From Hypertree Width to Submodular Width and Data-dependent Structural Decompositions

Francesco Scarcello

DIMES, University of Calabria, 87036, Rende, Italy
scarcello@dimes.unical.it

Abstract. Identifying the frontier of tractability for conjunctive queries is a long-standing question in database theory. Recent advances in structural decomposition methods provide an important answer regarding the fixed-parameter tractability of conjunctive queries, and efficient algorithms based on decomposition methods have been implemented in database management systems. The paper shows that these techniques are useful not only for long and complex queries, but even for short and simple ones. Moreover, it sheds some light on some subtle issues that are not completely understood yet, and discuss open problems, such as the gap remaining between the theoretical fixed-parameter polynomial-time upper-bound and the running times required by actual algorithms used in practice.

1 Introduction

Conjunctive queries are defined through conjunctions of atoms (without negation), and are known to be equivalent to Select-Project-Join queries. The problem of evaluating such queries is NP-hard in general, but it is feasible in polynomial time on the class of acyclic queries (we omit "conjunctive," hereafter), which was the subject of many seminal research works since the early ages of database theory This class contains all queries Q whose associated query hypergraph $\mathcal{H}_Q$ is acyclic, where $\mathcal{H}_Q$ is a hypergraph having the variables of Q as its nodes, and the (sets of variables occurring in the) atoms of Q as its hyperedges.

In fact, queries arising from real applications are hardly precisely acyclic. Yet, they are often not very intricate and, in fact, tend to exhibit some limited degree of cyclicity, which suffices to retain most of the nice properties of acyclic ones. Therefore, several efforts have been spent to investigate invariants that are best suited to identify nearly-acyclic hypergraphs, leading to the definition of a number of so-called *(purely) structural decomposition-methods* [12,9], such as the *tree decompositions* [20] and, after some years, the *(generalized) hypertree* [7,8], *fractional hypertree* [15], *component hypertree* [11], and *greedy hypertree* [14,13] decompositions. These methods aim at transforming a given cyclic hypergraph into an acyclic one, by organising its edges (or its nodes) into a polynomial number of clusters, and by suitably arranging these clusters as a tree, called decomposition tree. The original problem instance can then be evaluated over such a tree of subproblems (in our case, subqueries), with a cost that is exponential with respect to a measure of the complexity of the subproblems, also called *width* of

SEBD 2018, June 24-27, 2018, Castellaneta Marina, Italy.

the decomposition, and polynomial if this width is bounded by some constant. For instance, according to the treewidth notion, the width of a decomposition is given by the maximum cardinality (minus one) of the set of variables occurring in the subqueries associated with the vertices of the decomposition tree. The (generalized) hypertree width considers instead the number of atoms occurring in each subproblem, in other terms, the number of hyperedges needed to cover the relevant variables in each vertex of the decomposition tree. The fractional hypertree width [15] is similar, but it is based on a fractional cover of such variables, instead of an integral one, as in the case of (plain) hypertree decompositions.

Besides the theoretical guarantees on the cost of evaluating class of queries having bounded (fractional) hypertree width, experimental evidence of the benefits gained from these structural decomposition methods in query optimization (and in the equivalent Constraint Satisfaction Problem) has been provided in the literature by a number of different authors and in different application domains, see, e.g., [5,6,1].

A yet more general width-concept is the *submodular width* [19], where the cost of a decomposition is determined by the worst possible submodular function of sets of variables of the given hypergraph. Recall that a set function f is submodular if, for every element x and any pair of sets S and T with $S \subseteq T$ and $x \notin T$, $f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T)$ holds. That is, the marginal contribution of any element decreases when sets become larger. This powerful notion is strictly more general than all previous techniques, as there are classes of hypergraphs having bounded submodular width, but unbounded fractional hypertree width. However, having bounded submodular width (smw) does not guarantee a polynomial-time combined complexity as for hypertree decompositions, but just fixed-parameter tractability where the query hypergraph is used as a parameter. This means that we have a polynomial-time data complexity of the form $O(f_1(\mathcal{H}_Q) \cdot |\mathrm{DB}|^{f_2(\mathrm{smw})})$, where f_1 is an exponential function of the query size. Marx also proved that having bounded smw is in fact a necessary condition for the fixed-parameter tractability of conjunctive queries (unless the Exponential Time Hypothesis fails) [19]. Unfortunately, it is not clear how to recognise queries of bounded submodular width. It is worthwhile noting that, unlike the previous methods, the decompositions used to answer the query depend not only on the hypergraph $\mathcal{H}_Q$, but on the actual database DB, too. Nevertheless, the submodular width of $\mathcal{H}_Q$ depends only on $\mathcal{H}_Q$, which makes it an actual structural measure.

The primary aim of the paper is to shed light on the recent achievement in structural decomposition methods, in particular

- to make clear why the powerful notion of fractional hypertree width is not able to catch the precise complexity of a given query;
- to show, by using the case study of queries whose hypergraphs are simple cycles, that using data-dependent decompositions, as in the case of submodular width, provides better upper bounds on the cost of queries, and effective algorithms;
- to make evident that the benefits of such sophisticated notions occur not only in long and complex queries, but even in short and simple ones, occurring in most real-world applications;
- to give a hint of what is the submodular width of a query, and a hint of the problems that are still open in this area of research.

2 Hypertree decompositions and the AGM bound

Let $\mathcal{H}$ a hypergraph, and denote the set of its nodes by $nodes(\mathcal{H})$ and the set of its hyperedges by $edges(\mathcal{H})$. Formally, a *tree decomposition* [20] of $\mathcal{H}$ is a pair $\langle T, \chi \rangle$, where $T = (V, F)$ is a tree, and χ is a labelling function assigning to each vertex $p \in V$ a set of vertices $\chi(p) \subseteq nodes(\mathcal{H})$, such that the following three conditions are satisfied: (1) for each node b of $\mathcal{H}$, there exists $p \in V$ such that $b \in \chi(p)$; (2) for each hyperedge $h \in edges(\mathcal{H})$, there exists $p \in V$ such that $h \subseteq \chi(p)$; and (3) for each node b in $nodes(\mathcal{H})$, the set $\{p \in V \mid b \in \chi(p)\}$ induces a connected subtree of T. The *width* of $\langle T, \chi \rangle$ is the number $\max_{p \in V}(|\chi(p)| - 1)$. The *treewidth* of $\mathcal{H}$, denoted by $tw(\mathcal{H})$, is the minimum width over all its tree decompositions.

Hypertree Decompositions A *generalized hypertree decomposition* of a hypergraph $\mathcal{H}$ is a triple $HD = \langle T, \chi, \lambda \rangle$, called a *hypertree* for $\mathcal{H}$, where $\langle T, \chi \rangle$ is a tree decomposition of $\mathcal{H}$, and λ is a function labelling the vertices of T by sets of hyperedges of $\mathcal{H}$ such that, for each vertex p of T, $\chi(p) \subseteq \bigcup_{h \in \lambda(v)} h$. That is, all nodes in the χ labelling are covered by hyperedges in the λ labelling.

Consider a generalized hypertree decomposition HD=$\langle T, \chi, \lambda \rangle$. The cyclicity measure used in standard generalized hypertree decompositions, called generalized hypertree width and denoted by $ghw(\mathcal{H})$, is based on the cardinality of the set $\lambda(p)$ in charge of covering $\chi(p)$, for each vertex p of the decomposition tree. Contrasted with the treewidth, which is based on the cardinality of $\chi(p)$, it is clear that $ghw(\mathcal{H}) \leq tw(\mathcal{H})$ always holds. It has been observed that more general notions can be obtained by allowing the use of more general functions $f_{HD}(\cdot)$, to measure the weight $f_{HD}(p)$ of any vertex of T, and hence to define the width of a hypertree decomposition.

A *fractional hypertree decomposition* [15] of a hypergraph $\mathcal{H}$ is a pair $FHD = \langle HD, \gamma \rangle$, where $HD = \langle T, \chi, \lambda \rangle$ is a generalized hypertree decomposition of $\mathcal{H}$, and γ is a mapping associating each vertex p of T with a function $\gamma_p : \lambda(p) \mapsto \mathbb{R}$ mapping hyperedges to non-negative real numbers. For each vertex p in T, γ_p encodes a *fractional cover* of the nodes in $\chi(p)$: for each $v \in \chi(p)$, $\sum_{h \in \lambda(p), v \in h} \gamma_p(h) \geq 1$. Define $\rho(p) = \sum_{h \in \lambda(p)} \gamma_p(h)$. The *width* of FHD is the maximum of $\rho(p)$ over all vertices p in T. The *fractional hypertree width* of $\mathcal{H}$, denoted by $fhw(\mathcal{H})$, is the minimum width over all its fractional hypertree decompositions.[2]

Generalized hypertree width is obtained if in the definition of fractional hypertree decomposition we restrict the codomain of each γ_p to be integral. It is thus not surprising that there are hypergraphs where the fractional hypertree width is smaller than the (generalized) hypertree width.

The additional power of fractional covers in the decomposition vertices makes the problem intractable, in general. Indeed, for a given hypergraph $\mathcal{H}$ and for any fixed $w \geq 2$, computing a hypertree decomposition of width at most w (if any) is feasible in polynomial-time, while checking whether $\mathcal{H}$ has fractional hypertree width at most w is NP-hard [4]. However, there is a polynomial-time algorithm for deciding whether the fractional hypertree width is $f(w)$ for a function f in $O(w^3)$ [18].

[2] The equivalent (original) definition in [15] does not use the λ-labeling, so that γ_p weighs all hyperedges. Our definition is used, e.g., in [1].

Query evaluation Queries whose (fractional) hypertree width is bounded by some constant k can be answered in polynomial time. Given a query Q and a hypertree decomposition HD for the query hypergraph $\mathcal{H}_Q$, the query can be transformed into an acyclic query, based on the decomposition HD. Each vertex of the decomposition tree can be viewed as a subproblem to be solved: in our database context, such a vertex p is associated with a subquery Q_p whose atoms are the atoms in $\lambda(p)$, and whose set of output variables is $\chi(p)$. By computing the answers of these subqueries, we obtain an acyclic query equivalent to Q that can be evaluated easily, by standard techniques. Equivalently, we can see this transformation as a query plan for Q (where some atoms can appear multiple times with different projections, if necessary). The crucial observation here is that the maximum number of answers of the subquery Q_p associated with each vertex p is at most $|r_p|^{\rho(p)}$, where r_p is the largest database relation associated with the hyperedges in $\lambda(p)$ [15,3]. Moreover, such answers can be efficiently computed within this upper bound [1].

Tight Bounds The so-called AGM bound [3] states that, in fact, the upper bound on the number of solutions at each subproblem Q_p provided by the best fractional cover ρ^* is tight: there exists some database DB such that the number of answers of Q_p over this database meets the fractional cover upper bound. This result has been refined in [10], where a bound based on the so-called *coloring number* $C(Q_p)$ of such a query is defined, in order to take into account the output variables, as well as certain kinds of functional dependencies.

It follows that, whenever the fractional hypertree width of a query Q is w, and we have any fractional hypertree decomposition of width w, there always exists a database DB such that evaluating the subproblem Q_p at some vertex p of the decomposition tree requires $\Omega(N^w)$ time, where N is the size of the largest relation occurring in Q_p. Yet, from the results based on the submodular width, it turns out that we could do much better. How is that possible?

3 When worst cases are unachievable

In this section, we show that fractional hypertree decompositions do not provide a tight bound on the cost of evaluating conjunctive queries. This is not very clear, at a first sight, because of the results mentioned in the previous section. However, we see that even in very simple queries the worst cases predicted by fractional hypertree width are impossible to achieve.

Let us start with a hint of what happens. Consider a conjunctive query q with an associated hypergraph $\mathcal{H}_q$ having $fhw(\mathcal{H}_q) = k$, and let HD be a width-k fractional hypertree decomposition for q. Let v be a vertex of HD whose associated subproblem has a fractional cover of weight k. Then, there exists a database DB such that solving this subproblem takes at least $O(n^k)$ time. However, it is possible that there exists another decomposition HD' with no subproblem having such an evaluation cost over this specific database DB. In its turn, HD' will have a critical database DB$'$ for which some of its subproblems require $O(n^k)$ time. But DB$'$ is not necessarily a bad case for the previous decomposition HD. Therefore, it is possible that, for the given query q, for every database DB there is a fractional hypertree decomposition where any subproblem occurring in its vertices can be evaluated in less than $O(n^k)$ time.

As a case study of this phenomenon, we investigate the class of conjunctive queries whose associated hypergraphs are (simple) cycles. These queries have (fractional) hypertree width 2, but they can be actually evaluated in subquadratic time on any given database.

A simple example: the cycles First, consider the following simple cyclic query q_4, whose (hyper)graph G_4 is shown in Figure 1:

$$ans(\bar{X}) \leftarrow r_1(X_1, X_2) \wedge r_2(X_2, X_3) \wedge r_3(X_3, X_4) \wedge r_4(X_4, X_1).$$

Consider the decomposition of this graph in the center of Figure 1, which involves subproblems such as $r_1(X_1, X_2) \wedge r_2(X_2, X_3)$ and $r_3(X_3, X_4) \wedge r_4(X_4, X_1)$ having a fractional cover of weight 2. It follows that there exists a database DB_4 such that, e.g., the subquery $ans(X_1, X_2, X_3) \leftarrow r_1(X_1, X_2) \wedge r_2(X_2, X_3)$ has $O(N^2)$ answers, where for the sake of simplicity we assume that the relations are the same and that they have N tuples each.

Think now of how this worst-case bound can be achieved. First assume that r_i is a cartesian product of the domain of its variables: we have $N = |d(X_1)| \cdot |d(X_2)|$ so that $|d(X_1)| = N^{1/2}$. Because $ans(X_1, X_2, X_3)$ cannot be larger than the cartesian products of the domain of its variables, it follows that, in this case, $|ans(X_1, X_2, X_3)| \leq N^{3/2}$. In order to reach the worst-case bound of N^2, in DB_4 the attributes should have quite different domain sizes. In particular, we must have many values on the extremal variables of the chain X_1, X_2, X_3, that is, almost N values for X_1 and X_3, and a few values for X_2. For instance, X_2 may have just one skew value that is connected to all values for X_1 and X_3, so that $ans(X_1, X_2, X_3) = d(X_1) \times 1 \times d(X_3)$ and $|ans(X_1, X_2, X_3)| = N^2$. For completeness, note that if X_2 has many values, say almost N, than it should behave like a key, and the size of $ans(X_1, X_2, X_3)$ will be almost linear in N. Summing-up, to achieve the $O(N^2)$ worst-case bound on $ans(X_1, X_2, X_3)$, we should have $|d(X_2)| = \Delta$, with Δ much smaller than N. However, in this case we can use the decomposition shown on the right in Figure 1: note that the subproblems in the vertices of this decomposition have at most $N\Delta$ answers, achieved in the vertices in the middle where we have a cartesian product of the form $r_i \times d(X_2)$. The cost of evaluating q_4 over DB_4 using this decomposition is thus $O(N\Delta)$, which is smaller than the $O(N^2)$ cost predicted by fractional hypertree width.

We next show that the above reasoning can be generalized to any database, showing that *the quadratic fractional hypertree-width worst-case is not tight*, in particular such a query can be evaluated in $O(N^{3/2})$, that is, within the same bound as the triangle query.

A general bound for evaluating cycles Following the intuition described in the previous section, we next show that actually we can guarantee a subquadratic evaluation time for every conjunctive query having a cycle as its hypergraph and on any given database. A further ingredient to obtain a precise upper bound is to consider horizontal fragments of the relations, and possibly use different decompositions for evaluating the given query on different fragments. Therefore, not only we look for the best decomposition with regard to specific database, but also the best ones for different fragments of the database.

This bound is a slight improvement of the upper bound described in [16] and it is quite similar to the upper bound given in [2] for the special case of queries asking for length-k simple cycles in graphs, whose techniques are indeed used both in [16] and

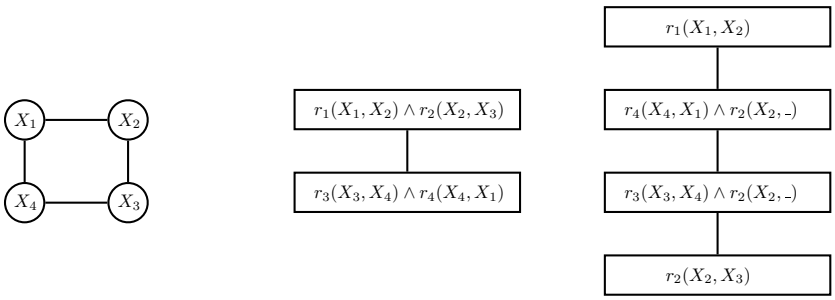

Fig. 1. The graph G_4 and two of its hypertree decompositions

in the present paper. Contrasted with this latter upper bound, our result is more general because it works with queries involving any combination of arbitrary binary relations, while the results on graphs assume only one relation (the edge relationship of the input graph).

Theorem 1. *Let $\mathcal{C}$ be any class of conjunctive queries whose associated hypergraphs are (simple) cycles. Then, the answers of any query in $\mathcal{C}$ having k atoms on a database of size N can be computed in $O((2/k)^{\frac{1}{\lceil k/2 \rceil}} \cdot N^{2-\frac{1}{\lceil k/2 \rceil}} + OUT)$, where OUT is the size of the output.*

Proof. Consider the following query $q_k \in \mathcal{C}$ over a database DB_k of size $\bar{N}$:

$$ans(\bar{X}) \leftarrow r_1(X_1, X_2) \wedge r_2(X_2, X_3) \wedge \cdots \wedge r_k(X_k, X_1),$$

where $r_1, \ldots, r_k$ are binary relation symbols, not necessarily distinct, and for each $i \in [k]$, r_i has N_i tuples. Its associated hypergraph G_k is a simple cycle having k vertices and k edges, and both $hw(G_k) = 2$ and $fhw(G_k) = 2$ holds.

Let $\delta > 0$ be a number that will be determined later. For each $i \in [k]$, let $r'_i \subseteq r_i$ be the (horizontal) fragment of r_i consisting of the tuples $\{\langle v, v' \rangle \in r_i \mid v$ *occurs in more than* δ *tuples in* $r_i\}$. Let q'_i be the variant of query q_k where the atom $r_i(X_i, X_{i+1})$ is replaced by $r'_i(X_i, X_{i+1})$, and consider a hypertree decomposition for q'_i of the form shown on the right in Figure 1, with X_i playing the role of X_2. Note that the number of tuples in the projection $r'_i(X_i, \_)$ occurring in the decomposition vertices is at most N_i/δ, so that q_i can be evaluated in $O(\bar{N}N_i/\delta + OUT_i)$. The OUT_i answers of this modified query are all those answers of q_k such that X_i is instantiated with any value of its domain having degree at least δ (if any). After the evaluation of all such modified queries, we get all the OUT_δ answers of q_k over DB_k where some variable takes a value having degree at least δ. The total time used for such computations is $O(\bar{N}\bar{N}/\delta + OUT_\delta)$. Note that, having these answers, each domain value having degree at least δ is no longer needed in any relation.

Then, we take care of the remaining fragment: consider the query q''_k: $ans''(\bar{X}) \leftarrow r''_1(X_1, X_2) \wedge r''_2(X_2, X_3) \wedge \cdots \wedge r''_k(X_k, X_1)$ over the database DB''_k with the relations $r''_i = r_i \setminus r'_i$, for each $i \in [k]$. We evaluate this query by using a tree decomposition of the form shown in the center of Figure 1 with two big vertices, each one covering half of the query. The width w of this decomposition is $\lceil k/2 \rceil$. Let r_s and r_t be the smallest relations in the database DB''_k, having number of tuples N_r and N_t, respectively. We can always choose the decomposition in such a way that r_s and r_t occur

in different vertices. Consider now the evaluation of the subproblem comprising the relation r_s: because every value has degree at most δ, every tuple of r_s can have at most δ extensions to tuples in the subsequent relation in the subproblem, say r_z; the same happens for every tuple in r_z, and so on for all the atoms in this subproblem. The evaluation of these atoms takes $O(N_s\delta^{w-1})$ time and, similarly, we need $O(N_t\delta^{w-1})$ to evaluate the subproblem occurring in the other vertex of the decomposition, for a total cost of $O((N_s + N_t)\delta^{w-1})$. Because r_s and r_t are the smallest relations, it can be seen easily that $(N_s + N_t) \leq 2\bar{N}/k$, so that the total cost for evaluating q''_k over DB''_k using such a decomposition is $O(2\delta^{w-1}\bar{N}/k + \mathit{OUT}'')$, where OUT'' is the size of the output relation $ans''(\bar{X})$. By equating the cost of using the two kinds of decompositions, we can compute the value $\delta = (k/2\bar{N})^{1/w}$ that guarantees that we obtain the same evaluation cost for either kind of fragment and for every possible input database. With this value, the total cost for evaluating q_k on any given database DB_k of size $\bar{N}$ is $O((2/k)^{1/w}N^{2-1/w} + \mathit{OUT})$, where OUT is the size of the output relation $ans(\bar{X})$. □

4 Discussion and Open problems

Recent advances have shown that powerful structural decompositions such as the (fractional) hypertree width or the submodular width may lead to effective algorithms even for very simple queries, and not just for long and complex ones (possibly involving atoms with large arities), which have been the typical targets for these sophisticated techniques. Moreover, we have seen that the worst case upper bounds of the so-called purely structural decomposition methods, where the choice of the decomposition depends only on the query hypergraph, are not tight.

Contrasted with such methods, the notion of submodular width, which is based on data-dependent decompositions, is more powerful and in fact characterises the frontier of fixed-parameter tractability for conjunctive queries. The algorithm used in [19] is based on the computation of almost uniform horizontal fragments of all possible subproblems that can be evaluated in polynomial-time. Very roughly, uniform means here that, in each fragment and for any subset of variables of a subproblem, partial solutions over these variables extend to full solutions of the subproblem in a similar way. Whenever a subproblem does not meet this condition, it can be split in more subproblems leading to new fragments. The subproblems of each new instance associated with a fragment are kept locally consistent. That is, tuples that are not useful in that fragment are filtered out, and the procedure continues looking for new small, consistent and uniform subproblems to deal with. This procedure works in fixed-parameter polynomial time, where the parameter is the size of the query. Eventually, under the promise that the submodular width is below some fixed threshold w (which is used to define precisely the above process), there should exist some tree decomposition whose subproblems are among those that we have computed. Furthermore, whenever for an instance associated with some fragment its subproblems are not empty, because they are pairwise consistent we can conclude that the given query has some answer on the input database; otherwise, we conclude that there are no answers (see also [14] for such consistency properties).

Marx's algorithm is quite involved and its actual cost is quite high even for queries with a few variables. The PANDA algorithm described in [17] aims at obtaining a similar worst case bound, but is more suitable to effective implementations. However, because of its different uniformization procedure, its cost involves a $(\log n)^h$ factor, where

h is the number of atoms in the query, which is not allowed in fixed-parameter polynomial time algorithms (the parameter cannot occur as the exponent of a number that depends on the input size). In fact, it is still open whether or not such a bad factor can be avoided, so that an effective fixed-parameter polynomial time algorithm can be implemented. An algorithm for computing the submodular width of a given hypergraph is also missing, as well as a possible approximation of this notion. Furthermore, the frontier of polynomial-time tractability for conjunctive queries involving atoms of arbitrary arities is still unknown, and it is even unknown whether it can actually be charted or not.

References

1. Aberger, C.R., Lamb, A., Tu, S., Nötzli, A., Olukotun, K., Ré, C.: Emptyheaded: A relational engine for graph processing. ACM TODS **42**(4), 20:1–20:44 (2017).
2. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997).
3. Atserias, A., Grohe, M., Marx, D.: Size bounds and query plans for relational joins. SIAM J. on Comp. **42**(4), 1737–1767 (2013)
4. Fischl, W., Gottlob, G., Pichler, R.: General and fractional hypertree decompositions: Hard and easy cases. In: Proc. of PODS'18 (2018)
5. Ghionna, L., Granata, L., Greco, G., Scarcello, F.: Hypertree decompositions for query optimization. In: Proc. of ICDE'07. pp. 36–45 (2007)
6. Ghionna, L., Greco, G., Scarcello, F.: H-DB: A Hybrid Quantitative-structural SQL Optimizer. In: Proc. of CIKM '11. pp. 2573–2576 (2011)
7. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. JCSS **64**(3), 579–627 (2002)
8. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. JCSS **66**(4), 775–808 (2003)
9. Gottlob, G., Greco, G., Scarcello, F.: Treewidth and hypertree width. In: Tractability: Practical Approaches to Hard Problems, pp. 3–38. Cambridge University Press (2014).
10. Gottlob, G., Lee, S.T., Valiant, G., Valiant, P.: Size and Treewidth Bounds for Conjunctive Queries. J.ACM **59**(3), 1–35 (2012)
11. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: NP-hardness and tractable variants. J.ACM **56**(6), 30:1–30:32 (2009)
12. Greco, G., Leone, N., Scarcello, F., Terracina, G.: Structural decomposition methods: Key notions and database applications. In: A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years., Studies in Big Data, vol. 31, pp. 253–267. Springer
13. Greco, G., Scarcello, F.: Greedy strategies and larger islands of tractability for conjunctive queries and constraint satisfaction problems. Inf. Comput. **252**, 201–220 (2017).
14. Greco, G., Scarcello, F.: The power of local consistency in conjunctive queries and constraint satisfaction problems. SIAM J. Comput. **46**(3), 1111–1145 (2017).
15. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. ACM Trans. on Alg. **11**(1), 4:1–4:20 (2014)
16. Joglekar, M., Ré, C.: It's all a matter of degree - using degree information to optimize multiway joins. Theory Comput. Syst. **62**(4), 810–853 (2018).
17. Khamis, M.A., Ngo, H.Q., Suciu, D.: What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In: Proc. of PODS 2017, pp. 429–444.
18. Marx, D.: Approximating fractional hypertree width. ACM T. on Alg. **6**(2), 29:1–17 (2010)
19. Marx, D.: Tractable hypergraph properties for constraint satisfaction and conjunctive queries. J.ACM **60**(6), 42:1–42:51 (2013)
20. Robertson, N., Seymour, P.: Graph minors. II. Algorithmic aspects of tree-width. J. of Alg. **7**(3), 309–322 (1986)