ be a fixed MSO_1 or MSO_2 formula over $\mathcal{A}$, where X is either a free set variable ranging over subsets of V, or a binary relation variable ranging over subsets of E. The task is to find an assignment[1] among all possible assignments z' to the variable X such that:

$$|z(X)| \;=\; opt\{|z'(X)| : (\mathcal{A}, z') \models \varphi(X)\}$$

where *opt* is either min or max. A suitable assignment z is called a *solution* to the extended MSO cardinality optimization problem.

Using this definition, Arnborg, Lagergren and Seese found the following important fixed-parameter tractability result [2]:

▶ **Theorem 7.** *Solving extended MSO cardinality problems is fixed-parameter tractable w.r.t. the treewidth of the input structure. In particular, for a fixed extended MSO cardinality optimization problem P, and a class C of input structures whose treewidth is bounded by some constant, the following can be done in linear time:*

- *Determine whether P has a solution for an input from C.*
- *Compute a solution for an input from C, if one exists.*

Applying the master theorem to multicut problems. We first define a useful formula that states that two vertices x and y are connected by a path that lies entirely in a set S of vertices.

[1] An assignment z to the variable X is an interpretation of X that maps X to a subset $z(X)$ of V if X is unary and a subset of E if X is binary.

▶ **Definition 8.** On structures $\mathcal{A} = (V, E, H)$ as above, let $connects(S, x, y)$ be defined as follows:

$$S(x) \wedge S(y) \wedge \forall P\Big(\big(P(x) \wedge \neg P(y)\big) \rightarrow \big(\exists v \exists w\,(S(v) \wedge S(w) \wedge P(v) \wedge \neg P(w) \wedge E(v, w))\big)\Big).$$

▶ **Lemma 9** ([28]). *Over a structure $\mathcal{A} = (V, E, H)$ as above, the formula $connects(S, x, y)$ states that there is a path in (V, E) that connects vertex x to vertex y, and this path lies entirely in S. In particular, this is also true for directed graphs.*

▶ **Theorem 10** ([28]). *The problems* UVMC, RVMC, *and* EMC *are fixed parameter linear with respect to the treewidth ω^* of the input structure (V, E, H).*

Proof. (Sketch.) We outline the proof for UVMC. The problem UVMC can in fact be expressed as an extended MSO cardinality optimization problem in the following way: Find an assignment $z \subseteq V$ to set variable X such that $|z(X)| = min\{|z'(X)| : \langle V, E, H, z'\rangle \models uvmc(X)\}$, where $uvmc(X)$ is an MSO_1 expression defined as:

$$uvmc(X) \quad \equiv \quad \forall x\, \forall y \Big(H(x, y) \rightarrow \forall S\big(\mathrm{connects}(S, x, y) \rightarrow \exists v(X(v) \wedge S(v))\big)\Big).$$

In words, the formula $uvmc(X)$ defines X to be such that for each pair $(x, y) \in H$ (i.e., for each pair (x, y) that must be disconnected), whenever there is a set S of vertices from V that contains a path from x to y, then X must intersect S, i.e. contain some vertex from S. In [28] it is formally proven that $uvmc(X)$ is true iff the set X is an unrestricted vertex multicut of (V, E, H). The theorem then follows immediately from Theorem 7. Slight variants of this proof yield the corresponding FPT results for RVMC and EMC. For EMC, quantification over subsets of the edge relation is used. For details, see [28]. ◀

Master theorems such as Courcelle's and the one of Arnborg, Lagergren and Seese are constructive and can be used for the implementation of model-checking tools such as MONA [45] that directly interpret an MSO-formulation of a problem, or directly compile an MSO-formula into a solution algorithm. However, in order to obtain more efficient algorithms and better upper bounds it is currently still advisable to attempt a more detailed *ad hoc* analysis of the problem at hand, once the master theorems show us they are FPT. The above FPT-results, for example, were used by Pichler, Rümmele and Woltran [51] as a starting point for the design of very effective algorithms and for the derivation of rather low complexity bounds for Multicut problems on inputs of low treewidth. In the future we may expect new tools that are able to automatically compile algorithms and bounds of a similar quality. As an intermediate step, we believe that it is rewarding to replace MSO by equally expressive but much simpler (and better optimizable) languages for expressing a problem. One such candidate is the monadic fragment of the well-known *Datalog* language [11, 10]. It was recently shown in [35] that over structures of bounded treewidth, when a tree decomposition is provided, monadic Datalog is exactly as expressive as MSO. As illustrated in [34, 36], many problems of practical relevance can be easily encoded in monadic Datalog, and special interpreters that execute monadic Datalog programs over structures of bounded treewidth turned out to be much more efficient than MONA fed with the MSO formulas that are logically equivalent to those Datalog programs. For a further discussion, see Section 5 of [34].

3.3 The Even Cycle Problem

The EVEN CYCLE PROBLEM (ECP) is the problem of determining whether a loop-free undirected graph has an even cycle. The tractability of ECP was open for a long time, until Carsten Thomassen proved it polynomial [53]. Actually, Thomassen proved the following more general result regarding the problems mCP of whether a loop-free undirected graph has a cycle of length 0 modulo m, where m is a fixed positive integer.

▶ **Theorem 11** ([53]). *For each integer $m > 0$, mCP is decidable in polynomial time.*

As the following proof outline shows, the proof is according to the pattern of Case 2.a in our taxonomy.

Proof. (Outline.) First it is shown that on graphs $G = (V, E)$ of bounded treewidth, mCP is tractable. With Courcelle's theorem this is very easy; it is sufficient to note that mCP can be expressed in MSO. (Given that Courcelle's Theorem was not known, Thomassen gave a slightly more involved *ad hoc* proof.)

The second part deals with input graphs of "large" treewidth. It is proven that for each fixed m, a number $t(m)$ can be determined, such that all graphs of treewidth $> t(m)$ must actually have a cycle whose length is a multiple of m. Thus, for the specific problem mCP, all instances of "high" treewidth are actually yes-instances. In particular, it is shown by using Robertson's and Seymour's result [52] that $t(m)$ can always be chosen large enough such that G must contain a subdivision of a grid H, which, in turn, must contain a cycle whose length is a multiple of m. ◀

In a similar fashion, while classifying the complexity of model checking for all prefix-classes of existential second-order logic (ESO) over graphs, it was shown in [27] that evaluating fixed closed formulas of type $\exists R_1, \ldots\ R_k \forall x \exists y \, \phi(V, E, R_1, \ldots, R_k, x, y)$ over a loop-free undirected input graph (V, E), where $R_1 \ldots, R_k$ are existentially quantified relation symbols of arbitrary arity, and where $\phi(V, E, R_1, \ldots, R_k, x, y)$ is a quantifier-free first order formula, is feasible in polynomial time. Note that this generalizes Thomassen's theorem, given that for each m, the problem mCP can be expressed by an ESO formula of this type.

4 The Partner Units Problem

In this section, we describe the Partner Units Problem (Pup). First, in Section 4.1, a general version of the problem is given, which is, however, intractable. In Section 4.2 we describe a special version of the Pup which is of particular industrial relevance. It is for this special case that we were able to establish tractability by exploiting the result that all yes-instances must necessarily have bounded pathwidth (and thus bounded treewidth). It is thus the special case which serves as a paradigmatic example of a problem in category 2.b of our taxonomy. In Section 4.2.4 we then report on a prototypical implementation for the special case that already could solve benchmark instances beyond the reach of the heuristic methods and "engineering approaches" previously used to solve this problem. This section gives only a short summary; the original work underlying this section, as well as detailed proofs can be found in [3].

4.1 Definition of the Partner Units Problem and Basic Facts

The Partner Units Problem (Pup) has recently been proposed as a new benchmark configuration problem [22]. It captures the essence of a specific type of configuration problem that frequently occurs in industry.

Informally it can be described as follows: Consider a set of sensors that are grouped into zones. A zone may contain many sensors, and a sensor may be attached to more than one zone. The Pup consists of connecting the sensors and zones to control units. These control units can be connected to the same fixed maximum number *UnitCap* of zones and sensors.[2] Moreover, if a sensor is attached to a zone, but the sensor and the zone are assigned to different control units,

[2] For ease of presentation we assume that *UnitCap* is the same for zones and sensors.

then the two control units in question have to be (directly) connected. However, a control unit cannot be connected to more than *InterUnitCap* other control units (the partner units).

The Pup occurs e.g. in the following application domain: Consider a museum where we want to keep track of the number of visitors that populate certain parts (zones) of the museum. To this end the doors leading from one zone to another are equipped with sensors. To keep track of the visitors the zones and sensors are attached to control units; the adjacency constraints on the control units ensure that communication between control units can be kept simple. It is worth pointing out that the Pup is not limited to this application domain: It occurs whenever sensors that are grouped into zones have to be attached to control units, and communication between units should be kept simple.

Figure 1 shows a Pup instance and a solution for the case $UnitCap = InterUnitCap = 2$ — six sensors (left) and six zones (right) which are completely inter-connected are partitioned into units (shown as squares) respecting the adjacency constraints. Note that for the given parameters this is a maximal solvable instance; it is not possible to connect a new zone or sensor to any of the existing ones.

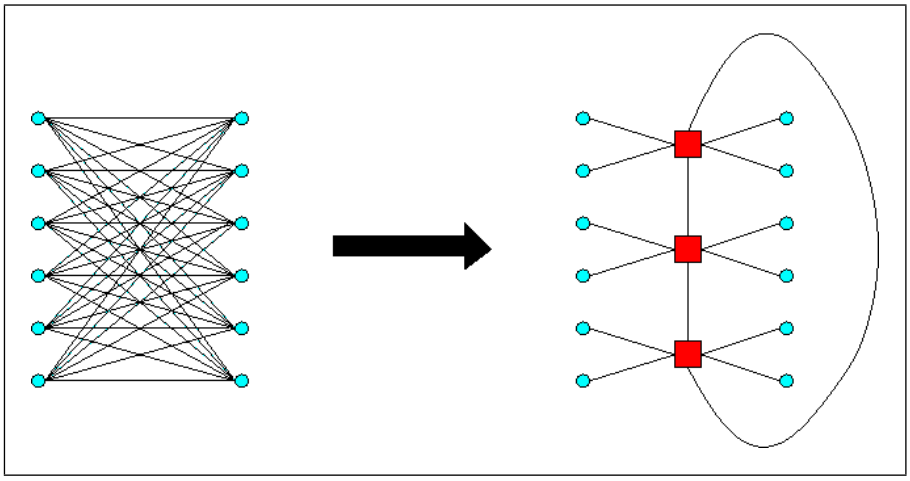

Figure 1 Solving a $K_{6,6}$ Partner Units Instance — Partitioning Sensors and Zones into Units

More formally, the Pup consists of partitioning the vertices of a bipartite graph $G = (V_1, V_2, E)$ into a set U of bags such that each bag

- contains at most *UnitCap* vertices from V_1 and at most *UnitCap* vertices from V_2; and
- has at most *InterUnitCap* adjacent bags where the bags U_1 and U_2 are adjacent whenever $v_i \in U_1$ and $v_j \in U_2$ and $(v_i, v_j) \in E$.

To every solution of the Pup we can associate a solution graph. For this we associate to every bag $u \in U$ a vertex $u' \in U'$. Then the solution graph G^* has the vertex set $V_1 \cup V_2 \cup U'$ and the set of edges $\{(v, u') \mid v \in u \wedge u \in U\} \cup \{(u'_i, u'_j) \mid u_i \text{ and } u_j \text{ are adjacent.}\}$. In the following we will refer to the subgraph of the solution graph induced by the u' as the *unit graph*.

The reasoning tasks for Pup instances that we consider in this paper are the following:

- Decide whether there is a solution (Pudp).
- Find a solution (Pusp).
- Find an optimal solution; i.e. one that uses the minimal number of control units (Puop).

The rationale behind the optimization version is that (a) units are expensive, and (b) connections are cheap. Especially the case where the maximum number *InterUnitCap* of connections between units is limited to two is of great interest for our partners in industry.

By a reduction from BinPacking it can be shown that Puop is NP-complete [3] when $\mathit{InterUnitCap} = 0$, and $\mathit{UnitCap}$ is part of the input. We also observe [3] that a Pup instance has no solution if it contains $K_{1,n}$ or $K_{n,1}$ as a subgraph, where $n = ((\mathit{InterUnitCap}+1)*\mathit{UnitCap})+1$.

4.2 A Special Case: $\mathit{InterUnitCap} = 2$

We now turn to the announced special case, which consists of Pups where $\mathit{InterUnitCap} = 2$, i.e. the number of neighbors of any given unit in a solution is bounded by 2. This special version of the Pup, which is of high industrial relevance, can be directly tackled. We will do this by giving an algorithm that decides this version of the Pup in NLogSpace by exploiting the notion of a path decomposition of a given graph.

For ease of presentation in the sequel we make the simplifying assumption that the underlying bipartite graph is connected. This does not affect solutions of the Pudp and the Pusp, where the connected components can be tackled independently. For optimal solutions, however, the connected components of an underlying graph will have to be considered simultaneously; cf. the discussion in section 4.2.3.

4.2.1 Basic Properties of the Pup (Special Case)

We proceed by identifying basic properties of the Pup in the special case. The key observation is that the units and their interconnections form a special kind of unit graph in any solution: either a simple path, or a simple cycle. This holds because each unit is connected to at most two partner units. Moreover, cycles are more general unit graphs than paths: Every solution can be extended to a cyclic solution; hence in the sequel we only consider cyclic solutions.

Exploiting this observation we can transform the Pup into the problem of finding a suitable path decomposition $\mathcal{P}$ of the zones-and-sensors-graph G:

▶ **Theorem 12** (Pup is Path-Decomposable). *Assume a* Pup *instance given by a graph* $G = (V_1, V_2, E)$ *is solvable with a solution graph* G^* *with* $|U| = n$. *Let* f *be the unit function that associates vertices from* G *to* U. *Then there is a path decomposition* $\mathcal{P} = (P, \chi)$ *of* G *of pathwidth* $\leq (3 * 2 * \mathit{UnitCap}) - 1$, *with the following special properties:*

(a) The length of P *is* $n - 1$; $P = w_1, \ldots w_{n-1}$.
(b) There are sets $S_1 \subseteq V_1$, $S_2 \subseteq V_2$ *with* $|S_i| \leq \mathit{UnitCap}$ *such that* $S_1 \cup S_2$ *are in every bag of* $\mathcal{P}$.
(c) Apart from $S_1 \cup S_2$ *each bag contains at most* $2 * \mathit{UnitCap}$ *elements from* V_1 *(or* V_2, *respectively).*
(d) For any vertex $v \in V_1 \cup V_2$ *all neighbors of* v *appear in three consecutive bags of* $\mathcal{P}$ *(assuming the first and last bag to be connected).*
(e) For each bag $\chi(w_i)$ *of* $\mathcal{P}$ *it holds* $\chi(w_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$ *for* $1 \leq i \leq n - 1$.
(f) $S_1 = f^{-1}(U_1) \cap V_1$ *and* $S_2 = f^{-1}(U_1) \cap V_2$.

Proof. If G is solvable then there is a solution G^* whose unit graph is a cycle $U_1, \ldots, U_n, U_1$. Consider $\mathcal{P} = (P = w_1, \ldots, w_{n-1}, \chi)$ where $\chi(w_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$. This $\mathcal{P}$ is indeed a path decomposition:

- Every edge (v_1, v_2) is in some bag. Assume v_1 and v_2 are assigned to two different connected units U_i and U_{i+1}. Then $\{v_1, v_2\} \subseteq \chi(w_i)$.
- The connectedness condition is satisfied: For the vertices connected to unit U_1 the induced subgraph is P. All other vertices occur in at most two consecutive bags.
- Every bag in $\mathcal{P}$ contains $\leq (3 * 2 * \mathit{UnitCap})$ elements; hence $\mathrm{pw}(\mathcal{P}) \leq (3 * 2 * \mathit{UnitCap}) - 1$.

An optimal path decomposition of the complete bipartite graph $K_{n,n}$ with $n = 3 * \mathit{UnitCap}$ has width $(3 * 2 * \mathit{UnitCap}) - 1$; cf. figure 1. Hence the bound is tight. The conditions $(a - f)$ are easily seen to hold for the path decomposition $\mathcal{P}$ constructed above. ◀

Intuitively, the vertices in the sets S_1 and S_2 from condition (b) above are those that close the cycle (i.e. that are connected to unit U_1). These have to be in every bag as some of their neighbors might only appear on the last unit U_n.

4.2.2 An Algorithm for the Special Case

By Theorem 12 we know that if a PUP instance is solvable then there is a path decomposition with specific properties. But we still need an algorithm for finding such suitable path decompositions. Many algorithms for finding path decompositions of bounded width have been proposed in the literature. But for the PUP we want to find path decompositions $\mathcal{P}$ with specific properties:

- The paths should be short (the number of bags reflects the number of units); and hence,
- The bags should be rather full (in "good" solutions the units will be filled up).
- The construction of the bags must be interleaved with checking the additional constraints.

Below we introduce a novel algorithm that fits the bill; it is inspired by the algorithm for finding hypertree decompositions from [24]. This non-deterministic algorithm does the following: The bags on the path decomposition are guessed. The initial bag partitions the graph into a set of remaining components that are recursively processed simultaneously. A single bag suffices to remember which part of the graph has already been processed; the bag *separates* the processed part of the graph from the remaining components. Consequently, the current bag and the remaining components can be stored in in logarithmic space, and the algorithm runs in NLOGSPACE. In addition to the bags the unit function is guessed, too. According to condition $(2d)$ of Theorem 12 all neighbors of any vertex in G occur in three consecutive bags in $\mathcal{P}$. Hence, for checking locally that the unit function is correct it suffices to remember three bags at each step.

A closer look reveals that it actually is enough to remember only U_1 and two "first" and "second" units U_{i-1} and U_i. At each step the current bag's content is then given by the union of U_1 with $U_{i-1} \cup U_i$. For the next step a third unit U_{i+1} is guessed. All neighbors of vertices assigned to the current second unit U_i are guaranteed to appear in $U_{i-1} \cup U_i \cup U_{i+1}$. For the current first unit U_{i-1} this will already have been established (if $i > 2$); hence, in the next step the new current first and second unit U_i and U_{i+1} together with U_1 are again a proper separator. The neighbors of U_1, however, are only guaranteed to appear somewhere on the first, second, or last unit. Upon termination the current "second" unit is the last unit in the cycle. But in addition to the first unit U_1 the second unit U_2 has to be stored throughout a run of the algorithm, too:

DECIDEPUP(G)
1 Guess disjoint non-empty $U_1, U_2 \subseteq V(G)$ with $|U_i \cap V_1| \leq \mathit{UnitCap} \geq |U_i \cap V_2|$
2 $C_R \leftarrow G \setminus (U_1 \cup U_2)$
3 **if** DECIDEPUP $(C_R, \langle U_1, U_2\rangle, \langle U_1, U_2\rangle)$
4 **then ACCEPT**
5 **else REJECT**

DecidePup$(C_R, \langle U_1, U_2\rangle, \langle U_{i-1}, U_i\rangle)$

```
1  if C_R = ∅
2     then
3          if ∀v ∈ U_1 nb(v) ⊆ U_1 ∪ U_2 ∪ U_i and
           ∀v ∈ U_i nb(v) ⊆ U_{i-1} ∪ U_i ∪ U_1
4            then ACCEPT
5            else REJECT
6     else
7          Guess non-empty U_{i+1} ⊆ V(⋃ C_R) with |U_{i+1} ∩ V_1| ≤ UnitCap ≥ |U_{i+1} ∩ V_2|
8          For v ∈ U_i check nb(v) ⊆ (U_{i-1} ∪ U_i ∪ U_{i+1})
9          C'_R ← (C_R \ U_{i+1})
10         DecidePup (C'_R, ⟨U_1, U_2⟩, ⟨U_i, U_{i+1}⟩)
```

Using this algorithm in [3] we show the following:

▶ **Theorem 13** (Tractability of Pudp). *The decision problem for the* Pup *is solvable by the algorithm* DecidePup *in* NLogSpace *for InterUnitCap* $= 2$ *and any given fixed value of UnitCap.*

Answer Extraction

For actually obtaining a solution to a Pup instance we face the following problem: In general it is not possible to remember the contents of all the bags in logarithmic space. Theoretically this problem can be solved as follows: On a first accepting run of DecidePup we clearly can remember the first bag's contents in logarithmic space. We can then run DecidePup again with a fixed first bag, and so forth. Hence the following holds:

▶ **Theorem 14** (Tractability of Pusp). *The problem of finding a solution to the* Pup *is solvable in* NLogSpace *for InterUnitCap* $= 2$ *and any given fixed value of UnitCap.*

Note that the problem of answer extraction disappears when actually implementing the nondeterministic algorithm on a deterministic computer; cf. section 4.2.4.

Towards an Efficient Algorithm

We next make a number of observations that can be exploited to turn DecidePup into a practically efficient algorithm.

Guiding the Guessing Not all zones and sensors assigned to units have to be chosen randomly. At most *UnitCap* neighbors of sensors and zones on the first unit can be assigned to the last unit. Hence the following holds:[3]

$$|\mathrm{nb}_s(U_1) \setminus (U_1 \cup U_2)| \leq \mathit{UnitCap} \geq |\mathrm{nb}_z(U_1) \setminus (U_1 \cup U_2)|.$$

Moreover, the neighbors of U_1 not assigned to U_1 or U_2 may only be guessed in the last step, where the number of unprocessed sensors (or zones) is at most *UnitCap*.

Starting from $i \geq 2$ we have the stronger:

$$(\mathrm{nb}_s(U_i) \setminus (U_i \cup U_{i-1})) \subseteq U_{i+1} \supseteq (\mathrm{nb}_z(U_i) \setminus (U_i \cup U_{i-1})).$$

Finding Optimal Solutions First Next recall that "good" solutions correspond to short path decompositions with filled-up bags. Moreover, the number of units used in the solution of a

[3] We denote by $\mathrm{nb}_s(U)$ the set of sensors adjacent to vertices in U.

PUP instance $G = (V_1, V_2, E)$ can always be bounded by $lb = \lceil \frac{\max(|V_1|,|V_2|)}{UnitCap} \rceil$ from below and $ub = \max(|V_1|, |V_2|)$ from above [3]. Hence we can apply iterative deepening search: First, try to find a solution with lb units; if that fails increase lb by one. This has the effect that the first solution found will be optimal. This yields the following:

▶ **Corollary 15** (Tractability of PUOP). *On connected input graphs the optimization problem for the* PUP *is solvable in* NLOGSPACE.

In this context let us point out that branch-and-bound-search (on the number of units used) does not work: E.g. a $K_{6,6}$ graph does not admit solutions with more than three units.

Symmetry Breaking We already observed that cycles are more general unit graphs than paths. But with cycles for unit graphs there is rotational symmetry: For a solution with unit graph $U_1, \ldots, U_n, U_1$ there is a solution $U_2, \ldots, U_n, U_1, U_2$, etc.. We can break this symmetry without additional computational cost by requiring that

- the first sensor is assigned to unit U_1; and
- the second sensor appears somewhere on the first half of the cycle.

4.2.3 PUOP and Multiple Connected Components

Next let us discuss the problem of finding optimal solutions when the input graph consists of more than one connected component. Here, part of the problem is that any two connected components may either have to be assigned to the same, or to two distinct unit graph(s). A priori it is unclear which of the two choices leads to better results. E.g. if we assume that $UnitCap = 2$ then two $K_{3,3}$ should be placed on one cyclic unit graph, while two $K_{6,6}$ must stand alone. In [3] we are able to show the following:

▶ **Theorem 16** (Tractability of PUOP on Multiple Connected Components). *For $InterUnitCap = 2$ and any given value of UnitCap the optimization problem for the* PUP *on multiple connected components is solvable in* NLOGSPACE *if there are only logarithmically many connected components in the input graph.*

4.2.4 Implementation and Evaluation

We prototypically implemented the DECIDEPUP algorithm in Java, replacing the non-determinism by a backtracking search mechanism. A detailed description of the procedure is beyond the scope of this paper. However, in [25] a deterministic backtracking version of the non-deterministic hypertree decomposition algorithm from [24] is described, and the issues we face when making DECIDEPUP deterministic are very similar. Suffice it to say the following: To avoid repeated sub-computations we store those pairs of bags and remaining components (represented by unassigned neighbors) that could not be decomposed. We don't store successful pairs — the first such pair occurs when finding a solution. As there are only polynomially many such pairs the overall runtime of the algorithm is polynomial [3]. Finally observe that for the backtracking search we have to store the choices made, and hence answer extraction is easy.

We have evaluated our algorithm on a set of benchmark instances that we received from our partners in industry. Using our prototypical implementation we could solve many instances that were beyond the reach of the previously used heuristic methods and engineering approaches [3].

5 Conclusion

In this work we have reviewed structural problem decomposition methods, such as path-, tree-, and hypertree decompositions. We have introduced a taxonomy of usages of treewidth for proving tractability results, and illustrated each category by an example. In particular, we have shown

that treewidth (or pathwidth) can be applied in two distinct main ways: Either to show that a problem is efficiently solvable when a width parameter is fixed, or to prove that the unrestricted (or some width-parameter free) version of a problem is tractable by using a width-notion as a mathematical tool for directly solving the problem at hand.

As a show case for the latter usage we have reported on some recent results concerning the Partner Units Problem, a type of configuration problem that was proposed to us by an industrial partner. We have shown that, while the PUP is intractable in general, the notion of a path decomposition can be used to obtain a polynomial algorithm for a highly relevant special case. Our prototypical implementation of the respective DECIDEPUP algorithm could solve many previously unsolvable problem instances.

There is still significant work to be done on the PUP:

- We need to analyze the cases with $\mathit{InterUnitCap} = k$ and $\mathit{UnitCap} = m$ for fixed constants $k > 2, m$.
- We would like to find better algorithms for the NP-hard general case (when *InterUnitCap* and *UnitCap* are unbounded).
- We have not yet exploited heuristics for the search.

Acknowledgment. Work funded by EPSRC Grant EP/G055114/1 "Constraint Satisfaction for Configuration: Logical Fundamentals, Algorithms and Complexity. G. Gottlob's work, in particular, with respect to specific topics arising in the context of the co-operation with Siemens Austria and the University of Klagenfurt, was also partially funded by the FFG FIT-IT project RECONCILE. G. Gottlob would also like to acknowledge the Royal Society Wolfson Research Merit Award.

References

1 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

3 M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen. Tackling the Partner Units Problem. Technical Report RR-10-28, Computing Laboratory, University of Oxford, 2010. Available from the authors.

4 Wolfgang Bibel. Constraint satisfaction from a deductive viewpoint. *Artif. Intell.*, 35(3):401–413, 1988.

5 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 226–234, New York, NY, USA, 1993. ACM.

6 Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

7 Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 672–683, 2006.

8 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations i. upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.

9 Gruia Calinescu, Cristina G. Fernandes, and Bruce A. Reed. Multicuts in unweighted graphs with bounded degree and bounded tree-width. *J. Algorithms*, 48(2):333–359, 2003.

10 Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng.*, 1(1):146–166, 1989.

11 Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.

12 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

13 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

14 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

15 Rina Dechter. From local to global consistency. *Artif. Intell.*, 55(1):87–108, 1992.

16 Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

17 Rina Dechter and Judea Pearl. Tree-clustering schemes for constraint-processing. In *AAAI*, pages 150–154, 1988.

18 R.G. Downey, M.R. Fellows, and U. Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. *Complexity, and Logic, volume 39 of Proceedings of DMTCSJ*, pages 149–213, 1996.

19 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

20 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152, 2010.

21 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

22 A. Falkner, A. Haselböck, and G. Schenner. Modeling Technical Product Configuration Problems. In *Proceedings of the Workshop on Configuration at ECAI 2010*, pages 40 – 45, Lisbon, Portugal, 2010.

23 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 181:3–20, 1997.

24 G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decomposition and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

25 G. Gottlob and M. Samer. A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13, 2008.

26 Georg Gottlob and Gianluigi Greco. On the complexity of combinatorial auctions: structured item graphs and hypertree decomposition. In *ACM Conference on Electronic Commerce*, pages 152–161, 2007.

27 Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *J. ACM*, 51(2):312–362, 2004.

28 Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. *Inf. Process. Lett.*, 103(4):136–141, 2007.

29 Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

30 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

31 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: A survey. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, pages 37–57, 2001.

32 Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6), 2009.

33 Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.

34 Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.

35 Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic datalog over finite structures of bounded treewidth. *ACM Trans. Comput. Log.*, 12(1):3, 2010.

36 Georg Gottlob, Reinhard Pichler, and Fang Wei. Tractable database design and datalog abduction through bounded treewidth. *Inf. Syst.*, 35(3):278–298, 2010.

37 Martin Grohe. Descriptive and parameterized complexity. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic CSL'99*, volume 1683, pages 14–31. Springer, 1999.

38 Martin Grohe. The parameterized complexity of database queries. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 82–92, New York, NY, USA, 2001. ACM.

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

40 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, pages 657–666, 2001.

41 Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for multicut. In Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM*, volume 3831 of *Lecture Notes in Computer Science*, pages 303–312. Springer, 2006.

42 Jiong Guo and Rolf Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Netw.*, 46(3):124–135, 2005.

43 Marc Gyssens, Peter Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell.*, 66(1):57–89, 1994.

44 Petr Hlinený, Sang il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

45 Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. Mona implementation secrets. *Int. J. Found. Comput. Sci.*, 13(4):571–586, 2002.

46 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

47 Dániel Marx. Parameterized graph separation algorithms. *Theoretical Computer Science*, 351(3):394–406, 2006.

48 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 735–744, 2010.

49 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *CoRR*, abs/1010.3633, 2010.

50 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

51 Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Multicut algorithms via tree decompositions. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 167–179, 2010.

52 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

53 Carsten Thomassen. On the presence of disjoint subgraphs of a certain type. *Journal of Graph Theory*, 12(1):101–111, 1988.

54 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.