Abstract

Evaluating conjunctive queries and solving constraint satisfaction problems are fundamental problems in database theory and artificial intelligence, respectively. These problems are NP-hard, so that several research efforts have been made in the literature for identifying tractable classes, known as islands of tractability, as well as for devising clever heuristics for solving efficiently real-world instances.

Many heuristic approaches are based on enforcing on the given instance a property called local consistency, where (in database terms) each tuple in every query atom matches at least one tuple in every other query atom. Interestingly, it turns out that, for many well-known classes of queries, such as for the acyclic queries, enforcing local consistency is even sufficient to solve the given instance correctly. However, the precise power of such a procedure was unclear, but for some very restricted cases.

The paper provides full answers to the long-standing questions about the precise power of algorithms based on enforcing local consistency. In particular, the paper deals with both the general framework of tree projections, where local consistency is enforced among arbitrary views defined over the given database instance, and the specific cases where such views are computed according to so-called structural decomposition methods, such as generalized hypertree width, component hypertree decompositions, and so on.

The classes of instances where enforcing local consistency turns out to be a correct query-answering procedure are however not efficiently recognizable. In fact, the paper finally focuses on certain subclasses defined in terms of the novel notion of greedy tree projections. These latter classes are shown to be efficiently recognizable and strictly larger than most islands of tractability known so far, both in the general case of tree projections and for specific structural decomposition methods.
1 Introduction

1.1 Acyclic Conjunctive Queries

Answering conjunctive queries to relational databases is a basic problem in database theory, and it is equivalent to many other fundamental problems, such as conjunctive query containment and constraint satisfaction. Recall that 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 (see, e.g., [7]). This class contains all queries \( Q \) whose associated query hypergraph \( H_Q \) is acyclic\(^1\), where \( H_Q \) 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. It is well known that acyclic queries enjoy a number of highly desirable properties, recalled next.

First, acyclic queries can be efficiently solved. From any acyclic query, we can build (in linear time) a join tree [8], which is a tree whose vertices correspond to the various atoms and where the subgraph induced by vertices containing any given variable is a tree. According to Yannakakis’s algorithm [52], Boolean acyclic queries can be evaluated by processing any of their join trees bottom-up, by performing upward semijoins between the relations associated with the query atoms, thus keeping the size of the intermediate relations small. At the end, if the relation associated with the root of the join tree is not empty, then the answer of the query is not empty. For non-Boolean queries, after the bottom-up step described above, one can perform the opposite top-down step by filtering each child vertex from those tuples that do not match with its parent tuples. The filtered database, called full reducer, then enjoys the global consistency property: every tuple in every relation participates in some solution. By exploiting this property, all solutions can be computed with a backtrack-free procedure (i.e., with backtracks used to look for further solutions, and never caused by wrong choices).

Second, the class of acyclic instances coincides with the class of queries where local consistency entails global consistency. We say that local (also, pairwise) consistency holds if the relations associated with the query atoms are not empty and we do not miss any tuple by taking semijoins between any pair of them. The acyclic instances that fulfil this property also fulfil the global consistency property [7]. Note that local consistency may easily be enforced by taking the semijoins between all pairs of atoms until a fixpoint is reached. Therefore, in abstract terms, any acyclic query can be answered by means of “local” computations only, without any additional knowledge about the whole structure, in

\(^1\)For completeness, observe that different notions of hypergraph acyclicity have been proposed in the literature. This paper follows the standard definition of acyclic conjunctive queries, so that hypergraph acyclicity always refers to the most liberal notion, known as \( \alpha \)-acyclicity [18].
particular without computing any join tree of the query. In addition, and more surprisingly, if a class of instances can be answered by means of this approach, then it only contains acyclic instances \cite{7}.

Finally, acyclicity is efficiently recognizable. Deciding whether a hypergraph is acyclic is feasible in linear time \cite{50}, and also in deterministic logspace. In fact, this latter property follows from the fact that hypergraph acyclicity belongs to SL \cite{23}, and that SL is equal to deterministic logspace \cite{45}. Note that, in the light of this property and the first one above, these queries identify a so-called (accessible) “island of tractability” for the query answering problem \cite{38}.

1.2 Generalization of Acyclicity

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.

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, such as the (generalized) hypertree \cite{24}, fractional hypertree \cite{35}, spread-cut \cite{14}, and component hypertree \cite{26} decompositions. These methods aim at transforming a given cyclic hypergraph into an acyclic one, by organizing 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, with a cost that is exponential in the cardinality of the largest cluster, also called width of the decomposition, and polynomial if this width is bounded by some constant.

Despite their different technical definitions, there is a simple mathematical framework that encompasses all the above decomposition methods, which is the framework of the tree projections \cite{29}. In this setting, a query $Q$ is given together with a set $V$ of atoms, called views, which are defined over the variables in $Q$. The question is whether (parts of) the views can be arranged as to form a tree projection (playing the role of a decomposition tree), i.e., a novel acyclic query that still “covers” $Q$. By representing $Q$ and $V$ via the hypergraphs $\mathcal{H}_Q$ and $\mathcal{H}_V$, where hyperedges one-to-one correspond with query atoms and views, respectively, the tree projection problem reveals its graph-theoretic nature. For a pair of hypergraphs $\mathcal{H}_1, \mathcal{H}_2$, let $\mathcal{H}_1 \leq \mathcal{H}_2$ denote that each hyperedge of $\mathcal{H}_1$ is contained in some hyperedge of $\mathcal{H}_2$. Then, a tree projection of $\mathcal{H}_Q$ w.r.t. $\mathcal{H}_V$ is any acyclic hypergraph $\mathcal{H}_a$ such that $\mathcal{H}_Q \leq \mathcal{H}_a \leq \mathcal{H}_V$. If such a hypergraph exists, then we say that the pair of hypergraphs $(\mathcal{H}_Q, \mathcal{H}_V)$ has a tree projection.

Example 1.1 Consider the conjunctive query

$$Q_0 : \ r_1(A, B, C) \land r_2(A, F) \land r_3(C, D) \land r_4(D, E, F) \land \ r_5(E, F, G) \land r_6(G, H, I) \land r_7(I, J) \land r_8(J, K),$$

\footnote{Actually, this classical result holds only for queries where every relation symbol is used at most once. The precise power of local computations in the general case is identified in this paper (for acyclic queries too).}
whose associated hypergraph $H_{Q_0}$ is depicted in Figure 1 together with other hypergraphs that are discussed next.

To answer $Q_0$, assume that a set $V_0$ of views is available comprising some views, called query views, playing the role of query atoms, plus four additional views. The set of variables of each view is a hyperedge in the hypergraph $H_{V_0}$ (query views are depicted as dashed hyperedges). In the middle between $H_{Q_0}$ and $H_{V_0}$, Figure 1 reports the hypergraph $H_a$ which covers $H_{Q_0}$, and which is in its turn covered by $H_{V_0}$—e.g., $\{C, D\} \subseteq \{A, B, C, D\} \subseteq \{A, B, C, D, H\}$. Since $H_a$ is in addition acyclic (just check the join tree $JT_a$ in the figure), $H_a$ is a tree projection of $H_{Q_0}$ w.r.t. $H_{V_0}$.

Observe that, in the tree projection framework, views can be arbitrary, i.e., they do not depend on the specific conjunctive query $Q$, and can be reused to answer different queries. In particular, views may be the materialized output of any procedure over the database, possibly much more powerful than conjunctive queries. Moreover, it is known and easy to see that any decomposition method based on clustering subproblems can be viewed as an instance of this general setting, identifying a specific set of views to answer a given query $Q$ efficiently (see Section 2 and Section 4).

For example (see, e.g., [4, 30, 32]), for any fixed natural number $k$, the generalized hypertree decomposition method associates with any query $Q$ a set $v$-$h_{Q}(Q)$ of views, containing one distinct view over each set of variables that can be covered by at most $k$ query-atoms. For any hypergraph $H$, let $H^k$ be the hypergraph whose hyperedges are all possible sets obtained by the union of at most $k$ hyperedges of $H$, and notice that $H^k_{Q}$ is precisely the hypergraph associated with $v$-$h_{Q}(Q)$. A query $Q$ has generalized hypertree width bounded by $k$ if, and only if, there is a tree projection of $H_{Q}$ w.r.t. $H^k_{Q}$.

For another example, we recall the tree decomposition method [17, 21], based on the notion of treewidth [42], which is the most general decomposition method over classes of bounded-arity queries (see, e.g., [22, 34]). For any fixed natural number $k$, the method defines the set $v$-$t_{Q}(Q)$ of views containing one distinct view over each set of at most $k + 1$ variables occurring in $Q$. Let $H^k_{Q}$ be the hypergraph associated with $v$-$t_{Q}(Q)$, i.e., the hypergraph whose hyperedges
are all possible sets of at most \( k + 1 \) variables. Then, a query \( Q \) has \textit{treewidth} bounded by \( k \) if, and only if, there is a tree projection of \( H_Q \) w.r.t. \( H_Q^k \) (see, e.g., [30] [32]).

In fact, the notion of tree projection is quite natural and may be exploited in different applications where hypergraphs naturally represent structural properties of input instances. For example, Adler [3] pointed out that the notion of acyclicity for a conjunctive query with negation \( Q \), as defined in [19], can be immediately recast as the existence of a tree projection of \( H_Q \) w.r.t. \( H_Q^+ \), where the hyperedges of \( H_Q^+ \) are the sets of variables occurring in the positive atoms of \( Q \) only, while the hyperedges of \( H_Q \) correspond to all atoms, including the negative ones. Then, we can generalize this notion to obtain larger classes of tractable instances, by saying that a query with negation \( Q \) has \textit{generalized hypertree width} at most \( k \) if the pair \((H_Q, H_Q^k)\) has a tree projection. Indeed, following the same reasoning as in [19], it is easy to see that, given such a tree projection, the query \( Q \) can be evaluated in polynomial time.

1.3 Open Questions About Tree Projections and Structural Decomposition Methods

The interest on the tree projection framework goes back to the eighties, when it was noticed that queries that admit a tree projection can be evaluated in polynomial time [29] (see, also, [44]). Thus, tree projections smoothly preserve the first crucial property of acyclic queries discussed in Section 1.1. Our knowledge on the preservation of the other properties of acyclic queries was less clear, instead. In fact, the following two questions have been posed in the literature for the general tree projection framework as well as for structural decomposition methods specifically tailored to deal with classes of queries without a fixed arity bound. Such questions were in particular open for the generalized hypertree decomposition method, which on classes of unbounded-arity queries is a natural counterpart of the tree decomposition method.

(Q1) \textbf{What is the precise power of local-consistency based algorithms?}

This question was firstly raised in [7] and specifically for the general case of tree projections in [44], and remained open so far, despite it was attacked via different approaches and proof techniques, which gave some partial results, reported below.

Let \( \mathcal{V} \) be an arbitrary set of views, which also contains the query views representing the atoms of a given query \( Q \). Let \text{lc}(\mathcal{V}, DB)\) denote that the views in \( \mathcal{V} \) evaluated over a database DB enjoy the local consistency property, i.e., they are non-empty and we do not miss any tuple by taking the semijoin between any pair of views. Let \text{red}(\mathcal{V}, DB)\) be the \textit{reduct} of DB according to \( \mathcal{V} \), computed by taking all possible semijoins until a fixpoint is reached. More precisely, \text{red}(\mathcal{V}, DB)\) is the (set-inclusion) maximal subset of DB such that \text{lc}(\mathcal{V}, DB)\) holds, or \text{red}(\mathcal{V}, DB) = \emptyset, whenever such a maximal subset does not exist. Let \text{gc}(\mathcal{V}, DB, Q)\) denote that the global consistency property holds, i.e., every tuple in every query view (evaluated over DB) participates in the query answer. Let
\(Q^{DB} \neq \emptyset\) denote that the answer of \(Q\) on DB is not empty. Then, the picture emerging from the literature is as follows:

- The existence of a tree projection of \(\mathcal{H}_Q\) w.r.t. \(\mathcal{H}_V\) entails that, \(\forall DB, \text{lc}(\mathcal{V}, DB) \Rightarrow gc(\mathcal{V}, DB, Q)\) [44]. In words, the existence of a tree projection is a sufficient condition for the global consistency property to hold, whenever the database is local consistent. Thus, if a tree projection exists, then both deciding whether the query is not empty and computing a query answer (if any) are feasible in polynomial time, by enforcing local consistency. Observe that such a procedure is based on local computations only, and hence there is no need to actually compute a tree projection. This is a remarkable result, since computing a tree projection is instead not feasible in polynomial time, unless \(P = NP\) [26]. It was conjectured that the existence of a tree projection is also a necessary condition for having this property [29, 44].

- Consider classes of bounded-arity queries \(Q\), and the tree decomposition method, hence the view set \(v-tw_k(Q)\) with its associated hypergraph \(H^k_Q\). For any database DB, let \(d-tw_k(Q, DB)\) be the database obtained by associating each view in \(v-tw_k(Q)\) with the cartesian product of the set of constants that variables occurring in it may take. It is known that \(\forall DB, (\text{red}(v-tw_k(Q), d-tw_k(Q, DB)) \neq \emptyset) \Rightarrow (Q^{DB} \neq \emptyset)\) if [15], and only if [6], there is a tree projection of \(\mathcal{H}_Q\) w.r.t. \(\mathcal{H}^k_Q\), for some core \(Q'\) of \(Q\). In fact, the result holds for any query \(Q'\) that is homomorphically equivalent to \(Q\), denoted by \(Q' \approx_{\text{hom}} Q\) (instead of just for a core, which is any smallest one). This result provides a necessary and sufficient condition for query answering via local consistency, without computing any tree-decomposition of such a subquery \(Q'\), which would be an NP-hard task [15]. Observe that the necessary condition holds only for structures of bounded arity, and the result provides only information about the decision problem (i.e., checking whether the answer is empty or not).

- For the general case of queries \(Q\) with unbounded arity, consider the generalized hypertree decomposition method and hence the view set \(v-hw_k(Q)\), containing one distinct view over each set of variables that can be covered by at most \(k\) query-atoms, and its associated hypergraph \(H^k_Q\). Moreover, for any database DB, let \(d-hw_k(Q, DB)\) be the database obtained by associating each view in \(v-hw_k(Q)\) with the (natural) join of all query-views over which it is defined. It is known that \(\forall DB, (\text{red}(v-hw_k(Q), d-hw_k(Q, DB)) \neq \emptyset) \Rightarrow (Q^{DB} \neq \emptyset)\) if there exists a tree projection of \(\mathcal{H}_Q\) w.r.t. \(\mathcal{H}^k_Q\), where \(Q'\) is any query such that \(Q' \approx_{\text{hom}} Q\) [12]. Note that, when we focus on generalized hypertree decompositions, instead of looking at views in \(v-hw_k(Q)\) and tree projections, we may directly look at the consistency between every pair of sets of \(k\) atoms, also called \(k\)-local consistency. Hence, the result states a sufficient condition for deciding whether the answer is empty or not by enforcing \(k\)-local consistency, (again) without actually identifying such a subquery \(Q'\) and without computing a generalized hypertree decomposition of
\( Q' \), which are both NP-hard tasks. It was open whether the condition is also necessary \[12\]. Moreover, as in the above point about tree decompositions, the relationship with global consistency and hence with the related problem of computing solutions was missing.

From these results, it emerges that the precise power of local-consistency based computations and of their relationships with tree projections and with the other structural decomposition methods (in particular, tree decompositions and generalized hypertree decompositions) was far from being clear: Is it possible that there are queries where such local computations do work even if no decomposition (or tree projection) exists?

For instance, from the above recent results based on homomorphically equivalent subqueries for tree decompositions and generalized hypertree decompositions, one may deduce that the mentioned conjecture in \[29, 44\] (i.e., that local consistency implies global consistency if, and only if, a tree projection of the query hypergraph exists) may not hold, in general. This is because in the case of queries with multiple occurrences of the same relation symbol, the concept of core of the query plays a crucial role \[15\], as it should be clear from the next example.

**Example 1.2** Consider the following queries:

\[
Q_1 : r(A, B) \land r(B, C) \land r(C, D) \land r(D, A) \\
Q_2 : r(A, B) \land r(B, C) \land r(D, C) \land r(A, D) \\
Q_3 : r(B, A) \land r(C, B) \land r(C, D) \land r(D, A)
\]

These queries are completely equivalent as far as their hypergraphs are concerned, since \( \mathcal{H}_{Q_1} = \mathcal{H}_{Q_2} = \mathcal{H}_{Q_3} \). However, \( Q_1 \) is already a core, while a core of \( Q_2 \) (resp., \( Q_3 \)) is the acyclic sub-query \( r(A, B) \land r(B, C) \) (resp., \( r(C, D) \land r(D, A) \)). Thus, by focusing on \( Q_2 \) and \( Q_3 \) rather than on their cores, we could overestimate their intricacy.

However, the above conjecture might still hold in the original setting considered in \[29\], where all relation symbols in a query are distinct.

(Q2) Are there unexplored islands of tractability based on tree projections? An island of tractability in the tree projection framework is a class \( C \) of pairs \((Q, \mathcal{V})\) that can be efficiently recognized, and such that \( Q \) can be efficiently evaluated on every database, by possibly exploiting the views that are available in \( \mathcal{V} \).

Many specializations of tree projections, such as tree decompositions \[42\], hypertree decompositions \[24\], component decompositions \[26\], and spread-cuts decompositions \[14\], define islands of tractability whenever some fixed bound is imposed on their widths. This is also the case for fractional hypertree decompositions \[35\], whenever the resources sufficient for computing their \( O(w^3) \) approximation \[40\] are used as available views. However, this is not the case.
for general tree projections. Indeed, while Goodman and Shmueli [29] observed that queries that admit a tree projection can be evaluated in polynomial time, Gottlob et al. [26] proved that checking whether a tree projection exists or not is an NP-hard problem. Hence, the class $C_{tp} = \{(Q,V) \mid H_Q \text{ has a tree projection w.r.t. } H_V\}$, which includes all the above mentioned islands of tractability, is not an island of tractability in its turn. In fact, in addition to the above result, we also know that:

- Deciding whether a tree projection of $H_Q$ w.r.t. $H_Q^{tk}$ (corresponding to a tree decomposition) exist is feasible in time $O(2^{ck^2} \times n)$, where $n$ is the size of $H_Q$, $k$ is the treewidth, and $c$ is a constant [9], hence in linear time for a fixed width $k$.

- The problem remains NP-hard for the case of generalized hypertree decompositions, that is, when we have to decide the existence of a tree projection of $H_Q$ w.r.t. $H_Q^k$, even if $k$ is a fixed number (greater than 2) [26].

Moreover, recall that the sufficient conditions we have discussed in the previous point (Q1) do not identify (accessible) islands of tractability, because their recognition problems are NP-hard, too. Such conditions are particularly useful in those settings where it is intractable to compute any tree projection, so that answers are computed via procedures enforcing local consistency. However, having a tree projection at hands allows queries to be evaluated more efficiently w.r.t. techniques based on “blind” local-consistency enforcing. Intuitively, by having such a projection $H_a$ and hence a join tree for $H_a$, we are able to exploit all the well known algorithms developed for acyclic queries. In particular, in this approach, only the views occurring in the join tree are involved in the query evaluation, while all available views should be used if no tree projection is available. Furthermore, the number of semijoin operations to be performed having the join tree is at most the number of nodes in such a tree and does not depend on the database, as it happens instead while enforcing local consistency. Therefore, a natural question is whether there is any subclass of $C_{tp}$, at least including all the tractable classes mentioned above, which identifies an actual island of tractability where tree projections can be computed efficiently.

### 1.4 Contribution

In this paper, we provide a clear picture of the power of tree projections and structural decomposition methods, by answering the two questions illustrated above.

It is worthwhile noting that our answers, summarized below, find applications in all those problems that can be solved efficiently on acyclic and quasiacyclic instances, even outside the Database area. In particular, our results can be exploited immediately for solving Constraint Satisfaction Problems (CSPs) where constraints are represented as finite relations encoding allowed tuples of values (see, e.g., [22]).
(Q1) The first achievement of this paper is to solve the long-standing question about the power of local-consistency based computations, by addressing in the analysis both the decision problem of checking whether the query is not empty, and the problem of characterizing a necessary and sufficient condition guaranteeing that local consistency entails global consistency, which is useful from the query answering perspective.

Concerning the decision problem of checking whether the query has a solution, we show that the sufficient conditions identified for some specializations of tree decompositions are also necessary, even in the most general framework. However, the technical machinery needed for obtaining our results is quite different from the one used in [6] for tree decompositions, which does not work when we have arbitrary signatures or arbitrary views. Our first contribution is to show that:

The following are equivalent:

1. For every database $DB$, $lc(V, DB)$ entails $Q^{DB} \neq \emptyset$.
2. There is a subquery $Q'$ $\cong_{hom} Q$ for which $(H_{Q'}, H_V)$ has a tree projection.

Our second contribution is then to single out the (stronger) conditions under which local consistency entails global consistency. We show that finding a necessary and sufficient condition requires to exploit possible endomorphisms of the query. It emerged that to characterize when, at local consistency, an atom $p$ contains all, and only, the correct tuples of the query $Q$ projected over the variables $vars(p) = \{X_1, ..., X_n\}$ of $p$, we must look for tree projections of some “output-aware” substructures of $Q$. We say that $\{X_1, ..., X_n\}$ is $tp$-covered in $Q$ (w.r.t. $V$) if there is a tree projection of $(H_{Q_\Delta}, H_V)$, where $Q_\Delta$ is a core of the novel query $Q \land r(X_1, ..., X_n)$, in which $r$ is a fresh relation symbol. Intuitively, $r$ is used to force any such a core to contain the desired variables $\{X_1, ..., X_n\}$. It turns out that, for having global consistency guaranteed by local consistency, for each query atom $p$, a tree projection $H_p$ of such a $Q_p$ must exist.

The following are equivalent:

1. For every database $DB$, $lc(V, DB)$ entails $gc(V, DB, Q)$.
2. For each query atom $q$, $vars(q)$ is $tp$-covered in $Q$.

Thus, if (2) holds and one is interested in computing query answers over output variables included in some query atom, then all solutions are immediately available. In fact, the above result comes in the paper as a specialization of a more general result dealing with those cases where one is interested in computing answers over an arbitrary subset of variables covered by some available view.
Moreover, observe that in the above condition different tree projections for different query atoms are allowed. That is, global consistency can hold even if there is no tree projection that is able to cover all query atoms at once. However, if every relation symbol is used at most once in the query, it is easy to see that (2) is equivalent to requiring that a tree projection of the whole query exists. Hence, the conjecture of [29] about the necessity of having a tree projection of the query does not hold in general, but it does hold for such a restricted setting (in fact, the one considered in [29]).

Actually, in this informal statement we have implicitly assumed databases where views are not more restrictive than the query; otherwise, using such views may clearly lead to missing some tuple in the query answer. Note that this condition trivially holds whenever views are computed from parts of the query (i.e., they are in fact subqueries), which happens in structural decomposition methods. However, this is not necessarily true if one would like to exploit existing materialized views. Anyway, we show that soundness of query answers is always guaranteed. If views are too restrictive w.r.t. $Q$, then we may just miss completeness.

(Q1: Application to Decomposition Methods) As a direct consequence of our contribution w.r.t. question (Q1), we get in a unique result the generalization of all tractability results known for purely structural decompositions methods (because all of them are specializations of the notion of tree projections). Moreover, we provide the precise characterization of the power of $k$-local consistency for classes of queries without a fixed bound on the arity, which was missing in [6] and [12].

In particular, we provide a necessary and sufficient condition such that $k$-local consistency entails global consistency, which is useful for computing solutions. Furthermore, concerning the decision problem (query non-emptiness), we show that the sufficient condition identified in [12] is in fact necessary, too:

| The following are equivalent: |
|--------------------------------|
| (1) For every database $DB$, $\text{red}(v-hw_k(Q), d-hw_k(Q, DB)) \neq \emptyset$ entails $Q^{DB} \neq \emptyset$. |
| (2) $Q$ has a core having generalized hypertree width at most $k$. |

We point out that the result is not an immediate corollary of the previous one about tree projections (by setting $\mathcal{H}_V = \mathcal{H}_Q^k$, where $\mathcal{H}_Q^k$ is the hypergraph where each hyperedge is the set of variables occurring in some group of at most $k$ query-atoms). Indeed, let $Q'$ be any core of $Q$, and recall that $Q'$ may be much smaller than $Q$. Thus, the set of views that can be used to form a $k$-width generalized hypertree decomposition of $Q'$ only come from groups of at most $k$ atoms occurring in $Q'$. It follows that this set can be much smaller than $V_k$, which is built from the full query $Q$. For another difference between our general result and the above one, note that the database $d-hw_k(Q, DB)$ for the available
views, over which local consistency is considered, is functionally determined by the relations of query atoms in DB (instead of being almost arbitrary).

Note that, for \( k = 1 \), local consistency is required to hold only on the query views playing the role of the original query atoms. We thus obtain the precise characterization of the power of local consistency in acyclic queries, generalizing the classical result in [7] given for queries without multiple occurrences of the same relation symbol: for every database DB, local consistency (of query views) entails \( Q^{\text{con}} \neq \emptyset \) if, and only if, \( Q \) has an acyclic core.

(Q2) As discussed above, the classes of instances where enforcing local consistency is a correct query-answering procedure are not efficiently recognizable. Therefore, it is natural to look for subclasses that are efficiently recognizable and that are strictly larger than the islands of tractability known so far. Addressing this issue is the second main achievement of the paper. To this end, we exploit the game-theoretic characterization of tree projections in terms of the Captain and Robber game [30]. The game is played on a pair of hypergraphs \((H_1, H_2)\) by a Captain controlling, at each move, a squads of cops encoded as the nodes in a hyperedge \( h \in \text{edges}(H_2) \), and by a Robber who stands on a node and can run at great speed along the edges of \( H_1 \), while being not permitted to run through a node that is controlled by a cop. In particular, the Captain may ask any cops in the squad \( h \) to run in action, as long as they occupy nodes that are currently reachable by the Robber, thereby blocking an escape path for the Robber. While cops move, the Robber may run through those positions that are left by cops or not yet occupied. The goal of the Captain is to place a cop on the node occupied by the Robber, while the Robber tries to avoid her capture. The Captain has a winning strategy if, and only if, there is a tree projection of \( H_1 \) w.r.t. \( H_2 \).

1. We define the notion of greedy strategies, which are winning strategies for the Captain, possibly non-monotone, where it is required that all cops available at the current squad \( h \) and reachable by the Robber enter in action. If all of them are in action, then a new squad \( h' \) is selected, again requiring that all the active cops, i.e., those in the frontier, enter in action. In the Captain and Robber game, it is known that there is no incentive for the Captain to play a strategy that is not monotone [30]. Instead, by focusing on greedy strategies, we can exhibit examples where there exists non-monotone winning strategies but no monotone winning one.

2. We show that greedy strategies can be computed in polynomial time, and that based on them (even on non-monotone ones) it is possible to construct, again in polynomial time, tree projections, which are called greedy. Therefore, the class \( C_{\text{gtp}} \subset C_{\text{tp}} \) of all greedy tree projections turns out to be an island of tractability.

3. Finally, we show that \( C_{\text{gtp}} \) properly includes most previously known islands of tractability (based on structural properties), precisely because of the power of non-monotonic strategies. In particular, the novel notion of
greedy tree projections allows us to define new islands of tractability from any known structural decomposition method, such as the greedy (generalized) hypertree decomposition or the greedy component decomposition, which are tractable and strictly more powerful than their original versions.

1.5 Organization

The paper is organized as follows. Section 2 illustrates some basic notions and concepts. The characterization of the power of local consistency is given in Section 3 while its application to structural decomposition methods is reported in Section 4. Islands of tractability for tree projections are singled out in Section 5, and an application of the results to structures having “small” arities is presented in Section 6. A few further remarks and open issues are discussed in Section 7.

2 Preliminaries

Hypergraphs and Acyclicity. A hypergraph \( H \) is a pair \( (V, H) \), where \( V \) is a finite set of nodes and \( H \) is a set of hyperedges such that, for each \( h \in H \), \( h \subseteq V \). If \( |h| = 2 \) for each (hyper)edge \( h \in H \), then \( H \) is a graph. For the sake of simplicity, we always denote \( V \) and \( H \) by nodes \((H)\) and edges \((H)\), respectively.

A hypergraph \( H \) is acyclic (more precisely, \( \alpha \)-acyclic [18]) if, and only if, it has a join tree [8]. A join tree \( JT \) for a hypergraph \( H \) is a tree whose vertices are the hyperedges of \( H \) such that, whenever a node \( X \in V \) occurs in two hyperedges \( h_1 \) and \( h_2 \) of \( H \), then \( h_1 \) and \( h_2 \) are connected in \( JT \), and \( X \) occurs in each vertex on the unique path linking \( h_1 \) and \( h_2 \). In words, the set of vertices in which \( X \) occurs induces a (connected) subtree of \( JT \). We will refer to this condition as the connectedness condition of join trees.

Tree Decompositions. A tree decomposition [42] of a graph \( G \) is a pair \( (T, \chi) \), where \( T = (N, E) \) is a tree, and \( \chi \) is a labeling function assigning to each vertex \( v \in N \) a set of vertices \( \chi(v) \subseteq \text{nodes}(G) \), such that the following conditions are satisfied: (1) for each node \( Y \in \text{nodes}(G) \), there exists \( p \in N \) such that \( Y \in \chi(p) \); (2) for each edge \( \{X, Y\} \in \text{edges}(G) \), there exists \( p \in N \) such that \( \{X, Y\} \subseteq \chi(p) \); and (3) for each node \( Y \in \text{nodes}(G) \), the set \( \{p \in N \mid Y \in \chi(p)\} \) induces a (connected) subtree of \( T \). The width of \( (T, \chi) \) is the number \( \max_{p \in N}(|\chi(p)| - 1) \).

The Gaifman graph of a hypergraph \( H \) is defined over the set \( \text{nodes}(H) \) of the nodes of \( H \), and contains an edge \( \{X, Y\} \) if, and only if, \( \{X, Y\} \subseteq h \) holds, for some hyperedge \( h \in \text{edges}(H) \). The treewidth of \( H \) is the minimum width over all the tree decompositions of its Gaifman graph. Deciding whether a given hypergraph has treewidth bounded by a fixed natural number \( k \) is known to be feasible in linear time [9].

(Generalized) Hypertree Decompositions. A hypertree for a hypergraph \( H \) is a triple \( (T, \chi, \lambda) \), where \( T = (N, E) \) is a rooted tree, and \( \chi \) and \( \lambda \) are labeling functions which associate each vertex \( p \in N \) with two sets \( \chi(p) \subseteq \text{nodes}(H) \) and \( \lambda(p) \subseteq \text{edges}(H) \). A hypertree decomposition is an acyclic hypertree for \( H \).
nodes(ℋ) and \( \lambda(p) \subseteq \text{edges}(ℋ) \). If \( T' = (N', E') \) is a subtree of \( T \), we define \( \chi(T') = \bigcup_{v \in N'} \chi(v) \). In the following, for any rooted tree \( T \), we denote the set of vertices \( N \) of \( T \) by \( \text{vertices}(T) \), and the root of \( T \) by \( \text{root}(T) \). Moreover, for any \( p \in N \), \( T_p \) denotes the subtree of \( T \) rooted at \( p \).

A \textit{generalized hypertree decomposition} \( [24] \) of a hypergraph \( ℋ \) is a hypertree \( HD = \langle T, \chi, \lambda \rangle \) for \( ℋ \) such that: (1) for each hyperedge \( h \in \text{edges}(ℋ) \), there exists \( p \in \text{vertices}(T) \) such that \( h \subseteq \chi(p) \); (2) for each node \( Y \in \text{nodes}(ℋ) \), the set \( \{ p \in \text{vertices}(T) \mid Y \in \chi(p) \} \) induces a (connected) subtree of \( T \); and (3) for each \( p \in \text{vertices}(T) \), \( \chi(p) \subseteq \text{nodes}(\lambda(p)) \). The \textit{width} of a generalized hypertree decomposition \( \langle T, \chi, \lambda \rangle \) is \( \max_{p \in \text{vertices}(T)} |\lambda(p)| \). The \textit{generalized hypertree width \( ghw(ℋ) \)} of \( ℋ \) is the minimum width over all its generalized hypertree decompositions.

A \textit{hypertree decomposition} \( [24] \) of \( ℋ \) is a generalized hypertree decomposition \( HD = \langle T, \chi, \lambda \rangle \) where: (4) for each \( p \in \text{vertices}(T) \), \( \text{nodes}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p) \). Note that the inclusion in the above condition is actually an equality, because Condition (3) implies the reverse inclusion. The \textit{hypertree width \( hw(ℋ) \)} of \( ℋ \) is the minimum width over all its hypertree decompositions. Note that, for any hypergraph \( ℋ \), it is the case that \( ghw(ℋ) \leq hw(ℋ) \leq 3 \times ghw(ℋ) + 1 [3] \). Moreover, for any fixed natural number \( k > 0 \), deciding whether \( hw(ℋ) \leq k \) is feasible in polynomial time (and, actually, it is highly-parallelizable) \([24]\), while deciding whether \( ghw(ℋ) \leq k \) is NP-complete \([26]\).

**Tree Projections.** For two hypergraphs \( ℋ_1 \) and \( ℋ_2 \), we write \( ℋ_1 \leq ℋ_2 \) if, and only if, each hyperedge of \( ℋ_1 \) is contained in at least one hyperedge of \( ℋ_2 \). Let \( ℋ_1 \leq ℋ_2 \): then, a \textit{tree projection} of \( ℋ_1 \) with respect to \( ℋ_2 \) is an acyclic hypergraph \( ℋ_a \) such that \( ℋ_1 \leq ℋ_a \leq ℋ_2 \). Whenever such a hypergraph \( ℋ_a \) exists, we say that the pair of hypergraphs \( (ℋ_1, ℋ_2) \) has a tree projection.

Note that the notion of tree projection is more general than the above mentioned (hyper)graph based notions. For instance, consider the generalized hypertree decomposition approach. Given a hypergraph \( ℋ \) and a natural number \( k > 0 \), let \( ℋ^k \) denote the hypergraph over the same set of nodes as \( ℋ \), and whose set of hyperedges is given by all possible unions of \( k \) edges in \( ℋ \), i.e., \( \text{edges}(ℋ^k) = \{ h_1 \cup h_2 \cup \cdots \cup h_k \mid \{ h_1, h_2, \ldots, h_k \} \subseteq \text{edges}(ℋ) \} \). Then, it is well known and easy to see that \( ℋ \) has generalized hypertree width at most \( k \) if, and only if, there is a tree projection for \( (ℋ, ℋ^k) \).

Similarly, for tree decompositions, let \( ℋ^{tk} \) be the hypergraph over the same set of nodes as \( ℋ \), and whose set of hyperedges is given by all possible clusters \( B \subseteq \text{nodes}(ℋ) \) of nodes such that \( |B| \leq k + 1 \). Then, \( ℋ \) has treewidth at most \( k \) if, and only if, there is a tree projection for \( (ℋ, ℋ^{tk}) \).

**Relational Structures and Homomorphisms.** Let \( 𝑈 \) and \( 𝑋 \) be disjoint infinite sets that we call the \textit{universe of constants} and the \textit{universe of variables}, respectively. A (relational) vocabulary \( τ \) is a finite set of relation symbols of specified (finite) arities. A \textit{relational structure} \( ℬ \) over \( τ \) (short: \( τ \)-structure) consists of a universe \( A \subseteq 𝑈 \cup 𝑋 \) and, for each relation symbol \( r \) in \( τ \), of a relation \( r^A \subseteq A^{|r|} \), where \( |r| \) is the arity of \( r \).

Let \( 𝑈 \) and \( 𝑋 \) be two \( τ \)-structures with universes \( A \) and \( B \), respectively. A
homomorphism from $A$ to $B$ is a mapping $h : A \rightarrow B$ such that $h(c) = c$ for each constant $c$ in $A \cap U$, and such that, for each relation symbol $r$ in $\tau$ and for each tuple $(a_1, \ldots, a_\rho) \in r^A$, it holds that $(h(a_1), \ldots, h(a_\rho)) \in r^B$. For any mapping $h$ (not necessarily a homomorphism), $h((a_1, \ldots, a_\rho))$ is used, as usual, as a shorthand for $(h(a_1), \ldots, h(a_\rho))$.

A $\tau$-structure $A$ is a substructure of a $\tau$-structure $B$ if $A \subseteq B$ and $r^A \subseteq r^B$, for each relation symbol $r$ in $\tau$.

Relational Databases. Let $\tau$ be a given vocabulary. A database instance (or, simply, a database) $DB$ over $D \subseteq U$ is a $\tau$-structure $DB$ whose universe is the set $D$ of constants. For each relation symbol $r$ in $\tau$, $r^{DB}$ is a relation instance (or, simply, relation) of $DB$. Sometimes, we adopt the logical representation of a database [51, 1], where a tuple $(a_1, \ldots, a_\rho)$ of values from $D$ belonging to the $\rho$-ary relation (over symbol) $r$ is identified with the ground atom $r(a_1, \ldots, a_\rho)$. Accordingly, a database $DB$ can be viewed as a set of ground atoms. Unless otherwise stated, we implicitly assume that databases are finite.

Conjunctive Queries. A conjunctive query $Q$ consists of a finite conjunction of atoms of the form $r_1(u_1) \land \cdots \land r_m(u_m)$, where $r_1, \ldots, r_m$ (with $m > 0$) are relation symbols (not necessarily distinct), and $u_1, \ldots, u_m$ are lists of terms (i.e., variables or constants). The set of all atoms occurring in $Q$ is denoted by $atoms(Q)$. For a set of atoms $A$, $vars(A)$ is the set of variables occurring in the atoms in $A$. For short, $vars(Q)$ denotes $vars(atoms(Q))$. We say that $Q$ is a simple query if every atom is over a distinct relation symbol. Given a database $DB$ over $D$, $Q^{DB}$ denotes the set of all answers of $Q$ on $DB$, that is, all substitutions $\theta : vars(Q) \rightarrow D$ such that for each $1 \leq i \leq m$, $\theta'(r_{\alpha_i}(u_i)) \in DB$, where $\theta'(t) = \theta(t)$ if $t \in vars(Q)$ and $\theta'(t) = t$ otherwise (i.e., if the term $t$ is a constant).

Note that any conjunctive query $Q$ can be viewed as a relational structure $Q$, whose vocabulary $\tau_Q$ and universe $U_Q$ are the set of relation symbols and the set of terms occurring in its atoms, respectively. For each symbol $r_i \in \tau_Q$, the relation $r_i^Q$ contains a tuple of terms $u_i$ for any atom of the form $r_i(u_i) \in atoms(Q)$ defined over $r_i$. In the special case of simple queries, every relation $r_i^Q$ of $Q$ contains just one tuple of terms. According to this view, elements in $Q^{DB}$ are in a one-to-one correspondence with homomorphisms from $Q$ to $DB_Q$, where the latter is the (maximal) substructure of $DB$ over the (sub)vocabulary $\tau_Q$. Hereafter, we adopt this view but, for the sake of presentation, we identify queries and databases with their relational structures, i.e., we use directly $Q$ and $DB$ in place of $Q$ and $DB_Q$.

For any given set $S$ of variables, we denote by $Q^{DB}[S]$ the restriction of the (substitutions/)homomorphisms in $Q^{DB}$ over the variables in $S$. For the extreme case where $S = \emptyset$, define $h_{true}$ to be the restriction of any homomorphism over the empty set. Then, $Q^{DB}[\emptyset] = \{ h_{true} \}$ if $Q^{DB} \neq \emptyset$, and $Q^{DB}[\emptyset] = \emptyset$ if $Q^{DB} = \emptyset$. If $a$ is an atom, then $Q^{DB}[a]$ denotes $Q^{DB}[vars(a)]$.

Note that any atom $a$ can be viewed as a one-atom query, so that $a^{DB}$ is the set of all the homomorphisms from $a$ to $DB$, restricted to $vars(a)$ (i.e., projecting out possible constants occurring in $a$). For a set $A$ of atoms, we denote by $A^{DB}$
the set \( \{ a^{\text{DB}} \mid a \in A \} \).

A core of \( Q \) is a query \( Q' \) such that: (1) \( \text{atoms}(Q') \subseteq \text{atoms}(Q) \); (2) there is a homomorphism from \( Q \) to \( Q' \); and (3) there is no query \( Q'' \) satisfying (1) and (2) such that \( \text{atoms}(Q'') \subseteq \text{atoms}(Q') \). Equivalently, in terms of relational structures, \( Q' \) is a minimal substructure of \( Q \) such that (2) holds. The set of all the cores of \( Q \) is denoted by \( \text{cores}(Q) \). Elements in \( \text{cores}(Q) \) are isomorphic.

Hypergraphs and atoms. There is a very natural way to associate a hypergraph \( H = (N, H) \) with any set \( V \) of atoms: the set \( N \) of nodes consists of all variables occurring in \( V \); for each atom in \( V \), the set \( H \) of hyperedges contains a hyperedge including all its variables; and no other hyperedge is in \( H \).

For a query \( Q \), the hypergraph associated with \( \text{atoms}(Q) \) is briefly denoted by \( H_Q \). If \( H_Q \) is a connected hypergraph, we say that \( Q \) is a connected query.

3 The Power of Local Consistency

Throughout the paper, we assume that \( Q \) is a conjunctive query and that \( V \) is a non-empty set of atoms, which we call views, such that \( \text{vars}(V) = \text{vars}(Q) \). Moreover, \( \text{DB} \) is a database over the vocabulary \( \text{DS} \) containing the relation symbols of query atoms and views. We require w.l.o.g. that every available view is over a specific relation symbol, which does not occur in the given query, and that the list of terms of every view does not contain any constant or repeated variables (in fact, observe that from any given set of available views, one may immediately get a new set of views where these assumptions hold). Note that, within this setting, each view \( w \in V \) is univocally associated with a relation instance in \( \text{DB} \), whose tuples are in a one-to-one correspondence with the homomorphisms in \( w^{\text{DB}} \). Therefore, this relation instance will be simply denoted by \( w^{\text{DB}} \), and we freely use the term tuples interchangeably with homomorphisms, when we refer to its elements.

Our first goal is to characterize the relationships between tree projections and certain consistency properties that hold for \( Q \) and \( V \) over some (or all) given databases. To this end, we need to state some preliminary notions and definitions, which will be illustrated by referring to the following running example.

Example 3.1 Consider the following query \( Q_4 \), where all atoms are over the same binary relation symbol \( r \):

\[
Q_4 : \quad r(A, B) \land r(B, C) \land r(A, C) \land r(D, B) \land r(A, E) \land r(F, E).
\]

A graphical representation of this query is reported in Figure 2 where edge orientation just reflects the position of the variables in query atoms. Moreover, consider the database \( \text{DB}_4 \) shown in Figure 2 by focusing on the relation instance \( r^{\text{DB}_4} \). Then, it can be checked that the answers of \( Q_4 \) on \( \text{DB}_4 \) are the homomorphisms \( h_1, ..., h_{10} \), which are also reported, in tabular form, in Figure 2.

In this example, in order to answers \( Q_4 \), we assume the availability of the set of views \( V_4 = \{ v_1(A, B, C), v_2(A, F), v_3(A, B), v_4(A, C), v_5(A, E), v_6(B, C), \}

\]
$v_7(D, B), v_8(D, C), v_9(F, E)$}, and that the database DB4 includes a relation instance $w^{DB}$, for each view $w \in \mathcal{V}_4$. Note that, in the figure, such relation instances are identified by the list of variables on which the views are defined.

\[ \{ w^{DB} \mid w \in \text{views}(Q_4) \} \]

Figure 2: The (hypergraph of the) query $Q_4$, the tuples in the database DB4, and the answers in $Q_4^{DB}$, in Example 3.1.

### 3.1 Consistency Properties and Views

**View Consistency.** For a view $w \in \mathcal{V}$, we say that $w^{DB}$ is **view consistent** w.r.t. $Q$ if $w^{DB} \supseteq Q^{DB}[w]$. For the set of views $\mathcal{V}$, we say that $\mathcal{V}^{DB}$ is **view consistent** w.r.t. $Q$, if the property holds for each $w \in \mathcal{V}$. That is, views are not more restrictive than the query.

Note that view consistency holds in general for all views initialized from subsets of query atoms, such as those employed in all known decomposition methods, such as (hyper)tree decompositions. However, we are also interested in a wider framework where views are completely arbitrary and may be available from previous computations, possibly unrelated with the present query $Q$. Accordingly, we do not require that view consistency holds for such views, and we shall look for general results, which will be then smoothly inherited by more specific settings.
Example 3.2 Consider again the setting of Example 5.1 and in particular the views \( v_1(A, B, C) \) and \( v_2(A, B, C) \). Note that \( v_1(A, B, C)^{DB_4} \) is a set of two homomorphisms, which are precisely those in the set \( Q_4^{DB_4}[\{A, B, C\}] \) of the answers of \( Q_4 \) on \( DB_4 \) projected over the variables in \( \{A, B, C\} \). Therefore, 
\[ v_1(A, B, C) \] is view consistent w.r.t. \( DB_4 \). Similarly, it can be checked that the views \( v_3(A, B) \), \( v_4(A, C) \), \( v_5(A, E) \), \( v_6(B, C) \), \( v_7(D, B) \), \( v_8(D, C) \), and \( v_9(F, E) \) are all view consistent w.r.t. \( DB_4 \).

Instead, \( v_2(A, F) \) is not view consistent w.r.t. \( DB_4 \), since \( v_2(A, F)^{DB_4} \supseteq Q_4^{DB_4}[\{A, F\}] \) does not hold. For instance, \( v_2(A, F)^{DB_4} \) does not include the homomorphism mapping both \( A \) and \( F \) to the constant \( a_1 \). Hence, \( V_4^{DB_4} \) is not view consistent w.r.t. \( Q_4 \).

Local Consistency. We say that \( V^{DB} \) is locally (also, pairwise) consistent, denoted by \( lc(V, DB) \), if \( w^{DB} \neq \emptyset \) and \( w^{DB} = (w \wedge w')^{DB}[w] \), for each \( w, w' \subseteq V \).

From any set of views and any instance \( DB \), we may compute a subset of \( DB \) that is locally consistent. Let the reduct of \( DB \) according to \( V \), denoted by \( red(V, DB) \), be the (set-inclusion) maximal subset of \( DB \) such that \( V^{red(DB, V)} \) is locally consistent; or \( red(V, DB) = \emptyset \), whenever such a maximal subset does not exist. It is well known that the reduct can be computed as the unique fixpoint of a procedure consisting of semijoin operations over \( DB \), which runs in polynomial time. It is easy to see that such a reducing procedure preserves the given query, unless the used views are more restrictive than the query, of course. In fact, computing a reduct is often used as a useful heuristic procedure in different areas of computer science, where the homomorphism problem underlying conjunctive query evaluation comes out—e.g., in constraint satisfaction problems (CSP), where such a procedure is known as generalized arc consistency [16]. Indeed, if the reduct is empty, we may safely conclude that there are no solutions; otherwise, we got anyway a smaller instance of the problem to deal with.

Example 3.3 In the running example depicted in Figure 2, the set \( V_4 \) of views and the database \( DB_4 \) are such that \( V_4^{DB_4} \) is locally consistent. Consider for instance the views \( v_1(A, B, C) \) and \( v_3(A, B) \), and observe that both \( (v_1(A, B, C) \wedge v_3(A, B))^{DB_4}[\{A, B, C\}] = v_1(A, B, C)^{DB_4} \) and \( (v_3(A, B) \wedge v_1(A, B, C))^{DB_4}[\{A, B\}] = v_3(A, B)^{DB_4} \). Indeed, every tuple in the relation associated with either view matches with some tuple in the other view on the variables they have in common, so that no tuple is missed by performing such semijoin operations. This is easily seen because \( v_1(A, B, C)^{DB_4}[\{A, B\}] = v_3(A, B)^{DB_4} = \{a_1, b_1, a_2, b_2\} \) (where these two tuples also identify the homomorphisms mapping \( A \) to \( (a_1, b_1) \) and to \( (a_2, b_2) \), respectively).

Query Views. In the seminal paper about local and global consistency in acyclic queries [7], local consistency is enforced directly on the relations of query atoms, while we only consider (and possibly enforce) this property on views, in this paper. This is because that paper, as well as other related papers such as [29], uses a slightly different formal framework where every relation symbol may occur just once in a query, i.e., where only simple queries are considered.
In contrast with these classical papers, we do not assume anything about the query, which may contain multiple occurrences of the same relation symbol. This means that the same relation instance may be shared by different query atoms, and this feature plays a very relevant role, as it was first pointed out in [15]. In this case, a tuple may be useful for some atom and useless for another one defined over the same relation symbol. It follows that local consistency cannot be enforced on the relations of the query atoms, because such a filtering procedure would lead to undesirable side effects (possibly deleting all tuples in the database, including the useful ones).

Therefore, we always keep the “original” database relations untouched and we rather use suitable views, each one with its own database relation, to play the role of query atoms in the definition of consistency properties in general queries and in consistency enforcing procedures. Formally, we say that \( V \) is a view system (for \( Q \)) if it contains, for each atom \( q \in \text{atoms}(Q) \), a view \( w_q \) (over a distinct relation symbol) with the same set of variables as \( q \). These special views in \( V \) are called hereafter query views, and are denoted by \( \text{views}(Q) \). If \( Q' \) is a subquery of \( Q \), \( \text{views}(Q') \) denotes the set of query views associated with its atoms. In the following, the set of available views \( V \) is assumed to be a view system for the given query \( Q \), unless otherwise specified.

**Example 3.4** Consider again the setting of Example 3.1 and note that \( V_4 \) is in fact a view system for \( Q_4 \). Indeed, the views in the set \( \{v_3(A, B), v_4(A, C), v_5(A, E), v_6(B, C), v_7(D, B), v_8(D, C), v_9(F, E)\} \) are in a one-to-one correspondence with the query atoms of \( Q_4 \). For instance, \( v_3(A, C) \) is the query view \( w_{r(A,C)} \), with \( r(A, C) \) being a query atom of \( Q_4 \). Hence, \( \text{views}(Q_4) = \{v_3(A, B), v_4(A, C), v_5(A, E), v_6(B, C), v_7(D, B), v_8(D, C), v_9(F, E)\} \), and \( V_4 = \text{views}(Q_4) \cup \{v_1(A, B, C), v_2(A, F)\} \).

Observe that working with view systems instead that with arbitrary set of views is not a restrictive assumption, for our purposes. On the practical side, if some atom misses its associated query view \( w_q \) in the available views, one may just add a fresh view \( w_q \) to the views, with a corresponding relation in the database such that \( w_q^{\text{DB}} = q^{\text{DB}} \). On the theoretical side, recall that we are dealing with consistency properties of \( Q \) and \( V \), and with tree projections of \((H_Q, H_V)\). In fact, such a tree projection exists only if the set of variables of every atom \( q \) in \( Q \) is covered by some view \( w \in V \), i.e., \( \text{vars}(q) \subseteq \text{vars}(w) \). Therefore, whenever \( V \) is a set of “useful views,” for each query atom \( q \) there must exist some view in \( V \) that may play the role of the query view \( w_q \) (after projecting it on \( \text{vars}(q) \)). However, requiring that query views belong to \( V \) simplifies the presentation and allows us to define consistency properties in a clean way. In particular, the role of query views is crucial in the following definition.

**Global Consistency.** Informally, this is a highly desirable state of the database where query views contain all and only those tuples that can be returned by query answers. In this case, an answer of the query can be computed in polynomial time: for each query view \( w_q \), select one tuple \( h \) in the relation \( w_q^{\text{DB}} \) that is univocally associated with \( w_q \) in \( \text{DB} \), modify this relation so that \( w_q^{\text{DB}} = \{h\} \),

18
and propagate this choice by enforcing again local consistency (see Section 4 for more results and discussions about the problem of computing answers).

Observe that the classical definition, which states the above property for the relations of query atoms, is not useful whenever any relation symbol \( r \) is shared by some query atoms (because we miss the information relating any tuple in \( r^{DB} \) with those atoms where the tuple participates in some answer). By using query views instead of query atoms, no confusion may arise, and we get the desired extension of the classical definition given (in the literature discussed above) for simple queries.

We say that a database \( DB \) is \textit{globally consistent} with respect to \( Q \) and \( \mathcal{V} \), denoted by \( gc(V, DB, Q) \), if \( w^q_{DB} = Q^{DB}[q] \) (which is also equal to \( Q^{DB}[w^q] \)), for each \( q \in \text{atoms}(Q) \), where \( w^q \) is the query view associated with \( q \).

\textbf{Example 3.5} Let us focus on the query views in \( \text{views}(Q_4) \). Consider for instance the view \( v_3(A, B) \in \text{views}(Q_4) \) (associated with the query atom \( r(A, B) \)), and note that \( v_3(A, B)^{DB_4} = Q_4^{DB_4}[\{A, B\}] \). That is, the answers of \( Q_4 \) on \( DB_4 \) projected over the set \( \{A, B\} \) are immediately available by looking at the relation \( v_3(A, B)^{DB_4} \).

On the other hand, for the view \( v_8(D, C) \in \text{views}(Q_4) \), the set \( v_8(D, B)^{DB_4} \) contains two homomorphisms that do not belong to the set \( Q_4^{DB_4}[\{D, C\}] \) (identified by the two tuples marked with the symbol “◮” in Figure 2). Therefore, \( DB_4 \) is not globally consistent w.r.t. \( Q_4 \) and \( \mathcal{V}_4 \).

\textbf{Legal Database.} While no special requirement is assumed for the database relations of the available views in \( \mathcal{V} \), the relations associated with the query views cannot be arbitrary, otherwise we would lose any connection with the query \( Q \) to be solved using the view system \( \mathcal{V} \). In fact, these relations should reflect the intended initialization with the tuples contained in the relations associated with their corresponding query atoms (possibly filtered by eliminating tuples that are irrelevant w.r.t. query answers).

We say that \( DB \) is a \textit{legal} database instance (w.r.t. \( Q \) and \( \mathcal{V} \)) if (i) \( w^q_{DB} \subseteq q^{DB} \) holds, for each query view \( w^q \in \text{views}(Q) \); and (ii) \( \text{views}(Q)^{DB} \) is \textit{view consistent}. All other view instances may be arbitrary. Then, the following is immediate.

\textbf{Fact 3.6} For every legal database \( DB \),

\[
Q^{DB} = \bigwedge_{q \in \text{atoms}(Q)} q^{DB} = \bigwedge_{w^q \in \text{views}(Q)} w^q_{DB}.
\]

\textbf{Example 3.7} The database \( DB_4 \) is legal w.r.t. \( Q_4 \) and \( \mathcal{V}_4 \). Indeed, condition (i) is seen to hold by comparing the relations associated with the query views with the relation instance \( r^{DB_4} \). Moreover, in Example 3.2, we have observed that \( \text{views}(Q_4)^{DB_4} \) is view consistent, i.e., condition (ii) holds as well. Then, because of the above fact, the answers of \( Q_4 \) on \( DB_4 \) are also given by the expression \( \bigwedge_{w^q \in \text{views}(Q_4)} w^q_{DB_4} \).
Remark 3.8 Only legal databases over $Q$ and $V$ are meaningful for the purpose of this paper. Therefore, unless otherwise stated, we always implicitly assume hereafter this requirement for any database instance. In particular, whenever we say “for every database”, we actually mean “for every legal database”. Of course, whenever we define some database instance in proofs of our results, we deal with this requirement, and we explicitly prove that such a database is actually legal.

Now that the setting is clarified, our next task is to provide sufficient and necessary conditions to evaluate queries via local consistency. For the sake of presentation and without loss of generality, we assume that the given query $Q$ is connected and that $\text{vars}(Q) = \text{vars}(V)$. Note that, under these assumptions, whenever $V^{DB}$ is locally consistent, requiring that every relation associated with some view in $V$ is non-empty is equivalent to requiring that there is at least one $w$ in $V$ with $w^{DB} \neq \emptyset$. Indeed, the query views in the view system $V$ makes $H_V$ connected, and thus any empty relation in the database would entail that all relations must be empty, at local consistency.

3.2 From Tree Projections to Consistency...

The fact that local consistency holds for $V$ and $DB$ is of course unrelated with the fact that global consistency holds for $V$ and $DB$ with respect to $Q$, in general. In this section, we show how the existence of tree projections of some parts of the query is a sufficient condition to get the implication $\text{lc}(V, DB) \Rightarrow \text{gc}(V, DB, Q)$. Our analysis will consider arbitrary conjunctive queries, with any desired set $O$ of output variables, and tree projections w.r.t. arbitrary view systems.

We start by observing that, when arbitrary view systems are considered, it suddenly emerges that it does not make sense to talk about “the” core of a query, because different isomorphic cores may differently behave with respect to the available views. In fact, this phenomenon does not occur, e.g., for generalized hypertree decompositions (resp., tree decompositions) where all combinations of $k$ atoms (resp., $k + 1$ variables) are available as views (see Section 4).

Example 3.9 Consider again the query

$$Q_4 : r(A, B) \land r(B, C) \land r(A, C) \land r(D, C) \land r(D, B) \land r(A, E) \land r(F, E),$$

which has been discussed in Example 3.1, and which is graphically reported again in Figure 3, for the sake of presentation. The figure also reports the hypergraph $H_{V_4}$ associated with the views in $V_4$ (where, e.g., the hyperedges $\{A, B, C\}$ and $\{A, F\}$ are those corresponding to the views $v_1(A, B, C)$ and $v_2(A, F)$, and where (hyper)edges associated with the query views are still depicted with their original orientation in $Q_4$, as to make the correspondence clearer). Moreover, the figure reports the two queries

$$Q_5 : r(A, B) \land r(B, C) \land r(A, C)$$
$$Q_6 : r(D, B) \land r(B, C) \land r(D, C).$$
Figure 3: The (hypergraph of the) query $Q_4$, the cores $Q_5$ and $Q_6$, the hypergraph $H_{V_4}$, and the cores of the queries $Q_4 \land \text{atom} \{F, E\}$ (with its tree projection) and $Q_4 \land \text{atom} \{A, F\}$, in Example 3.9.

Note that $Q_5$ and $Q_6$ are two (isomorphic) cores of $Q_4$, but they have different structural properties. Indeed, $(H_{Q_5}, H_{V_4})$ admits a tree projection (note in the figure that the view over $\{A, B, C\}$ “absorbs” the cycle), while $(H_{Q_6}, H_{V_4})$ does not.

\begin{center} $\triangleright$ \end{center}

**Computation Problem.** Armed with the observation exemplified above, the relationship between consistency and structural properties will be next stated by considering the existence of a tree projection for *some* core of the query $Q$.

In addition, to properly deal with arbitrary sets of output variables (which may be not included in any core of $Q$), we need to define an “output-aware” notion of covering by tree projections, where cores are forced to contain the desired output variables.

**Definition 3.10** For any set of variables $O$ occurring in some atom $w \in V$, define $\text{atom}(O)$ to be a fresh atom (with a fresh relation symbol) over these variables, i.e., such that $O = \text{vars}(\text{atom}(O))$. Then, we say that $O$ is tp-covered in $Q$ (w.r.t. $V$) if there exists some core $Q'$ of $Q \land \text{atom}(O)$ such that $(H_{Q'}, H_{V'})$ has a tree projection.
A first easy observation is that the \textit{tp-covered} property holds for every set of variables occurring in every query atom, whenever \((\mathcal{H}_Q, \mathcal{H}_V)\) has a tree projection.

\textbf{Fact 3.11} Assume that \((\mathcal{H}_Q, \mathcal{H}_V)\) has a tree projection. Then, for every \(q \in \text{atoms}(Q)\) and every \(O \subseteq \text{vars}(q)\), \(O\) is \textit{tp-covered} in \(Q\) (w.r.t. \(V\)).

\textbf{Proof.} Let \(q\) be any atom occurring in \(Q\) and take any \(O \subseteq \text{vars}(q)\). Let \(Q'\) be any core of \(Q \land \text{atom}(O)\). Since \(Q'\) is a subquery of \(Q \land \text{atom}(O)\) and \(O \subseteq \text{vars}(q)\), \(\mathcal{H}_{Q'} \leq \mathcal{H}_Q\). Thus, \(\mathcal{H}_{Q'} \leq \mathcal{H}_Q \leq \mathcal{H}_a \leq \mathcal{H}_V\), where \(\mathcal{H}_a\) is any tree projection of \((\mathcal{H}_Q, \mathcal{H}_V)\), which exists by hypothesis. \(\Box\)

We next show that the above fact may be extended to those atoms occurring in some core of \(Q\) having a tree projection.

\textbf{Lemma 3.12} Let \(q \in \text{atoms}(Q')\) be an atom occurring in some core \(Q'\) of \(Q\) for which \((\mathcal{H}_{Q'}, \mathcal{H}_V)\) has a tree projection. Then, \(\forall O \subseteq \text{vars}(q)\), \(O\) is \textit{tp-covered} in \(Q\) (w.r.t. \(V\)).

\textbf{Proof.} Let \(O \subseteq \text{vars}(q)\), and consider the query \(Q \land \text{atom}(O)\). We first claim that there is a homomorphism from \(Q \land \text{atom}(O)\) to \(Q' \land \text{atom}(O)\). Indeed, since \(Q' \in \text{cores}(Q)\), it is also a retract of \(Q\) (see, e.g., [27]); that is, there is a homomorphism \(f\) from \(Q\) to \(Q'\) which is the identity on its range (i.e., \(f(X) = X\), for every term \(X\) occurring in \(Q'\)). Moreover, \(O \subseteq \text{vars}(Q')\), because \(q \in \text{atoms}(Q')\). It follows that \(f\) is also a homomorphism from \(Q \land \text{atom}(O)\) to \(Q' \land \text{atom}(O)\). In particular, note that \(f\) maps the atom \(\text{atom}(O)\) to itself. We thus conclude that \(Q' \land \text{atom}(O)\) is also a core of \(Q \land \text{atom}(O)\), because \(\text{atom}(O)\) is over a fresh relation symbol and hence must belong to any core, and dropping atoms from \(Q'\) would contradict the minimality of \(Q'\) as a core of \(Q\). Finally, since \(\text{vars}(\text{atom}(O)) = O \subseteq \text{vars}(q)\) and \(q \in \text{atoms}(Q')\), the hypergraph associated with \(Q' \land \text{atom}(O)\), say \(H'\), is such that \(H' \leq \mathcal{H}_{Q'}\). Hence, any tree projection of \(\mathcal{H}_{Q'}\) w.r.t. \(\mathcal{H}_V\), which exists by hypothesis, is a tree projection of \(H'\) w.r.t. \(\mathcal{H}_V\). That is, \(O\) is \textit{tp-covered} in \(Q\) (w.r.t. \(V\)). \(\Box\)

\textbf{Example 3.13} Consider again the setting of Example 3.9. The core \(Q_5\) contains the atoms \(r(A,B), r(B,C),\) and \(r(A,C)\), and we have noticed that \(Q_5\) admits a tree projection. Therefore, we can apply Lemma 3.12 to conclude that the sets of variables \(\{A,B\}, \{B,C\},\) and \(\{A,C\}\) are \textit{tp-covered} in \(Q_4\).

Consider now the set of variables \(\{F,E\}\), which does not occur in any core of the query, and the novel query \(Q_4 \land \text{atom}(\{F,E\})\). This query has a unique core, which is again depicted in Figure 3. Notice that this core does not coincide with any of the two cores of the original query. Yet, it admits a tree projection, consisting of the hyperedges \(\{F,E\}, \{A,E\},\) and \(\{A,B,C\}\), as shown in the figure. Thus, \(\{F,E\}\) is \textit{tp-covered} in \(Q_4\).

On the other hand, the hypergraphs associated with the cores of \(Q_4 \land \text{atom}(\{D,C\})\) and \(Q_4 \land \text{atom}(\{D,B\})\) are precisely the same as the hypergraph \(\mathcal{H}_{Q_6}\) associated with the core \(Q_6\), that is, the triangle with vertices \(D,B,\)
and $C$, having no tree-projection w.r.t. $\mathcal{H}_V$. Hence, $\{D, C\}$ and $\{D, B\}$ are not tp-covered in $Q_4$.

Finally, for an example application of Definition 3.10 with arbitrary set of variables (i.e., not just contained in query atoms), consider the set $\{A, F\}$. Consider then the query $Q_4 \land \text{atom}(\{A, F\})$ and note that its core does not have a tree projection. Thus, $\{A, F\}$ is not tp-covered in $Q_4$. 

The notion of tp-covering plays a crucial role in establishing consistency properties. To help the intuition, this role is next exemplified.

Example 3.14 Consider again the setting of Example 3.1 (and Example 3.9) and the database $\text{DB}_4$ shown in Figure 2 over the relation symbol $r$ (in $Q_4$) and the symbols for the views in $V_4 = \text{views}(Q_4) \cup \{v_1(A, B, C), v_2(A, F)\}$. Recall from Example 3.3 that $V_4^{\text{Dbase}}$ is locally consistent.

Observe that for the query view $v_4(A, C)$, $v_4(A, C)^{\text{Dbase}}$ consists of the two tuples/homomorphisms $\langle a_1, c_1 \rangle$ and $\langle a_2, c_2 \rangle$. That is, this query view provides exactly the two homomorphisms in $Q_4^{\text{Dbase}}[\{A, C\}]$, i.e., the answers of $Q_4$ projected over the variables ($A$ and $C$) of the view $w_r(A, C)$. Note that the same property holds for the views over the set of variables $\{A, C\}$, $\{A, B\}$, $\{B, C\}$, $\{F, E\}$, $\{A, E\}$, and $\{A, B, C\}$. Interestingly, each one of this set is tp-covered in $Q_4$ (see also Example 3.13).

On the other hand, each one of the sets $v_7(D, B)^{\text{Dbase}}$, $v_8(D, C)^{\text{Dbase}}$, and $v_2(A, F)^{\text{Dbase}}$ contains two homomorphisms that do not correspond to any answer of the query (suitably projected over the variables of interest), which are those identified by the tuples marked with the symbol "$\triangleright$" in Figure 2. In fact, we observe that, in this case, $\{D, C\}$, $\{D, B\}$, and $\{A, F\}$ are not tp-covered in $Q_4$.

In the above example, the fact that homomorphisms that are not correct answers are associated with views whose variables are not tp-covered is not by chance. Indeed, the intuition is now that to guarantee global consistency by just enforcing local consistency, all the variables contained in query atoms must be tp-covered.

Next, we establish a lemma that actually proves a slightly more general result dealing with any set of output variables covered by some view. For a set of variables $O$, let $\text{covers}(O)$ denote the set of all views $w \in V$ such that $O \subseteq \text{vars}(w)$.

Lemma 3.15 Assume that $V^{\text{Dbase}}$ is locally consistent. For any set of variables $O$ that is tp-covered in $Q$, $w^{\text{Dbase}}[O] \subseteq Q^{\text{Dbase}}[O]$ holds, for every $w \in \text{covers}(O)$. Moreover, if $w^{\text{Dbase}}$ is view consistent w.r.t. $Q$, for some $\bar{w} \in \text{covers}(O)$, then we actually get all the right homomorphisms for all of them, i.e., $w^{\text{Dbase}}[O] = Q^{\text{Dbase}}[O]$ holds, for every $w \in \text{covers}(O)$.

Proof. Let $Q_c = Q \land \text{atom}(O)$. Assume that $O$ is tp-covered in $Q$, that is, there exists $Q' \in \text{cores}(Q_c)$ for which $(\mathcal{H}_{Q'}, \mathcal{H}_V)$ has a tree projection. Since $Q'$ is a core, it is also a retract of $Q_c$; that is, there is a homomorphism $f$ from
Let us associate with \( Q \) the following query:

\[
Q_a = WQ' \wedge \bigwedge_{h \in \text{edges}(H_a)} \text{atom}(h).
\]

For any fresh atom \( \text{atom}(h) \in \text{atoms}(Q_a) \) (including \( \text{atom}(O) \)), let \( \text{atom}(h)^{DB} = v^{DB}[h] \), where \( v \in V \) is any view satisfying \( h \subseteq \text{vars}(v) \), chosen according to some fixed (arbitrary) criterium. Such a view always exists because \( H_a \) is a tree projection of \( (H_Q, H_V) \).

Note that \( Q_a^{DB} \subseteq WQ^{DB} \), because \( WQ' \) is a subquery of \( Q_a \). By construction \( Q_a \) is a simple acyclic query, and \( \text{atoms}(Q_a)^{DB} \) is locally consistent because all these relations are projections of views in the locally consistent set \( \nu^{DB} \). Thus, by the results in \([4]\), \( Q_a^{DB} \) is globally consistent and we get, for the atom \( \text{atom}(O) \),

\[
\text{atom}(O)^{DB} = Q_a^{DB}[O] \subseteq WQ^{DB}[O] \subseteq Q^{DB}[O].
\]

Moreover, since \( \nu^{DB} \) is locally consistent, this property must hold for every \( w^{DB} \), with \( w \in V \) and \( O \subseteq \text{vars}(w) \). That is, \( w^{DB}[O] \subseteq Q^{DB}[O] \), for every \( w \in \text{covers}(O) \).

Assume now that the output variables \( O \) are covered by some view consistent atom, i.e., \( O \subseteq \text{vars}(\bar{w}) \) for some \( \bar{w} \in V \) such that \( Q^{DB}[\text{vars}(\bar{w})] \subseteq \bar{w}^{DB} \) and thus \( Q^{DB}[O] \subseteq \bar{w}^{DB}[O] \). Since \( \nu^{DB} \) is locally consistent, it follows that \( \bar{w}^{DB}[O] = \text{atom}(O)^{DB} \) and thus \( Q^{DB}[O] \subseteq \text{atom}(O)^{DB} \). Combined with the above relationship, we get the desired equality \( Q^{DB}[O] = \text{atom}(O)^{DB} \). Again, since \( \nu^{DB} \) is locally consistent, this property must hold for every \( w^{DB} \), with \( w \in V \) and \( O \subseteq \text{vars}(w) \). That is, \( Q^{DB}[O] = w^{DB}[O] \), for every \( w \in \text{covers}(O) \). \( \square \)

Since query views are always view consistent (over legal databases), we immediately get the following sufficient condition for the global consistency, which clearly also holds for restricted tree projections corresponding to decomposition methods.

**Theorem 3.16** Assume that, for every \( q \in \text{atoms}(Q) \), \( \text{vars}(q) \) is tp-covered in \( Q \) (w.r.t. \( V \)). Then, for every database \( DB \), \( \text{ic}(V, DB) \) entails \( \text{gc}(V, DB, Q) \).
Having a tree projection of the full query is therefore not necessary for getting global consistency through local consistency. For instance, an unsuspectedly easy class of queries consists of the grid queries of the form $GQ_n = \bigwedge_{X,Y \in E_n} (e(X,Y) \land e(Y,X))$, where $E_n$ is the edge set of an $n \times n$ grid. Indeed, while such grids are well known obstructions to the existence of tree decompositions, any of their edges is a core (and, thus, trivially acyclic)—see Figure 4. Therefore, even the smallest possible set of views $V = \text{views}(GQ_n)$ is sufficient to obtain global consistency by enforcing local consistency.

As we shall prove in Section 3.3, Theorem 3.16 defines the most general possible condition to guarantee global consistency, which is what we need to answer the query by exploiting local consistency if the output variables are included in some query atom.

**Decision Problem.** The situation is rather different if we just look for the most general sufficient conditions to solve the decision problem $Q^{DB} \neq \emptyset$. In this case, it is sufficient the existence of a tree projection of any structure for which there is an endomorphism of the query. Of course, any such a subquery $Q'$ is homomorphically equivalent to $Q$, denoted by $Q' \approx_{\text{hom}} Q$ in the following. In fact, the concept of $tp$-covering is immaterial here, given that we are not interested in output variables (i.e., $O = \emptyset$). Thus, as a special case of our analysis on the computation problem, we get the following result, which generalizes to tree projections (where cores may behave differently) a similar sufficient condition known for the special cases of tree decompositions [15], and generalized hypertree decompositions [12].

**Theorem 3.17** Assume there is a subquery $Q' \approx_{\text{hom}} Q$ for which $(H_{Q'}, H_V)$ has a tree projection. Then, for every database $DB$, $lc(V, DB)$ entails $Q^{DB} \neq \emptyset$.

**Proof.** Let $H_a$ be a tree projection of $(H_{Q'}, H_V)$, for some $Q' \approx_{\text{hom}} Q$. Then, it is also a tree projection of $(H_{Q''}, H_V)$, for any $Q'' \in \text{cores}(Q') \subseteq \text{cores}(Q)$, because $H_{Q''} \leq H_{Q'}$. From Lemma 3.12, for any (query atom) $q \in \text{atoms}(Q'')$, $\text{vars}(q)$ is $tp$-covered in $Q$ and thus, from Lemma 3.13, $Q^{DB}[\text{vars}(q)] = w^{DB}_q$. Then, whenever $lc(V, DB)$ holds, $w^{DB}_q \neq \emptyset$ and hence $Q^{DB} \neq \emptyset$. \hfill $\square$

Note that the above condition is more liberal than what we need for having the global consistency. In the next section we prove that it is in fact also a necessary condition as far as the decision problem is concerned.

Moreover, we point out that, from an application perspective, either results above may be useful only if we have some guarantee (or some efficient way to
check) that the required conditions are met. Otherwise, as it happens for the decision problems in the special cases of (generalized) (hyper)tree decompositions [12, 34], we are in a promise setting where, in general, we are not able to actually compute any full (and thus polynomial-time checkable) query answer (or disprove the “promise”). In particular, it has been observed in a slightly different setting by [32] (see, also, [45, 10]) that, rather surprisingly, the global consistency property (and hence having a full reducer) is not sufficient to actually compute a full query answer (unless P = NP). Intuitively this is due to the fact that, as soon as we fix some tuple in a relation in order to extend it to a full solution, we are changing the set of available query endomorphisms and thus we may lose the property of some variables to be tp-covered. As a consequence, subsequent propagations are not guaranteed to maintain the global consistency.

3.3 ...and Back to Tree Projections

The question of whether the cases in which local consistency implies global consistency precisely coincide with the cases in which there is a tree projection of the query with respect to a set of views was a long-standing open problem in the literature [29, 44]. We next answer this question, both in the setting considered in those papers (where all relation symbols in the query are distinct), with the answer being positive there, and in the unrestricted setting where the answer is instead negative. In fact, we precisely characterize the relationships between local and global consistency and tree projections in the general setting too, by showing that tree projections are still necessary, but not necessarily involving the query as a whole.

**Decision Problem.** We start with the problem of checking whether the given query is not empty. Theorem 3.19 below provides the counterpart of Theorem 3.17. The proof requires some preparation.

Let DB be a database over the vocabulary DS. For the following results, we assume that each relation symbol r ∈ DS of arity ρ is associated with a set of ρ (distinct) attributes that identify the ρ positions available in r. In this context, r is also called relational schema, and DS is called database schema. An inclusion dependency is an expression of the form r₁[S] ⊆ r₂[S], where r₁ and r₂ are two relational schemas in DS and S is a set of attributes that r₁ and r₂ have in common. A database DB over DS satisfies this inclusion dependency if, for each tuple t₁ ∈ r₁[S], there is a tuple t₂ ∈ r₂[S] with t₁[S] = t₂[S] (where [ ] is here the classical projection relational operator applied to a set of attributes). Moreover, if DB satisfies each inclusion dependency in a given set I, then we simply say that DB satisfies I.

Define $A(DS)$ as the set of canonical atoms associated with the schema DS, that is, the set containing, for each relation r of DS, the atom $r(u)$ having as its variables the attributes of r. A conjunctive query Q is said to be a canonical query for DS whenever it consists of atoms from $A(DS)$, i.e., $atoms(Q) \subseteq A(DS)$ holds.

We are now ready to state a fundamental lemma on union of conjunctive
queries, i.e., on queries of the form $Q = Q_1 \lor \cdots \lor Q_n$, where $Q_i$ is a conjunctive query $\forall i \in \{1, \ldots, n\}$. We are interested in unions of Boolean queries, so that $\overline{Q}^0 \neq \emptyset$ if (and only if) $Q^0 \neq \emptyset$ for some query $Q_i$ in the union $Q$.

The ingredients in the lemma are a recent result on the finite controllability of unions of conjunctive queries in the framework of databases under the open-world assumption [43], and a connection between tree projections and the chase procedure firstly observed in [44].

**Lemma 3.18** Let $DS$ be a database schema equipped with a set $I$ of inclusion dependencies. Let $Q$ be a union of canonical queries for $DS$ such that, for (finite) $DB \neq \emptyset$ over $DS$, $DB$ satisfies $I \Rightarrow \overline{Q}^n \neq \emptyset$. Then, there exists a conjunctive query $Q'$ in the union $Q$ such that $(H_{Q'}, H_{A(DS)})$ has a tree projection.

**Proof.** Unlike all other proofs in the paper, we next deal both with finite and infinite databases, and thus we always point out whether a database is (or may be) infinite. All databases are implicitly assumed to be over the database schema $DS$. From the hypothesis, the following property holds for $Q$:

$$P_1 \forall \text{finite } DB \neq \emptyset, DB \text{ satisfies } I \Rightarrow \overline{Q}^n \neq \emptyset.$$

Let us start by taking an arbitrary atom $r_w(X_1, \ldots, X_m)$ in $Q$, and let $DB_0 = \{ r_w(c_{X_1}, \ldots, c_{X_m}) \}$, where $c_{X_1}, \ldots, c_{X_m}$ are fresh (distinct) constants. Trivially, $P_1$ entails the following property:

$$P_2 \forall \text{finite } DB \supset DB_0, DB \text{ satisfies } I \Rightarrow \overline{Q}^n \neq \emptyset.$$

Recall that the possibly infinite database $\text{chase}(I, DB_0)$ is built from $DB_0$ by adding iteratively new tuples to satisfy inclusion dependencies in $I$, until no dependency is violated by the current database (see for instance [11]). In the following, it is convenient to represent $\text{chase}(I, DB_0)$ as a tree $T$ of tuples rooted at $r_w(c_{X_1}, \ldots, c_{X_m})$, and where edges are built as follows. Let $DB_i$ denote the set of all the tuples in $\text{chase}(I, DB_0)$ associated with nodes in the first $i$ levels of $T$ (the root is level 0). Let $r(t)$ be a node of $T$ at level $i$. For each inclusion dependency $r[A] \subseteq r'[A] \in I$ such that there is no tuple $r'(t') \in DB_i$ that matches with $r(t)$ over the attributes in $A$, a node $r'(t'')$ is added as a child of $r(t)$, where $r'(t'')$ is a fresh tuple that matches with $r(t)$ over the attributes in $A$ and contains fresh constants of the form $c_Y$, for any (other) attribute $Y \notin A$ in the schema of relation $r'$.

A well known property of $\text{chase}(I, DB_0)$ is that it maps via homomorphism to any other (possibly infinite) database that satisfies $I$ and includes the non-empty database $DB_0$. Therefore, whenever $\overline{Q}^{\text{chase}(I, DB_0)} \neq \emptyset$, the same holds for every database that satisfies $I$ and includes $DB_0$.

We now use the finite controllability result by Rosati [43] which, applied to our $Q$, $I$, and $DB_0$, reads as follows: the answer of $Q$ is not empty on every (possibly infinite) database that satisfies $I$ and includes $DB_0$ if, and only if, the
answer of \( \bar{Q} \) is not empty on every finite database that satisfies \( I \) and includes \( DB_0 \) (by Theorem 2 in [28]). Therefore, P2 implies the following property:

\[
P_3 \quad \text{\( \bar{Q}^{chase(I,DB_0)} \neq \emptyset. \)}
\]

Because \( \bar{Q} \) is a union of conjunctive queries, this means that there is a query \( Q' \) in \( \bar{Q} \) having a homomorphism \( h : \text{vars}(Q') \rightarrow U_e \) from \( Q' \) to \( \text{chase}(I,DB_0) \), where \( U_e \) is the universe of \( \text{chase}(I,DB_0) \). In particular, from a well known result of Johnson and Klug [36], we may assume, w.l.o.g., that \( h \) maps \( Q' \) to a finite subtree \( T_f \) of \( T \).

Observe now that \( h \) is a bijection. Indeed, \( DB_0 \) contains the one tuple \( r_w(c_{X_1}, \ldots, c_{X_m}) \) with a distinct constant for each attribute of \( r_w \) and, by definition of \( \text{chase}(I,DB_0) \), any constant \( c_Y \) can never be used for an attribute different from \( Y \). In fact, either \( c_Y \) belongs to the starting tuple and it is then propagated to fresh tuples by the chase generating-rule, or it is a fresh constant belonging to a tuple created to satisfy some inclusion dependency (which does not involve attribute \( Y \)). Moreover, recall that attributes in \( DS \) are in fact variables in \( Q' \), because the latter is a canonical query. Then, since \( h \) is a homomorphism, for each variable (attribute) \( Y \), \( h(Y) \) has the form \( c_Y \) for some constant \( c_Y \) occurring in tuples of \( \text{chase}(I,DB_0) \).

We now define a labeling \( \lambda \), associating each node of \( T_f \) with a set of variables in \( \text{vars}(Q') \). Let \( V = \{ h(X) \mid X \in \text{vars}(Q') \} \). For each vertex \( p = r(c_{Y_1}, \ldots, c_{Y_n}) \) in \( T_f \), define \( \lambda(p) \) as the set \( \{ h^{-1}(c_{Y_j}) \mid c_{Y_j} \in V \} \). Let \( p_1 \) and \( p_2 \) be two vertices of \( T_f \) such that \( X \in \lambda(p_1) \cap \lambda(p_2) \) is a variable in \( \text{vars}(Q') \). Consider the chase constant \( h(X) \), which occurs in \( p_1 \) and \( p_2 \) in \( T_f \). Let \( p_X \) be the top-most vertex of \( T_f \) where \( h(X) \) occurs. Because of the chase generating-rule, each node in the path from \( p_X \) to \( p_1 \) (resp., \( p_2 \)) contains the constant \( h(X) \). Thus, since \( T_f \) is a tree, \( h(X) \) occurs in the path between \( p_1 \) and \( p_2 \). Therefore, \( X \) occurs in \( \lambda \)-labeling of each vertex in this path, too.

Now consider the hypergraph \( \mathcal{H}_a \) containing exactly one hyperedge \( \lambda(p) \), for each vertex \( p \) of \( T_f \), and note that \( \mathcal{H}_a \) is acyclic, because we have actually just shown that the \( \lambda \)-labeling on \( T_f \) defines a join tree of \( \mathcal{H}_a \). Moreover, since \( h \) is a homomorphism from \( Q' \) to \( \text{chase}(I,DB_0) \), for each atom \( q \in \text{atoms}(Q') \) there exists a vertex \( p = h(q) \) in \( T_f \) for which \( \lambda(p) = \text{vars}(q) \); thus, \( \mathcal{H}_{Q'} \subseteq \mathcal{H}_a \). Finally, by construction, each hyperedge \( \lambda(p) \) in \( \mathcal{H}_a \) is built from a tuple \( p = r(c_{Y_1}, \ldots, c_{Y_n}) \) of \( \text{chase}(I,DB_0) \), hence a tuple of (the relation of) some canonical atom \( a_r \) in \( \mathcal{A}(DS) \). Moreover, we observed that, for each variable \( Y_i \in \lambda(p) \), \( h^{-1}(c_{Y_i}) = Y_i \in \text{vars}(a_r) \). Then, \( \lambda(p) \subseteq \text{vars}(a_r) \), and hence \( \mathcal{H}_a \subseteq \mathcal{H}_{\mathcal{A}(DS)} \). All in all, we have shown that, for the query \( Q' \) in \( \bar{Q} \), there is a tree projection of \( \mathcal{H}_{Q'} \) w.r.t. \( \mathcal{H}_{\mathcal{A}(DS)} \).

\[\text{3In particular, it is shown that this is equivalent to the condition \( \bar{Q}^{chase(I,DB_0,m)} \neq \emptyset \), where \( m \) is a finite natural number that depends on the given instance (including the query) and \( f^{chase}(I,DB_0,m) \) is the so-called finite chase, that is, a non-empty finite database playing the same role of the (possibly) infinite chase, as far as the evaluation of \( Q \) is concerned.}\]
Theorem 3.19 Assume there is no tree projection of \((\mathcal{H}_{Q'}, \mathcal{H}_V)\), for each core \(Q' \in \text{cores}(Q)\). Then, local consistency does not entail global consistency. In particular, there exists a (legal) database \(DB\) such that \(lc(V, DB)\) holds but \(Q^{DB} = \emptyset\).

Proof. Recall that we assumed w.l.o.g. that no constants or repeated variables occur in the views in \(V\), while the query \(Q\) has no restriction. Moreover, each view \(w \in V\) is over a distinct relation symbol (let us denote it by \(r_w\), in the following), so that there is a one-to-one correspondence between relations and views. Therefore, \(V\) identifies a database schema \(DS\) consisting of such a relation \(r_w\), for each \(w \in V\), whose list of attributes is precisely the list of variables of the view \(w\). Thus, \(V\) is by construction the set of canonical atoms associated with \(DS\).

Let us equip \(DS\) with the following set \(I\) of inclusion dependencies: For each pair of views \(w, w' \in V\) such that \(S = vars(w) \cap vars(w') \neq \emptyset\), \(I\) contains the two inclusion dependencies \(r_w[S] \subseteq r_{w'}[S]\) and \(r_{w'}[S] \subseteq r_w[S]\).

Observe that, by the construction of \(I\), for each database \(DB\) over \(DS\), \(lc(V, DB)\) holds if, and only if, \(DB\) satisfies \(I\) and \(DB \neq \emptyset\) (recall that \(Q\) is connected and \(vars(Q) = vars(V)\), hence \(\mathcal{H}_V\) is also connected because \(\text{views}(Q) \subseteq V\)).

For any set of atoms \(D\), let us denote by \(\bigwedge D\) the Boolean conjunctive query defined as the conjunction of all atoms in \(D\). Let \(Q = \bigvee_{Q' \in \text{cores}(Q)} (\bigwedge \text{views}(Q'))\) be the union of (Boolean) canonical queries for \(DS\) obtained by considering the cores of \(Q\), and assume that there is no tree projection of \((\mathcal{H}_{Q'}, \mathcal{H}_V)\), and hence of \((\mathcal{H}_{\text{views}(Q')}, \mathcal{H}_V)\), for each core \(Q' \in \text{cores}(Q)\). Then, by Lemma 3.18 there exists a (finite) database \(DB_f \neq \emptyset\) that satisfies \(I\) and such that \(\forall Q' \in \text{cores}(Q), (\bigwedge \text{views}(Q'))^{DB_f} = \emptyset\). In particular, because this database satisfies \(I, lc(V, DB_f)\) holds.

From \(DB_f\), let us now build a new legal database instance \(DB'_f\) over the vocabulary including both views and query atoms. This database is obtained by slightly changing the relations in \(DB_f\) in order to keep the information about the (active) domains of the variables, and by adding the relation instances for the query atoms in \(Q\). Recall that more query atoms may share the same database relation.

Let \(q \in \text{atoms}(Q)\) be any query atom defined over a relation symbol \(r\) of arity \(\rho\), and let \(r_{w_q}(X_1, \ldots, X_n) \in \text{views}(Q)\) be the query view \(w_q\) associated with \(q\). Recall that both constants and repeated variables may occur in \(q\), so that \(\rho \geq n\). Let \(r_{w_q}(c_1, \ldots, c_n)\) be any tuple in \(DB_f\). Then, \(DB'_f\) contains a tuple \(r_{w_q}(\langle X_1, c_1 \rangle, \ldots, \langle X_n, c_n \rangle)\) in the relation instance for the query view \(w_q\) in \(V\). Moreover, for the relation \(r\), \(DB'_f\) contains a tuple \(r(v_1, \ldots, v_\rho)\) defined as follows. For each \(i \in \{1, \ldots, \rho\}\): if some constant term \(u_i\) occurs in \(q\) at

\[\text{We remark that the assumption that no constant or repeated variables occur in views is just for the sake of presentation. If this assumption does not hold, it is sufficient to define instead a database schema } DS' \text{ obtained from } V \text{ by removing such useless occurrences, to use its canonical atoms, and to manage, after the described construction, the correspondence between relations in } DS' \text{ and views in } V.\]
position \(i\), then \(v_i = u_i\); if some variable \(X_j\) occurs in \(q\) at position \(i\), then \(v_i = (X_j, c_j)\). Note that this value may occur in \(r(v_1, \ldots, v_p)\) at different positions, if \(X_j\) occurs more than once in \(q\). Moreover, if the relation \(r\) is shared by different query atoms, such a tuple \(r(v_1, \ldots, v_p)\) will be available to every atom defined over \(r\), besides \(q\). Finally, for any (non-query view) \(w\) over a relation \(r_w\) and any tuple \(r_w(c_1, \ldots, c_n) \in DB_f\), \(DB_f\) contains a tuple \(r_w((X_1, c_1), \ldots, (X_n, c_n))\). No further tuples belong to \(DB_f\).

As \(lc(V, DB_f)\) holds, we immediately have that \(lc(V, DB_f')\) holds, too. We now claim that \(Q^{DB_f'} = \emptyset\), for each subquery \(Q' \in \text{cores}(Q)\), which entails \(Q^{DB_f} = \emptyset\). Indeed, assume for the sake of contradiction that there is a core \(Q'\) such that \(Q^{DB_f} \neq \emptyset\), and let \(h'\) be a homomorphism from \(Q'\) to \(DB_f\). Define \(\pi_1\) and \(\pi_2\) to be the projections mapping a binary tuple \(\langle u, c \rangle\) to its first element \(u\) and to its second element \(c\), respectively; moreover, for a (plain (term) element \(u\), \(\pi_1(u) = \pi_2(u) = u\). In particular, for any tuple \(r(v_1, \ldots, v_p)\) in \(DB_f\), where any value \(v_i\) is either of the form \((u_i, c_i)\) or of the form \(u_i\) with \(u_i\) being a constant term, we have \(\pi_1(r(v_1, \ldots, v_p)) = r(\pi_1(v_1), \ldots, \pi_1(v_p)) = r(u_1, \ldots, u_p)\). By construction of the tuples in \(DB_f\), the composition \(h' \circ \pi_1\) is a homomorphism from \(Q'\) to \(Q\) (if we obtain a certain tuple of terms after applying \(\pi_1\), there must exist some query atom with that tuple of terms). But, since \(Q'\) is a core, we have that the image \(Q'' = (h' \circ \pi_1)(Q')\) is also a core in \(\text{cores}(Q)\), and thus \(h' \circ \pi_1\) is actually an isomorphism. In particular, \(h'' = ((h' \circ \pi_1)^{-1} \circ h')\) is now such that \(h''(u_i) = \langle u_i, c_i \rangle\). In particular, whenever \(u_i = X\), for some variable \(X \in \text{vars}(Q'')\), \(h''(X) = \langle X, c_i \rangle\). It follows that \(h''\) is a homomorphism from \(Q''\) to \(DB_f\). Then, we immediately get that \(h'' \circ \pi_2\) is a homomorphism from \(\bigwedge \text{views}(Q'')\) to \(DB_f\). Indeed, by construction, for each atom \(q \in \text{atoms}(Q'')\) defined on a relation \(r\), if \(r(u_1, \ldots, u_p) \in DB_f\), then \(\pi_2(r_w(\bar{u}_1, \ldots, \bar{u}_n) \in DB_f\) (with \(\bar{u}_1, \ldots, \bar{u}_n\) being the tuple derived from \(\langle u_1, \ldots, u_p \rangle\) by inverting the above construction, i.e., by eliminating constants and repeated variables). However, the existence of this homomorphism contradicts the fact that \(\bigwedge \text{views}(Q'')^{DB_f} = \emptyset\) holds by the construction of \(DB_f\).

Finally, note that \(DB_f\) is legal. Indeed, for each query view \(w_q\), by construction \(w_q^{DB_f'} \subseteq q^{DB_f'}\), and \(w_q\) is trivially view consistent because \(Q^{DB_f} = \emptyset\).

A consequence of the above result and Theorem 3.17 is the precise characterization of the power of local consistency, as far as the decision problem is concerned. This characterization was so far only known for the special case of treewidth and for structures of fixed arity [5], where, however, all the cores enjoy the same structural properties (and hence such results are defined in terms of “the core” of the query).

**Corollary 3.20** The following are equivalent:

1. For every database \(DB\), \(lc(V, DB)\) entails \(Q^{DB} \neq \emptyset\).
2. There is a subquery \(Q' \approx_{\text{hom}} Q\) for which \((H_{Q'}, H_Y)\) has a tree projection.
3. There is a core \(Q''\) of \(Q\) for which \((H_{Q''}, H_Y)\) has a tree projection.
Proof. From Theorem 3.17, we know that (2) implies (1). Theorem 3.19 entails that (1) implies (3). Finally, (3) implies (2) because any core of $Q$ is homomorphically equivalent to $Q$. □

Eventually, we can specialize Corollary 3.20 to the setting of simple queries (considered in many seminal papers about tree projections, as [29]), where every relation symbol occurs at most once in the query and thus the whole query is its (unique) core.

**Corollary 3.21** Let $Q$ be a simple query. Then, the following are equivalent:

1. For every database $DB$, $lc(V, DB)$ entails $Q^{DB} \neq \emptyset$.

2. $(H_Q, H_V)$ has a tree projection.

**Example 3.22** Consider the query

$$Q_7 : \ r_1(A, B) \land r_2(B, C) \land r_3(C, D) \land r_4(D, E) \land r_5(A, E),$$

the set of views $V_7 = \{v_1(A, B, E), v_2(B, C, E), v_3(A, C, E), v_4(A, C, D), v_5(A, D, E)\}$, and the database instance $DB_7$ depicted in Figure 5. It is easy to check that $(V_7 \cup \text{atoms}(Q_7))^{DB_7}$ is local consistent but $Q_7^{DB_7} = \emptyset$. Indeed, it can be checked that $(H_{Q_7}, H_{V_7})$ does not have a tree projection. △

**Computation Problem.** We next complete the picture and give the conditions that precisely characterize those cases where answers of the query over output variables covered by some view may be immediately obtained by enforcing local
consistency. Again, we start with the problem where we are interested in query answers over some arbitrary set of output variables. In this case, requiring that just some view covering \( O \) is trustable is sufficient to allow all such answers to be immediately obtained.

**Theorem 3.23** Let \( O \) be any set of variables occurring in some view in \( \mathcal{V} \). Then, the following are equivalent:

1. For each database \( \text{DB} \) such that \( \text{lC}(\mathcal{V}, \text{DB}) \) holds, \( w^{\text{DB}}[O] \subseteq Q^{\text{DB}}[O] \), for every \( w \in \text{covers}(O) \). If there is a view consistent \( \bar{w}^{\text{DB}} \) with \( \bar{w} \in \text{covers}(O) \), then \( w^{\text{DB}}[O] = Q^{\text{DB}}[O] \), for every \( w \in \text{covers}(O) \).

2. The set of variables \( O \) is tp-covered in \( Q \) (w.r.t. \( \mathcal{V} \)).

**Proof.** First observe that (2) entails (1), by Lemma 3.15. Then, in order show that (1) entails (2), it suffices to consider the case where there exists \( Q'' \in \text{cores}(Q) \) for which \( (\mathcal{H}_{Q''}, \mathcal{H}_V) \) has a tree projection. Otherwise, we immediately get the contradiction that all views are incorrect for some database, from Theorem 3.19. Consider the new query \( Q_e = Q \land \text{atom}(O) \), and assume by contradiction that \( O \) is not tp-covered in \( Q \). That is, for every \( Q' \in \text{cores}(Q_e) \), \( (\mathcal{H}_{Q_e}, \mathcal{H}_V) \) has no tree projections. We show that there exists a database \( \text{DB} \) such that \( \text{lC}(\mathcal{V}, \text{DB}) \) but \( Q^{\text{DB}}[O] \nsubseteq a^{\text{DB}}[O] \), for every \( a \in \text{covers}(O) \), where \( \text{covers}(O) \neq \emptyset \), by hypothesis.

Let \( \mathcal{V}_e = \mathcal{V} \cup \{ \text{atom}(O) \} \). Since no core of \( Q_e \) has tree projections, by Theorem 3.14, it follows that there is a (non-empty legal) database \( \text{DB}' \) such that \( \text{lC}(\mathcal{V}_e, \text{DB}') \), but \( Q^{\text{DB}'}[O] = \emptyset \). Now define a new database \( \text{DB} \) such that, for every \( a \in \mathcal{V}_e \), \( a^{\text{DB}} = a^{\text{DB}'} \cup Q^{\text{DB}'}[a] \), and where the relations in \( \text{DB}' \) over which the original query atoms are defined are just copied into \( \text{DB} \). By construction, \( \text{lC}(\mathcal{V}_e, \text{DB}) \) holds, because \( \text{lC}(\mathcal{V}_e, \text{DB}') \) holds and the tuples possibly added to any view are projections of mappings over the full set of variables, as they are obtained from the total homomorphisms in \( Q^{\text{DB}'} \). Moreover, note that only views are modified, as no tuple is added to the relations over which the original atoms in the query are defined. Thus, \( Q^{\text{DB}} = Q^{\text{DB}'} \) holds.

Observe that \( \text{DB} \) is a legal database instance w.r.t. \( Q \). Indeed, the relations for query views are still subsets of the relations of the original query atoms (as in \( \text{DB}' \)). Moreover, by construction, they include all tuples that are part of some query answer, and thus all query views are view consistent w.r.t. \( Q \).

Recall now that we are considering the case where some cores of \( Q \) have tree projections, and \( \text{lC}(\mathcal{V}_e, \text{DB}) \) and hence \( \text{lC}(\mathcal{V}, \text{DB}) \) hold. From Theorem 3.17, it follows that \( Q^{\text{DB}} = Q^{\text{DB}'} \neq \emptyset \). However, \( (Q \land \text{atom}(O))^{\text{DB}'} = \emptyset \). It follows that all homomorphisms that are answers of \( Q \) over \( \text{DB}' \) does not satisfy \( \text{atom}(O) \), that is, \( Q^{\text{DB}'}[O] \cap \text{atom}(O)^{\text{DB}'} = \emptyset \), and recall that \( \text{atom}(O)^{\text{DB}'} \neq \emptyset \), because \( \text{lC}(\mathcal{V}_e, \text{DB}') \) holds.

Therefore, we get the proper inclusion \( Q^{\text{DB}}[O] \subset \text{atom}(O)^{\text{DB}} \). Indeed, \( \text{atom}(O)^{\text{DB}'} \) is not empty and all its tuples, which do not belong to \( Q^{\text{DB}}[O] = Q^{\text{DB}'}[O] \neq \emptyset \), are kept in \( \text{atom}(O)^{\text{DB}} \). Finally, since \( \mathcal{V}^{\text{DB}} \) and hence \( \mathcal{V}^{\text{DB}} \) are locally consistent, this also entails \( \text{atom}(O)^{\text{DB}} = a^{\text{DB}}[O] \) and thus \( Q^{\text{DB}}[O] \subset a[O]^{\text{DB}} \), for each view.
\[ a \in \text{covers}(O). \]

The following corollary is the specialization to the case where we are interested in output variables covered by some query atom.

**Corollary 3.24** The following are equivalent:

1. For every database DB, lc(\(\mathcal{V},\mathcal{D}B\)) entails gc(\(\mathcal{V},\mathcal{D}B, Q\)).
2. For each \(q \in \text{atoms}(Q)\), vars(q) is \(tp\)-covered in Q (w.r.t. \(\mathcal{V}\)).

**Proof.** Since query views covers the variables of query atoms and are always view consistent w.r.t. Q in any legal database, the statement immediately follows from Theorem 3.23 and Theorem 3.16.

The specialization of Corollary 3.24 to the setting where every relation symbol occurs at most once in the query provides the answer to the question posed by [29].

**Corollary 3.25** Let Q be a simple query. Then, the following are equivalent:

1. For every database DB, lc(\(\mathcal{V},\mathcal{D}B\)) entails gc(\(\mathcal{V},\mathcal{D}B, Q\)).
2. \((\mathcal{H}_Q, \mathcal{H}_\mathcal{V})\) has a tree projection.

Finally, we point out that Theorem 3.23 may be equivalently stated in terms of any arbitrary (legal) database DB, by considering its reduct \(\text{red}(\mathcal{V}, \mathcal{D}B)\) obtained enforcing local consistency.

**Corollary 3.26** Let O be any set of variables occurring in some view in \(\mathcal{V}\). Then, the following are equivalent:

1. For each database DB, \(w^{\mathcal{D}B'}[O] \subseteq Q^{\mathcal{D}B}[O]\), for every \(w \in \text{covers}(O)\), where DB' = \(\text{red}(\mathcal{V}, \mathcal{D}B)\). If there is a view consistent \(w^{\mathcal{D}B'}\) with \(\bar{w} \in \text{covers}(O)\), then \(w^{\mathcal{D}B'}[O] = Q^{\mathcal{D}B}[O]\), for every \(w \in \text{covers}(O)\).
2. The set of variables O is \(tp\)-covered in Q (w.r.t. \(\mathcal{V}\)).

**Proof.** (1) \(\Rightarrow\) (2) follows from the corresponding implication (1) \(\Rightarrow\) (2) in Theorem 3.23 which entails that, whenever O is not \(tp\)-covered in Q (w.r.t. \(\mathcal{V}\)), there exists a locally-consistent legal database DB and a view \(w \in \text{covers}(O)\) such that \(w^{\mathcal{D}B}[O] \supset Q^{\mathcal{D}B}[O]\). In fact, because it is locally consistent, \(\mathcal{D}B = \text{red}(\mathcal{V}, \mathcal{D}B)\) holds.

(2) \(\Rightarrow\) (1) follows from the corresponding implication in Theorem 3.23 and from the fact that the only tuples occurring in DB and deleted in its reduct DB' do not participate in any query answer. Therefore DB' is a legal locally consistent database. \[\square\]
4 Application to Structural Decomposition Methods

In this section, we specialize our results about consistency properties and tree projections to the purely structural decomposition methods described in the literature (both in the database and in the constraint satisfaction area), because all of them can be recast in terms of tree projections. In fact, each of them can be seen as a method to define suitable set of views to be exploited for solving the given query answering instance. Here, views represent subproblems over subsets of variables, whose solutions can be computed efficiently.

We also provide further results that hold on such special cases only, such as the positive answer to the question in [12] about $k$-local consistency and generalized hypertree decomposition, and the precise relationship between acyclic queries and local consistency, solved in [7] for the simple queries.

4.1 Decomposition Methods and Views

We start by formalizing the concept of structural decomposition method in our framework. Let the pair $(Q, DB)$ be any query answering problem instance. For any subset of variables $S \subseteq \text{vars}(Q)$, let $(Q|_S, DB|_S)$ be the subproblem of $(Q, DB)$ induced by $S$ defined as follows: for each atom $a \in \text{atoms}(Q)$ with $\text{vars}(a) \cap S \neq \emptyset$, $Q|_S$ contains an atom $a'$ over a fresh relation symbol $r_a'$ having $\text{vars}(a) \cap S$ as its set of variables, and whose database relation is such that $a'^{DB|_S} = a^{DB}[S]$. No further atom belongs to $Q|_S$, and no further relation belongs to $DB|_S$. Intuitively, $(Q|_S, DB|_S)$ is the most constrained subproblem of $(Q, DB)$ where only variables from $S$ occur, because all atoms involving (even partially) those variables are considered. In particular, for each subquery $Q'$ whose set of variables is $S$, we have $Q'^{DB} \supseteq Q'^{DB|_S} \supseteq Q^{DB}[S]$.

**Definition 4.1** A structural decomposition method $\mathcal{DM}$ is a pair of polynomial-time computable functions $v_{\mathcal{DM}}$ and $d_{\mathcal{DM}}$ that, given a conjunctive query $Q$ and a database $DB'$, compute, respectively, a view system $\mathcal{V} = v_{\mathcal{DM}}(Q)$ and a database $DB'' = d_{\mathcal{DM}}(Q, DB')$ over the vocabulary of $\mathcal{V}$ such that:

- the database $DB = DB' \cup DB''$ over the (disjoint) vocabularies of $Q$ and $\mathcal{V}$ is legal;
- for each $w \in \mathcal{V}$, $w^{DB} \supseteq Q^{DB|_{\text{vars}(w)}}$. That is, any view $w$ contains at least the solutions of the subproblem of $(Q, DB)$ induced by its variables (subproblem completeness). □

Note that the above completeness property is a local property, and clearly entails the (global) view consistency property for $\mathcal{V}^{DB}$.

\footnote{For the sake of presentation, we do not consider FPT decomposition methods (where functions $v_{\mathcal{DM}}$ and $d_{\mathcal{DM}}$ are computable in fixed-parameter polynomial-time), but our results can be extended easily to them.}
Every known purely-structural decomposition method $\mathcal{DM}$, where views (subproblems) are only determined by the query and do not depend on the database instance, can be recast this way, with decompositions of $Q$ according to $\mathcal{DM}$ being tree projections of $(H_Q, H_V)$. Indeed, all such methods are in fact subproblem-based, because any view relation $w_{\mathcal{DM}}^Q$ is instantiated with the solutions $Q_{\mathcal{DM}}^Q$ of some subquery $Q'$ (depending on the specific method), which is not necessarily an induced subproblem. Some exemplifications of the above definition are discussed below.

**Tree Decompositions.** For any fixed natural number $k$, the tree decomposition method \cite{17, 21} ($tw_k$) is characterized by the functions $v_{\mathcal{tw}}$ and $d_{\mathcal{tw}}$ that, given a query $Q$ and a database $DB$, build the view system $v_{\mathcal{tw}}(Q)$ and the database $d_{\mathcal{tw}}(Q, DB)$. In particular, for each subset $S$ of at most $k + 1$ variables, there is a view $w_S$ over the variables in $S$ (i.e., $vars(w_S) = S$) whose tuples are the solutions of the subproblem induced by $S$ (or, more liberally, the cartesian product of the set of constants that variables in $S$ may take). An illustration of the view set characterizing treewidth is reported below.

**Example 4.2** Consider the query

$$Q_8 : r_1(A, B) \land r_2(B, C) \land r_3(A, C) \land r_4(C, D),$$

whose associated hypergraph is depicted on the left of Figure 6. Consider the application on $Q_8$ of the tree decomposition method. The set of views $v_{\mathcal{tw}}(Q_8)$ defined by this method for $k = 2$ is graphically illustrated on the right of Figure 6. In fact, the figure shows how $Q_8$ can be covered via an acyclic hypergraph that consists of two hyperedges covered by two available views, the largest of which includes three variables. In fact, the treewidth of $Q_8$ is 2.

**Generalized Hypertree Decompositions.** For any fixed natural number $k$, the generalized hypertree decomposition method \cite{24} (short: $hw_k$) is characterized by the functions $v_{\mathcal{hw}}$ and $d_{\mathcal{hw}}$ that, given a query $Q$ and a database $DB$, build the view system $v_{\mathcal{hw}}(Q)$ and the database $d_{\mathcal{hw}}(Q, DB)$ where, for each subquery $Q'$ of $Q$ such that $|\text{atoms}(Q')| \leq k$, there is a view $w_{Q'}$ over all variables in $Q'$ (i.e., $vars(w_{Q'}) = vars(Q')$) and whose tuples are the answers of $Q'$. Note that $hw_k$ satisfies the subproblem completeness property too, because
$Q'$ is in general more liberal than the subproblem induced by $\text{vars}(Q')$. Indeed, the latter also deals with further atoms where such variables occur (possibly together with other variables not occurring in $Q'$).

**Acyclicity.** Recall that a hypergraph is acyclic if, and only if, it has (generalized) hypertree width 1 [24]. Therefore, the acyclicity method (short: acyc) is just the specialization of the above method for the case of $k = 1$. In particular, $v$-acyc($Q$) is precisely the set of query views views($Q$).

**Fractional Hypertree Decompositions.** For any fixed natural number $k$, consider the subqueries characterizing the fractional hypertree decomposition method [35]: they are defined precisely as in the case of the generalized hypertree decomposition method, except that a view $w_{Q'}$ is built more generally if, for a subquery $Q'$, its hypergraph $\mathcal{H}_{Q'}$ has fractional edge-cover number at most $k$. Unfortunately, these views may be exponentially-many even if $k$ is a fixed constant, and in fact there is no known polynomial time algorithm to decide whether the fractional hypertree-width of a hypergraph is at most $k$. However, we may still define the required pair of polynomial-time functions $v$-fwp and $d$-fwp for this decomposition method, by actually exploiting for their computation the subproblems identified by Marx in his $O(k^3)$ polynomial-time approximation of the fractional hypertree-width [40]. Moreover, following the same kind of arguments used for the generalized hypertree decompositions, it can be seen that the subproblem completeness property is satisfied by such a pair of functions, too.

**Submodular Width.** For the sake of completeness, note that the only known decomposition technique that does not fit the above framework is the one based on the submodular width [41]. This method is in fact not “purely” structural. Indeed, according to this technique, a number of view schemas are computed in fixed-parameter polynomial time (hence not polynomial-time, in general) by looking at the database $DB$ of the given instance, too (while $v$-$DM$ functions depend on the query only). Moreover, their associated database relations are not necessarily subproblem-complete.

### 4.2 Decomposition Methods and Consistency Properties

By using Theorem 3.23 it is possible to characterize the power of local-consistency based algorithms in structural decomposition methods, as stated in the following result. In fact, this result is not a trivial consequence of Corollary 3.26 as it is evident by contrasting their statements: here, the database DB” for the views is computed from a database over the vocabulary of the query $Q$ only, according to the specific function $d$-$DM$ characterizing the method DM, while it is an arbitrary (legal) database in Corollary 3.26.

**Theorem 4.3** Let $DM$ be a decomposition method, let $Q$ be a conjunctive query, and let $V = v$-$DM(Q)$. The following are equivalent:

---

6For completeness, we observe that a similar result has been proved in [32] in the setting of constraint satisfaction problems, by precisely exploiting Theorem 3.23.
(1) For every database $DB$ (over the vocabulary of $Q$) and for every view $w \in V$ with $O \subseteq \text{vars}(w)$, $w^{DB}[O] = Q^{DB}[O]$, where $DB' = \text{red}(V, DB'')$ and $DB'' = d-\text{DM}(Q, DB)$.

(2) A set of variables $O \subseteq \text{vars}(Q)$ is $tp$-covered in $V$.

Proof. The fact that (2) $\Rightarrow$ (1) immediately follows from Corollary 3.26. We have to show that (1) $\Rightarrow$ (2) holds as well. Observe that if $O$ is not $tp$-covered in $Q$ w.r.t. $V = v-\text{DM}(Q)$, by Theorem 3.23 we conclude the existence of a locally consistent (legal) database $DB = DB_Q \cup DB_V$, with $DB_Q$ being over the vocabulary of $Q$ and with $DB_V$ being over the vocabulary of $V$, respectively, and the existence of a view $v \in V$ such that $w^{DB}[O] \supset Q^{DB}[O]$, with $O \subseteq \text{vars}(\bar{w})$. Let $DB'' = d-\text{DM}(Q, DB_Q)$ be the database comprising the relations for the views in $V$ built according to method $\text{DM}$, and let $DB' = \text{red}(V, DB'')$ be its reduct, obtained by enforcing local consistency.

We first claim that the database $DB_V$ is included in $DB''$, formally, for any $w \in V$, $w^{DB} = w^{DB'} \subseteq w^{DB''}$. Consider such a view $w \in V$, having variables $S = \text{vars}(w)$, and the subproblem $(Q_S, DB_S)$ of $(Q, DB)$ induced by $S$. By construction, only variables from $S$ occur in $Q_S$ and thus, for each atom $a \in \text{atoms}(Q_S)$, $\text{vars}(a) \subseteq S$. It trivially follows that the pair of hypergraphs $(H_{Q_S}, H_V)$ has a tree projection. Let $V^+ = V \cup \text{views}(Q_S)$ be the set of views obtained by adding to $V$ the query views associated with the induced subproblem, and let $DB^+ = DB^+ \subseteq DB^+$ be the database obtained by adding to $DB$ the relations in $DB_S$, as well as their copies on the relation symbols of the query views $\text{views}(Q_S)$. Clearly, $DB^+$ is a legal database (w.r.t. $Q^+$ and $V^+$) and $V^+$ is a view system for $Q^+$. Moreover, since we just added new views to $V$, the pair $(H_{Q_S}, H_{V^+})$ has a tree projection, too. In particular, from Fact 3.11, $S$ is $tp$-covered in $Q_S$ w.r.t. $V^+$. Moreover, observe that the database relations for the new views in $V^+$ are just projections of the relations of the original query views, which already belong to $V$. Therefore, their presence has no impact on the local consistency property, and $\text{lc}(V^+, DB^+) = \text{lc}^{\text{reduct}}(V^+, DB^+)$ holds. By Theorem 3.23 for every $O' \subseteq S$, we get $w^{DB}[O'] = w^{DB'}[O'] \subseteq Q^{DB}[O']$. That is, $w^{DB}$ contains only solutions of the subproblem induced by $w$. On the other hand, the subproblem completeness condition entails that $w^{DB''} \supseteq Q^{DB}[O']$. Hence, the claim follows, as for any chosen $w \in V$ with variables $S = \text{vars}(w)$, $w^{DB} \subseteq Q^{DB}[O'] \subseteq w^{DB''}$.

To conclude, recall that $\text{lc}(V, DB_V)$ holds, so that $DB_V$ is a locally consistent database included in $DB''$, and thus all its tuples will survive after enforcing local consistency on $DB''$, that is, all of them belongs to the reduct $DB' = \text{red}(V, DB'')$. Therefore, $w^{DB} \subseteq w^{DB'}$, $\forall w \in V$. In particular, for the view $\bar{w}$ and the set of variables $O \subseteq \text{vars}(\bar{w})$, we get $Q^{DB}[O] \subseteq \bar{w}^{DB}[O] \subseteq \bar{w}^{DB''}[O]$, hence we get wrong solutions (over $O$) using the view $\bar{w}$ with the database $DB'$.

For the decision problem ($O = \emptyset$), we get the following special case.
Corollary 4.4 Let $DM$ be a decomposition method, let $Q$ be a conjunctive query, and let $V = v-DM(Q)$. The following are equivalent:

1. For every database $DB$ (over the vocabulary of $Q$), $\text{red}(V, d-DM(Q, DB)) \neq \emptyset$ entails $Q_{DB} \neq \emptyset$.

2. There is a subquery $Q' \approx_{\text{hom}} Q$ for which $(\mathcal{H}_{Q'}, \mathcal{H}_V)$ has a tree projection.

3. There is a core $Q''$ of $Q$ for which $(\mathcal{H}_{Q''}, \mathcal{H}_V)$ has a tree projection.

If we consider decision problem instances ($O = \emptyset$) and the treewidth method ($V = v-\text{tw}_k(Q)$), from Corollary 4.4 we (re-)obtain the nice characterization of [6] about the relationship between $k$-local consistency and the treewidth of the core of $Q$.

If we consider the generalized hypertree-width ($V = v-\text{hw}_k(Q)$), we next provide the answer to the corresponding open question for the unbounded arity case. Recall that in [12] it was shown that if the core of $Q$ has generalized hypertree-width at most $k$, then the procedure enforcing $k$-union (of constraints/atoms) consistency is always correct, i.e., the reduct of the database is not empty if, and only if, the query has some answer. We next show that this sufficient condition is necessary, too.

In fact, observe that the following result does not follow immediately from Corollary 4.4. Indeed, any core $Q'$ of $Q$ may be much smaller than $Q$, and thus the set of views $v-\text{hw}_k(Q')$ available using $Q'$ is in general (possibly much) smaller than the set of views $v-\text{hw}_k(Q)$ available when the whole query $Q$ is considered. For an extreme example, think of the undirected grid (see again Figure 4), where any edge is a core: in this case, the set of available views for computing a hypertree decomposition of the core is precisely this one edge (for any $k$), while considering the whole query, the available views comprise all unions of $k$ edges.

This subtle issue is irrelevant for the treewidth method, because such a technique considers all possible combinations of at most $k$ variables, and clearly only those variables occurring in the core are useful for computing any of its tree decompositions. Instead, when generalized hypertree decomposition is considered, in principle using some particular combination of variables occurring in some atom outside any core $Q'$ may be necessary for getting a width-$k$ generalized hypertree decomposition of $Q'$.

Theorem 4.5 Let $Q$ be a conjunctive query, and let $V = v-\text{hw}_k(Q)$. The following are equivalent:

1. For every database $DB$ (over the vocabulary of $Q$), $\text{red}(V, d-\text{hw}_k(Q, DB)) \neq \emptyset$ entails $Q_{DB} \neq \emptyset$.

2. There is a subquery $Q' \approx_{\text{hom}} Q$ having generalized hypertree-width at most $k$.

\[\text{As already observed, for treewidth and (generalized) hypertree-width isomorphic substructures behave in the same way, so that all cores have equivalent properties. Thus, for these methods one may simply say “the core” $Q'$ (instead of some core).}\]
(3) There is a core $Q'$ of $Q$ having generalized hypertree-width at most $k$.

**Proof.** It suffices to show that (3) is equivalent to (3') below. Then, the theorem follows from Corollary 4.4.

(3') There is a core $Q'$ of $Q$ for which $(\mathcal{H}_{Q'}, \mathcal{H}_V)$ has a tree projection, with $V = v-hw_k(Q)$.

Let $V' = v-hw_k(Q')$. Note that (3) is equivalent to say that $(\mathcal{H}_{Q'}, \mathcal{H}_{V'})$ has a tree projection, which entails (3'), because $Q'$ is a subquery of $Q$ and thus $\mathcal{H}_{V'} \subseteq \mathcal{H}_V$.

It remains to show that (3') $\Rightarrow$ (3). Assume by contradiction that this is not the case, hence there is a core $Q'$ of $Q$ for which $(\mathcal{H}_{Q'}, \mathcal{H}_V)$ has a tree projection $\mathcal{H}$, but every core of $Q$ has generalized hypertree width greater than $k$. In particular, this must hold for $Q'$, too. It follows that there exists some hyperedge $h$ that belongs to $\mathcal{H}$ and thus is covered by some hyperedge of $\mathcal{H}_V$, but it is not covered by any hyperedge of $\mathcal{H}_{V'}$, where $V' = v-hw_k(Q')$. That is, there is no view $w$ in $V'$ such that $h \subseteq \text{vars}(w)$. Recall that, by definition of function $v-hw_k$, views in $V$ (resp., $V'$) contain the union of variables from all possible sets of at most $k$ atoms occurring in $Q$ (resp., $Q'$). It follows that there is some atom $a \in \text{atoms}(Q)$ with $X = \text{vars}(a) \cap h \neq \emptyset$ which does not belong to $Q'$ and whose role in $w$ cannot be played by any other atom in $Q'$. Formally, there is no atom $a' \in \text{atoms}(Q')$ such that $X' \subseteq \text{vars}(a')$, where $X' = X \cap \text{vars}(Q')$.

In fact, note that $X'$ are the only possible crucial variables: further variables of $w$ not occurring in $Q'$ are never necessary in any tree projection of $\mathcal{H}_{Q'}$ (w.r.t. any hypergraph), as it is known and easy to see that, if a tree projection exists, there always exists one that uses only nodes from $\mathcal{H}_{Q'}$.

However, $Q'$ is a core of $Q$, and thus it is a retract, which means that there must exist a homomorphism $f$ from $Q$ to $Q'$ where $f(X) = X$, for each term $X$ occurring in $Q'$. Therefore, the atom $a$ should be mapped to some atom $a' \in \text{atoms}(Q')$ that contains all variables $f(X)$ for each $X \in \text{vars}(a)$. In particular, this entails that all variables in $X'$ occur in $a'$, because $f$ is the identity mapping over them. Contradiction. \qed

For the special case of $k = 1$, the above result provides the precise relationship between local consistency and acyclic queries, extending the classical result given in [7] for simple queries (in fact, for acyclic schemas). Recall that, for the acyclic method, the set of views $v$-acyclic$(Q)$ is just the set of query views views$(Q)$, and their database relations in $d$-acyclic$(Q, DB)$ are just the copies of their corresponding query atoms.

**Theorem 4.6** For any conjunctive query $Q$, the following are equivalent:

1. For every database $DB$ (over the vocabulary of $Q$), red$(v$-acyclic$(Q), d$-acyclic$(Q, DB)) \neq \emptyset$ entails $Q^\text{DB} \neq \emptyset$.
2. There is an acyclic subquery $Q' \approx_\text{hom} Q$.
3. $Q$ has an acyclic core.
5 Larger Islands of Tractability

In this section, we investigate a tractable variant of the notion of tree projections that allows us to identify new islands of tractability for query answering, constraint satisfaction problems, and further problems that are easy on tree-like structures. Indeed we argue that, in practical database applications, “blind” local-consistency enforcing procedures are hardly used, because the number of semijoin operations to be performed depends on the database size and may be very high. On the other hand, if one is able to compute a tree projection, then the views to be processed will be only those involved in the tree projection, and the number of semijoin operations to be performed will be at most the number of these views (hence, independent of the database).

The new notion is based on the game characterization of tree projections proposed in [30]. To formalize our results, we need to introduce some additional definitions and notations, which will be intensively used in the following.

Assume that a hypergraph $H$ is given. Let $V$, $W$, and $\{X,Y\}$ be sets of nodes. Then, $X$ is said $[V]$-adjacent (in $H$) to $Y$ if there exists a hyperedge $h \in \text{edges}(H)$ such that $\{X,Y\} \subseteq (h - V)$. A $[V]$-path from $X$ to $Y$ is a sequence $X = X_0, \ldots, X_{\ell} = Y$ of nodes such that $X_i$ is $[V]$-adjacent to $X_{i+1}$, for each $i \in [0...\ell-1]$. We say that $X$ $[V]$-touches $Y$ if $X$ is $[\emptyset]$-adjacent to $Z \in \text{nodes}(H)$, and there is a $[V]$-path from $Z$ to $Y$; similarly, $X$ $[V]$-touches the set $W$ if $X$ $[V]$-touches some node $Y \in W$. We say that $W$ is $[V]$-connected if $\forall X, Y \in W$ there is a $[V]$-path from $X$ to $Y$. A $[V]$-component (of $H$) is a maximal $[V]$-connected non-empty set of nodes $W \subseteq (\text{nodes}(H) - V)$. For any $[V]$-component $C$, let $\text{edges}(C) = \{h \in \text{edges}(H) \mid h \cap C \neq \emptyset\}$, and for a set of hyperedges $H \subseteq \text{edges}(H)$, let $\text{nodes}(H)$ denote the set of nodes occurring in $H$, that is $\text{nodes}(H) = \bigcup_{h \in H} h$. For any component $C$ of $H$, we denote by $\text{Fr}(C, H)$ the frontier of $C$ (in $H$), i.e., the set $\text{nodes}(\text{edges}(C))$. Moreover, $\partial(C, H)$ denote the border of $C$ (in $H$), i.e., the set $\text{Fr}(C, H) \setminus C$. Note that $C_1 \subseteq C_2$ entails $\text{Fr}(C_1, H) \subseteq \text{Fr}(C_2, H)$.

In the following sections, given any pair of hypergraphs $(H_1, H_2)$ and a set of nodes $C \subseteq H_1$, we write for short $\text{Fr}(C)$ and $\partial C$ to denote $\text{Fr}(C, H_1)$ and $\partial(C, H_1)$, respectively.

5.1 Game-Theoretic Characterization

The Robber and Captain game is played on a pair of hypergraphs $(H_1, H_2)$ by a Robber and a Captain controlling some squads of cops, in charge of the surveillance of a number of strategic targets. The Robber stands on a node and can run at great speed along the edges of $H_1$. However, (s)he is not permitted to run trough a node that is controlled by a cop. Each move of the Captain involves one squad of cops, which is encoded as a hyperedge $h \in \text{edges}(H_2)$. The Captain may ask some cops in the squad $h$ to run in action, as long as they

---

\[\text{Fr}(C_1, H) \subseteq \text{Fr}(C_2, H)\]

8The choice of the term “frontier” to name the union of a component with its outer border is due to the role that this notion plays in the hypergraph game described in the subsequent section.
occupy nodes that are currently reachable by the Robber, thereby blocking an escape path for the Robber. Thus, “second-lines” cops cannot be activated by the Captain. Note that the Robber is fast and may see cops that are entering in action. Therefore, while cops move, the Robber may run through those positions that are left by cops or not yet occupied. The goal of the Captain is to place a cop on the node occupied by the Robber, while the Robber tries to avoid her/his capture.

**Definition 5.1** Let \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \) be two hypergraphs. The Robber and Captain game on \( (\mathcal{H}_1, \mathcal{H}_2) \) is formalized as follows. A position for the Captain is a pair \((h, M)\) where \( h \) is a hyperedge of \( \mathcal{H}_2 \) and \( M \subseteq h \). A configuration is a triple \((h, M, C)\), where \((h, M)\) is a position for the Captain, and \( C \) is the \([M]\)-component where the Robber stands.\(^9\) The initial configuration is \((\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))\).

A strategy \( \sigma \) is a function that encodes the moves of the Captain. Its domain includes the initial configuration. For each configuration \( v_p = (h_p, M_p, C_p) \) in the domain of \( \sigma \), \( \sigma(v_p) = (h_r, M_r) \), with \( M_r \subseteq h_r \cap \text{Fr}(C_p) \), is the novel position for the Captain. After this move, the Robber can select any \([v_p, M_r]\)-option, i.e., any \([M_r]\)-component \( C_r \) such that \( C_p \cup C_r \) is \([M_p \cap M_r]\)-connected. If there is no \([v_p, M_r]\)-option, then \((h_r, M_r, \emptyset)\) is said a capture configuration induced by \( \sigma \). The move of the Captain is monotone if, for each \([v_p, M_r]\)-option \( C_r \), \( C_r \subseteq C_p \). The domain of \( \sigma \) includes the configuration \((h_r, M_r, C_r)\), for each \([v_p, M_r]\)-option \( C_r \). No other configuration is in the domain of \( \sigma \). The strategy \( \sigma \) is monotone if it encodes only monotone moves over the configurations in its domain.

A strategy \( \sigma \) can be represented as a directed graph \( G(\sigma) = (N, A) \), called strategy graph, as follows. The set \( N \) of nodes is the set of all configurations in the domain of \( \sigma \) plus all capture configurations induced by \( \sigma \). If \( v_p = (h_p, M_p, C_p) \) is a configuration and \( \sigma(v_p) = (h_r, M_r) \), then \( A \) contains an arc from \( v_p \) to \((h_r, M_r, C_r)\) for each \([v_p, M_r]\)-option \( C_r \), and to \((h_r, M_r, \emptyset)\) if there is no \([v_p, M_r]\)-option. We say that \( \sigma \) is a winning strategy (for the Captain) if \( G(\sigma) \) is acyclic. Otherwise, i.e., if \( G(\sigma) \) contains a cycle, then the Robber can avoid her/his capture forever.\(\square\)

**Example 5.2** Consider the two hypergraphs \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \) reported in Figure\(^4\) together with the strategy graph \( G(\sigma) \). The graph encodes a winning strategy \( \sigma \) for the Captain. From the initial configuration \((\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))\), the Captain activates all the cops in the hyperedge \( \{A, C, D, E, G\} \), so that the Robber has two available options, i.e., \( \{B\} \) and \( \{F\} \). In the former (resp., latter) case, the Captain activates all the cops in the hyperedge \( \{B, C\} \) (resp., \( \{E, F\} \)), so that the Robber has necessarily to occupy the node \( A \) (resp., \( G \)). Finally, the Captain activates the cops in \( \{A, B\} \) (resp., \( \{F, G\} \)) and captures the Robber.

\(^9\)It is easy to see that in such games, being the robber arbitrarily fast, what matters is not the precise node where the robber stands, but just the \([M]\)-component where (s)he is free to move.
Figure 7: The hypergraphs $H_1$ and $H_2$, plus the graph $G(\sigma)$ in Example 5.2.

Figure 8: A tree projection $H_\alpha$ for the pair in Example 5.2, plus the graph $G(\overline{\sigma})$.

Note that the strategy $\sigma$ is non-monotone, because the Robber is allowed to return on $A$ and $G$, after that these nodes have been previously occupied by the Captain in the first move.

In the above example, the hyperedge $\{A, C, D, E, G\}$ of $H_2$ "absorbs" the cycle in $H_1$, so that it is easily seen that there is a tree projection $H_\alpha$ of $H_1$ w.r.t. $H_2$ (see Figure 8). The fact that on this pair the Captain has a winning strategy is not by chance.

**Theorem 5.3** ([30]) There is a tree projection of $H_1$ w.r.t. $H_2$ if, and only if, there is a winning strategy in the Captain and Robber game played on $(H_1, H_2)$.

Recall that the winning strategy in Example 5.2 is not monotone. However, an important property of this game is that there is no incentive for the Captain to play a strategy that is not monotone.

**Theorem 5.4** (cf. [30]) In the Captain and Robber game played on the pair $(H_1, H_2)$, a winning strategy exists if, and only if, a monotone winning strategy exists.

Moreover, from any monotone winning strategy, a tree projection of $H_1$ w.r.t. $H_2$ can be computed in polynomial time.

**Example 5.5** Consider again the setting of Example 5.2 and the strategy graph $G(\overline{\sigma})$ shown in Figure 8. Note that the strategy $\overline{\sigma}$ is monotone, and in fact the moves of the Captain one-to-one correspond with the hyperedges in the tree projection $H_\alpha$. 

42
The crucial properties to establish Theorem 5.4 are next recalled, as they will be useful in our subsequent analysis too. Let \( \sigma \) be a strategy, and let \( v_p = (h_p, M_p, C_p) \) and \( v_r = (h_r, M_r, C_r) \) be two configurations in its domain such that \( \sigma(v_p) = (h_r, M_r) \) and \( C_r \) is a \([v_p, M_r]\)-option. Let \( \sigma(v_r) = (h_s, M_s) \) and define \( \text{ED}((M_r, C_r), M_s) = M_r \cap \text{Fr}(C_r) \setminus M_s \) (which is equivalent to \( \partial C_r \setminus M_s \), because \( C_r \) is an \([M_r]\)-component) as the escape-door of the Robber in \( v_r \) when attacked with \( M_s \). From [30], a move is monotone if, and only if, such an escape door is empty; in particular, \( \sigma(v_r) \) is non-monotone if (and only if) \( \text{ED}((M_r, C_r), M_s) \neq \emptyset \).

Let \( M'_r = M_r \setminus \text{ED}((M_r, C_r), M_s) \), let \( C'_r \) be the \([M'_r]\)-component with \( C_r \cup \text{ED}((M_r, C_r), M_s) \subseteq C'_r \), which exists since \( \text{ED}((M_r, C_r), M_s) \subseteq \text{Fr}(C_r) \) and \( M'_r \subseteq M_r \), and let \( v'_r = (h_r, M'_r, C'_r) \). Finally, consider the following strategy \( \sigma' \):

\[
\sigma'(h, M, C) = \begin{cases} (h_r, M'_r) & \text{if } (h, M, C) = (h_p, M_p, C_p) \\ \sigma(h, M, C) & \text{otherwise.} \end{cases}
\]

For such a state of the game, a number of technical properties have been proved in [30]. We summarize them in the following lemma.

**Lemma 5.6** ([30]) The following properties hold:

1. \( \text{ED}((M'_r, C'_r), M_s) = \emptyset \).
2. For each \([v_p, M_r]\)-option \( C \), either \( C \subseteq C'_r \) or \( C \) is a \([v_p, M'_r]\)-option.
3. For each \([v_p, M'_r]\)-option \( C' \neq C'_r \), \( C' \) is a \([v_p, M_r]\)-option.
4. A set \( C \) is a \([v_r, M_s]\)-option if, and only if, it is a \([v'_r, M_s]\)-option.
5. If \( \sigma \) is a winning strategy, then \( \sigma' \) is a winning strategy too.

### 5.2 Greedy Strategies

Since winning strategies correspond to tree projections, there is no efficient algorithm for their computation. Indeed, just recall that deciding the existence of a tree projection is not feasible in polynomial time, unless \( P = \text{NP} \) [26]. Our goal is then to focus on certain “greedy” strategies that are easy to compute. Intuitively, in greedy strategies it is required that all cops available at the current squad \( h_p \) and reachable by the Robber enter in action. If all of them are in action, then a new squad \( h_r \) is selected, again requiring that all the active cops, i.e., those in the frontier, enter in action.

**Definition 5.7** On the Captain and Robber game played on \((H_1, H_2)\), a strategy \( \sigma \) is greedy if, for any configuration \( v_p = (h_p, M_p, C_p) \) in the domain of \( \sigma \), the next position \( \sigma(v_p) = (h_r, M_r) \) is such that \( M_r = h_r \cap \text{Fr}(C_p) \), where \( h_r = h_p \) if \( h_p \cap C_p \neq \emptyset \), and \( h_r \) is any squad in edges \((H_2)\) if \( h_p \cap C_p = \emptyset \). \( \square \)
Given such a greedy way to select cops at each step, observe that the former case \((h_p \cap C_p \neq \emptyset)\) may only occur if the Robber is able to come back to some position previously controlled by the Captain. Greedy winning strategies are indeed non-monotone in general, and for some pair of hypergraphs it is possible that there is no monotone winning greedy strategy, although monotone winning strategies (non-greedy) exist.

**Example 5.8** Consider again the hypergraphs \(\mathcal{H}_1\) and \(\mathcal{H}_2\) shown in Figure 4 and recall that the strategy graph of a monotone winning strategy \(\tilde{\sigma}\) is depicted in Figure 5. However, there is no monotone greedy strategy in this case. Indeed, if at the beginning of the game the Captain asks the squad \(\{A, C, D, E, G\}\) to enter in action and the Robber goes on \(B\), then in the next move the Robber is forced to lose the control on \(A\) in order to move on \(\{C, B\}\) and eventually win via \(\{B, A\}\)—see again Figure 7. On the other hand, if the attack of the Captain starts on either side, say on the left branch, the Captain has then to attack the component that includes the triangle and the other branch. At this point, the only available greedy choice is use the big squad and hence to employ cops \(\{C, D, E, G\}\). However, as in the previous case, \(G\) will be later (necessarily) left free to the Robber, in order to win the game.

We now show that, differently from arbitrary strategies, the existence of greedy winning strategies can be decided in polynomial time. To establish the result, a useful technical property is that greedy strategies can only involve a polynomial number of configurations. Let us denote by \(\text{MaxGreedyStrat}(\mathcal{H}_1, \mathcal{H}_2)\) the maximum domain cardinality over any greedy strategy in the Robber and Captain game on a pair \((\mathcal{H}_1, \mathcal{H}_2)\).

**Lemma 5.9** Let \((\mathcal{H}_1, \mathcal{H}_2)\) be a pair of hypergraphs. Then, \(\text{MaxGreedyStrat}(\mathcal{H}_1, \mathcal{H}_2)\) is at most \(|\text{edges}(\mathcal{H}_2)| \times \text{nodes}(\mathcal{H}_1)\)\(|\text{edges}(\mathcal{H}_2)| \times \text{nodes}(\mathcal{H}_1)| + 1| + 1\).

**Proof.** Let \(\sigma\) be a greedy strategy, and let \(v_p = (h_p, M_p, C_p)\) be a configuration in its domain. Note that the only configuration where \(h_p = M_p = \emptyset\) is the starting configuration \((\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))\), which is taken into account by the final “+1” in the statement. Therefore, we next assume \(M_p \neq \emptyset\).

Consider the case where \(h_p \cap C_p = \emptyset\). In this case, a new squad \(h_r \in \text{edges}(\mathcal{H}_2)\) is chosen by the Captain according to \(\sigma\). Since \(C_p\) is an \([M_p]\)-component and thus \(\partial C_p \subseteq M_p \subseteq h_p\), we get that this case occurs only if \(C_p\) is actually an \([h_p]\)-component, too. Such a component is uniquely identified by any pair of the form \((h_p, X_p)\) such that \(X_p \in \text{nodes}(\mathcal{H}_1)\) is a representative of the component (e.g., the node in \(C_p\) having the smallest position according to any fixed ordering over the nodes). It follows that the new set of cops \(M_r = h_r \cap \text{Fr}(C_i)\) is uniquely determined by \(h_r, C_p\), and thus may be identified through a triple \((h_r, h_p, X_p)\). Thus, the maximum number of such sets \(M_r\) of cops is \(|\text{edges}(\mathcal{H}_2)|^2 \times \text{nodes}(\mathcal{H}_1)|\). Moreover, the possible configurations \((h_r, M_r, C_r)\) following \((h_p, M_p, C_p)\) in the game where the Captain plays according to \(\sigma\) are identified by quadruples of the form \((h_r, h_p, X_p, X_r)\), where \(h_r\) is used both to identify itself and to determine the set \(M_r\) together with \(h_p\) and \(X_p\), and
Boolean function GreedyWinningStrategy(h_p, M_p, C_p, i);
/* (h_p, M_p, C_p) is an extended configuration over (H_1, H_2).
 i ≥ 0 is a natural number */

1) if i > MaxGreedyStrat(H_1, H_2), then return False;
2) if h_p ∩ C_p ≠ ∅, then let h_r = h_p;
   else guess a hyperedge h_r ∈ edges(H_2);
3) let M_r = h_r ∩ Fr(C_p);
4) for each [(h_p, M_p, C_p), M_r]-option C_r do
   if not GreedyWinningStrategy(h_r, M_r, C_r, i + 1), then return False;
5) return True;

Figure 9: GreedyWinningStrategy.

where X_r is a representative of the [M_r]-component. In fact, if there is no [v_p, M_p]-option, then X_r is a distinguished element not in \( \text{nodes}(H_1) \) (or some element in \( M_r \) occupied by some cop) meaning that the only configuration following \( (h_p, M_p, C_p) \) is \( (h_r, M_r, ∅) \) where the Robber is captured. Overall, the maximum number of such configurations is \( |\text{edges}(H_2)|^2 \times |\text{nodes}(H_1)|^2 \).

Finally, consider the case where \( h_p \cap C_p ≠ ∅ \). In this case, \( M_r = h_p \cap Fr(C_p) \). Since \( C_p \) is an [M_p]-component, \( \partial C_p \subseteq M_p \subseteq h_p \). It follows that the new nodes from Fr(\( C_p \)) to be included in \( M_r \) belong to \( C_p \), that is, we may also write \( M_r = M_p \cup (h_p \cap C_p) \). Note that no configuration of the game following this one can be of this type. Indeed, every [M_r]-component \( C_r \) where the Robber may go from \( C_p \) will be a subset of \( C_p \) (because \( \partial C_p \subseteq M_p \subseteq M_r \subseteq h_p \)), and will have intersections with \( h_p \). As a further consequence, such a \( C_r \) must be an [h_p]-component. By contradiction, if there is some node \( X_p \in C_r \subseteq C_p \) that is [M_r]-connected to some \( X_r \) in \( h_p \\setminus M_r \), then \( X_p \) is also [M_p]-connected to \( X_r \). However, this is impossible because \( X_p \) is also in \( C_p \) and hence \( X_r \) would be in \( C_p \), too, and hence in \( h_p \cap C_p \) and in \( M_r \), by construction. Therefore, the possible configurations \( (h_p, M_r, C_r) \) following \( (h_p, M_p, C_p) \) in the game where the Captain plays according to \( σ \) are identified by pairs of the form \( (h_p, X_p) \), where \( X_p ∈ \text{nodes}(H_1) \) is the representative of the [h_p]-component \( C_r \) (and where \( M_r \) is computed from them). As above, if there is no [v_p, M_p]-option, then \( X_p \) is a distinguished element witnessing that the configuration is a capture configuration of the form \( (h_p, M_r, ∅) \). Overall, the maximum number of such configurations is \( |\text{edges}(H_2)| \times |\text{nodes}(H_1)| \).

To see that the existence of a winning greedy strategy is decidable in polynomial time, consider the GreedyWinningStrategy algorithm illustrated in Figure 9 which receives as input a configuration \( (h_p, M_p, C_p) \) for the Robber.
and Captain game, plus a “level” $i$. Note that this algorithm is a high-level specification of an alternating Turing machine, say $M$. After the first step, where we check that the number of recursive calls has not exceeded the number of all distinct configurations, the algorithm suddenly evidences its non-deterministic nature. Indeed, it guesses a hyperedge $h_r$ corresponding to the next move of the Captain (existential step of $M$). Eventually, it returns $\text{True}$ if, and only if, the recursive calls $\text{GreedyWinningStrategy}(h_r, M_r, C, i + 1)$ with $M_r = h_r \cap \text{Fr}(C_p)$ succeed on each $[(h_p, M_p, C_p), M_r]$-option (universal step of $M_G$).

**Theorem 5.10** Deciding the existence of a greedy winning strategy in the Robber and Captain game is feasible in polynomial time.

**Proof.** Let $(H_1, H_2)$ be a pair of hypergraphs, and consider the execution of the Boolean function $\text{GreedyWinningStrategy}$ on input the starting configuration $([\emptyset, \emptyset, \text{nodes}(H_1), 0])$. Due to its non-deterministic nature, it is easily seen that, by getting rid of step (1), it returns $\text{True}$ if, and only if, the Captain has a greedy winning strategy in the game played on $(H_1, H_2)$ (which we assume to be “visible” by the function at every call, to avoid a longer signature). Moreover, we claim that the check performed at step (1) cannot lead to a wrong $\text{False}$ output. Indeed, just observe that the number of recursive calls is bounded by the number of all distinct configurations, which is $\text{MaxGreedyStrat}(H_1, H_2)$ at most, by Lemma 5.9. Therefore, if the recursion level $i$ exceeds this threshold, then we can safely answer $\text{False}$.

Let us now focus on the running time. We have already observed that $\text{GreedyWinningStrategy}$ may be implemented on an alternating Turing machine $M_G$, whose existential steps correspond to the guess statements at step 2, while universal steps are used for checking that the conditions at step 4 are satisfied by all the relevant components. In addition, by indexing the various data structures and by referring each component via one point contained in it (selected through any fixed criterium), the machine can be implemented to use logarithmic many bits on its worktape. For instance, recall from the proof of Lemma 5.9 that every configuration is identified by at most four elements of the form $(h_p, h_r, X_p, X_r)$ with $h_p, h_r \in \text{edges}(H_2)$ and $X_p, X_r \in \text{nodes}(H_1)$. Therefore, any configuration may be encoded by (at most) four indexes whose maximum size is $\log \max(|\text{edges}(H_2)|, |\text{nodes}(H_1)|)$. Moreover, the check at step (1) ensures that the length of each branch of the computation tree of $M_G$ is finite, and actually bounded by a polynomial in the size of the input. For the sake of completeness, observe that all subtasks in the function, such as computing connected components and the like, are easily implementable in nondeterministic logspace, so that such tasks just correspond to further (polynomially-bounded) branches of the computation tree of $M_G$. Thus, $\text{GreedyWinningStrategy}$ may be implemented in a log-space alternating Turing machine, which immediately entails the result, because Alternating Logspace is equal to Polynomial Time [11].
It is well known that an alternating Turing machine \( M_G \) can be simulated by a standard machine in polynomial time. First, compute the polynomially-many possible instant descriptions (IDs) of the machine, and build a graph representing the possible connections between any pair of IDs, according to its transition relation. Then, evaluate this graph along some topological ordering as follows. Mark all IDs without outcoming arcs associated with final accepting states; then mark all IDs associated with existential states having a marked successor, or associated with universal states, and whose successors are all marked. Then, the machine \( M_G \) accepts its input if, and only if, the starting ID is marked. Moreover, the subgraph induced by the marked nodes encodes its accepting computations.

Moreover, from such a marked graph it is straightforward to compute the strategy graph of a greedy winning strategy, because IDs associated with (children of) existential states encode the possible choices of the Captain. Just visit the graph starting from the initial configuration, but for each ID associated with an existential state, select one child to be visited arbitrarily (all choices are marked and hence accepting).

**Corollary 5.11** The strategy graph of a greedy winning strategy (if any) in the Robber and Captain game is computable in polynomial time.

### 5.3 Greedy Tree Projections and Larger Islands of Tractability

From the previous sections (see Theorem 5.4 and Example 5.8), we know that monotone winning strategies for the Captain in the game over \((\mathcal{H}_1, \mathcal{H}_2)\) are associated with tree projections of \(\mathcal{H}_1\) w.r.t. \(\mathcal{H}_2\), and that in some cases it is possible that there is no monotone winning greedy strategy, although monotone winning strategies (non-greedy) exist. In this section, we show that from any (possibly non-monotone) greedy winning strategy a tree projection can be still computed in polynomial time. The key fact here is that any non-monotone greedy strategy can be converted into a monotone one, though not a greedy one in general.

To show the result, it is useful to consider a special form of strategies that we call *nice* (for they remind the notion of nice tree decompositions of graphs), where at every configuration the Captain first removes those cops that are no longer in the frontier.

Formally, \( \sigma \) is a *nice* strategy if \( \sigma(h_p, M_p, C_p) = (h_p, \partial C_p) \), whenever \( \partial C_p \subset M_p \). Because such inactive cops play no role in the Robber and Captain game, a winning nice strategy exists if (and only if) there exists a winning strategy, and the same holds for greedy strategies. Just note that restricting the cops

---

\(^{10}\)For the sake of completeness note that, by using these ideas, one might also provide a direct dynamic programming algorithm to compute a strategy graph by using a bipartite graph representing all possible configurations and positions of the Robber and Captain game. However, we find the non-deterministic function \texttt{GreedyWinningStrategy} more elegant and easy to present.
to the border of $C_p$ is a legal choice in greedy strategies (it corresponds to the selection of the same squad $h_r = h_p$ before attacking the robber in the component $C_p$ with some further squad). Clearly enough, such a nice strategy can be computed in polynomial time from any given strategy. Also, if desired, the above polynomial time algorithm for computing a greedy strategy may be easily adapted to compute directly a winning nice greedy strategy (if any).

Example 5.12 Consider again the setting discussed in Example 5.2 and illustrated in Figure 7. Note that the strategy $\sigma$ is not nice. Indeed, Figure 10 reports the strategy graph associated with a strategy $\sigma_n$ that is nice and that is obtained from $\sigma$ by just explicitly adding the configurations where the Captain has to remove the cops that are no longer in the frontier. ◐

The reason for introducing these nice strategies is that they admit a more compact representation. First, given any configuration $(h_p, M_p, C_p)$ and a Captain’s choice $M_r$, the $[(h_p, M_p, C_p), M_r]$-options for the Robber are actually determined by $C_p$ and $M_r$ only, because $\partial C_p$ is computable from $C_p$. Therefore, we use hereafter the simplified notation $[C_p, M_r]$-option to refer to this set of $[M_r]$-components. Moreover, in place of the strategy graph, we can use a component graph, defined as follows.

Definition 5.13 Let $(\mathcal{H}_1, \mathcal{H}_2)$ be a pair of hypergraphs. Let $G = (N, A)$ be a directed graph whose nodes are pairs of the form $(h_p, C_p)$, where $h_p \in \text{edges}(\mathcal{H}_2)$,
and \( C_p \) is either the emptyset or a \([\partial C_p]\)-component of \( H_1 \) such that \( \partial C_p \subseteq h_p \). Then, we say that \( G \) is a component graph if it meets the following conditions:

1. There is a root node \((\emptyset, \text{nodes}(H_1)) \in N\) that is the only node without incoming arcs.

2. Each node \((h_p, C_p) \in N\), with \( C_p \neq \emptyset \), has outgoing arcs to \( m \geq 0 \) nodes \((h_r, C_1), \ldots, (h_r, C_m)\) such that, if \( M_r \) is the set \( \bigcup_{j=1}^m \partial C_j \cup (C_p \setminus \bigcup_{j=1}^m C_j) \), it holds that \( M_r \subseteq h_r \) and the \([C_p, M_r]\)-options are the components \( C_1, \ldots, C_m \).

3. Each node \((h_p, C_p) \in N\) has an outgoing arc to \((h_r, \emptyset)\) if \( C_p \subseteq h_r \).  

Note that every nice strategy \( \sigma \) is encoded by the component graph \( G_c(\sigma) = (\emptyset, N, A) \) defined as follows. There is a node \((h_p, C_p)\) in \( N \) if there is a configuration \((h_p, \partial C_p, C_p)\) in the domain of \( \sigma \) (resp., a capture configuration \((h_p, \partial C_p, \emptyset)\) induced by \( \sigma \)). There is an arc in \( A \) from a node \((h_p, C_p)\) to a node \((h_r, C_r)\) if there is an arc from \((h_p, M_p, C_p)\) to \((h_r, M_r, C_r)\) in the strategy graph \( G(\sigma) \).

No more nodes and arcs occur in \( N \) and \( A \), respectively. For instance, the graph depicted on the bottom part of Figure 7 is the component graph associated with the nice strategy \( \sigma_n \) of Example 5.12.

Conversely, any component graph \( G \) encodes a nice strategy \( \sigma_G \), via the following procedure. Associate the root \((\emptyset, \text{nodes}(H_1))\) with the initial configuration \((\emptyset, \emptyset, \text{nodes}(H_1))\). Inductively, assume that a node \((h_p, C_p)\) is associated with a configuration \((h_p, M_p, C_p)\), and that \((h_r, C_1), \ldots, (h_r, C_m)\) are the labels of the nodes having an incoming arc from \((h_p, M_p, C_p)\). Let \( M_r = \bigcup_{j=1}^m \partial C_j \cup (C_p \setminus \bigcup_{j=1}^m C_j) \), with \( M_r \subseteq h_r \). Then, define \( \sigma_G(h_p, M_p, C_p) = (h_r, M_r) \), and define \( \sigma_G(h_r, M_r, C_j) = (h_r, \partial C_j) \), with \( j \in \{1, \ldots, m\} \), in the case where \( \partial C_j \subseteq M_r \).

**Theorem 5.14** A tree projection of \( H_1 \) w.r.t. \( H_2 \) can be computed in polynomial time if the Captain has a greedy winning strategy on \((H_1, H_2)\).

**Proof.** By Theorem 5.10 we can decide in polynomial time whether a winning greedy strategy for the Captain in the game played on \((H_1, H_2)\) exists or not. In the negative case, we are done. Otherwise, compute in polynomial time a winning nice greedy strategy \( \sigma \) (or turn a given strategy into a nice one), and compute its component graph \( G_c(\sigma) \). Make a copy \( G' = (N', A') \) of \( G_c(\sigma) \), and note that \( G' \) is a directed acyclic graph, because it encodes a winning strategy.

Let \( \overline{N} = v_1, \ldots, v_{|N'|} \) be the topologically ordered sequence of the nodes of \( G' \), where the nodes without outgoing arcs, called leaves, are in the first positions, and the node without incoming arcs, its root, is at the last position. Note that leaves correspond to capture configurations for the robber, while the root \( v_{|N'|} = (\emptyset, \text{nodes}(H_1)) \) is associated with the starting configuration \((\emptyset, \emptyset, \text{nodes}(H_1))\) of the game. Moreover, if \((v, v') \in A'\), the node \( v \) is said to be a parent of \( v' \), while \( v' \) is said to be a child of \( v \). Then, modify the graph \( G' \), by navigating the sequence \( \overline{N} \) using an index \( j \).

49
Starting with $j = 1$, while $j < |N'|$, consider the current node $v_j$ in the sequence, associated with a configuration $(h_j, M_j, C'_j)$ (initially, the first leaf) in the domain of $\sigma^{C'}$. If every child of $v_j$ is labeled by some $(h'', C'')$ with $C'' \subseteq C_j$, then let index $j := j + 1$ and continue the “while” loop, or stop and output the current graph $G'$ if $v_j$ is the root. Otherwise, let $v_s$ be a child of $v_j$ labeled by $(h_s, C_s) \in N'$ such that $C_s \not\subseteq C_j$, and associated with the configuration $(h_s, M_s, C_s)$. That is, $\sigma^{C'}(h_j, M_j, C_j) = (h_s, M_s)$ is a non-monotone move. Then, take any parent $v_p$ of $v_j$, and let $(h_p, M_p, C_p)$ the configuration associated with $v_p$ (whose label is thus $(h_p, C_p)$). Modify the graph so that $\sigma^{C'}(h_p, M_p, C_p) = (h_j, M'_j)$, where $M'_j = M_j \setminus \text{ED}(v_j, M_s)$. In particular, let $C'_j$ be the $[M'_j]$-component that properly includes $C_j$, and for which thus $C_j \cup C'_j$ is $[M_j \cap M'_j]$-connected. Then, the modified component graph will also encode the choice $\sigma^{C'}(h_j, M'_j, C'_j) = (h_j, \partial C'_j)$ if $\partial C'_j \subset M'_j$, and $\sigma^{C'}(h_j, \partial C'_j, C'_j) = (h_s, M_s)$. The transformation of the graph is as follows:

(i) Add a node $v'_j$ labeled by $(h_j, C'_j)$ to $N'$ and to the sequence $\vec{N}$ in the position before $v_j$, and add to $A'$ an arc from $v'_j$ to each child of $v_j$, i.e., to nodes labeled by $(h_s, C'')$, for each $[C'_j, M_s]$-option $C''$.

(ii) Remove from $A'$ all outgoing arcs of $v_p$ to nodes whose labels do not contain $[C_p, M'_j]$-options (in particular, the arc towards $v_j$ is removed).

(iii) Add to $A'$ an arc from $v_p$ to $v'_j$.

(iv) Remove from $N'$ any node different from the root which is left without incoming arcs, and continue the “while” loop considering again node $v_j$, or the next available node in $\vec{N}$ if $v_j$ has been removed by $N'$.

**Example 5.15** The application of the above procedure to the nice strategy $\sigma_n$ discussed in Example 5.12 is illustrated in Figure 11. Note that two non-monotone moves are removed in total. Note that, at the end of the transformation, we get a component graph encoding precisely the monotone strategy $\bar{\sigma}$, whose strategy graph has been illustrated in Figure 8.

First observe that every iteration of the loop at step 1 above, precisely implements on the graph $G'$ the transformation (of the non-monotone strategy encoded by $G'$) described by Expression (1), and whose properties are described by Lemma 5.6. In more detail, with these properties in mind, by executing steps (i)–(iii) we replace the Captain’s choice $(h_j, M_j)$ at $(h_p, M_p, C_p)$ by the new choice $(h_j, M'_j)$, and we get the following situation: (a) Because of the new choice $M'_j$, only one new $[C_p, M'_j]$-option is available to the robber, that is, the $[M'_j]$-component $C'_j$ properly including the $[M_j]$-component $C_j$. As a consequence, at step (i) the one node $v'_j$ corresponding to this component is added to $N'$. (b) The $[C'_j, M_s]$-options are the same as the $[C_j, M_s]$-options, so that the outgoing arcs of $v'_j$ will be the same as the node $v_j$. That is, we keep the same winning strategy as before, as the Robber’s options after the
Captain’s choice $M_s$ are the same as before (and hence the Captain knows how to successfully attack them). (c) The set of $[C_p, M_j]$-options, with the exception of the new $C_j'$, are a subset of the $[C_p, M_j]$-options. In fact, some components may collapse after the new choice of the Captain. Then, at step (iv), we remove the nodes associated with $[C_p, M_j]$-options that are now left without incoming arcs. For instance, it is possible that we delete $v_j$ if $v_p$ was its only parent, or it is possible that we delete some nodes associated with collapsed components. Note that the new graph $G'$ obtained from these steps is still a component graph, hence it encodes a (new) nice strategy $\sigma_{G'}$.

Therefore, Lemma 5.6 entails that, after each iteration and thus after the entire procedure, the strategy $\sigma_{G'}$ is a winning strategy. We claim that it is actually a monotone winning strategy, by a simple inductive argument: if $v_j$ is the current node, after the execution of steps (i)–(iv), $\sigma_{G'}$ is a monotone winning strategy for the game starting at the configuration $v_j$. Then, the claim follows because, for $j = |N'|$, it means that $\sigma_{G'}$ is a monotone winning strategy for the whole game starting at the root. The base case is when the algorithm starts at $j = 1$, and hence the statement holds because the first position in $\overrightarrow{N}$ is occupied by some leaf, which is a capture configuration of the winning strategy. Now assume that the statement holds for $j - 1$, and consider the execution of the above procedure on node $v_j$. Note that the proposed transformation deals with just one (possibly new) component $C_j'$ instead of the strictly smaller $C_j$; everything else in the strategy does not change, in particular no node preceding $v_j$ in the topological order is affected by the transformation. Then, the monotonicity of the strategy on the game starting at $v_j$ immediately follows from the induction hypothesis and from Lemma 5.6(1), which says that $ED(v'_j, M_s) = \emptyset$ and hence...
that this move is monotone, so that $C'' \subseteq C''_j$, for each $[C'_j, M_{i+1}]$-option $C''$.

Because each iteration is feasible in polynomial time, it just remains to show that the whole procedure requires at most polynomially many iterations. To this end, note that whenever some node $v_j$ encodes a non-monotone move, one node $v'_j$ is added to $N'$ for each parent $v_p$ of $v_j$. Indeed, the node $v_j$ is considered again after the first iteration where it was evaluated, if it still has incoming arcs (see step (iv)). However, after steps (i)–(iv), $\sigma_{G'}$ is a monotone winning strategy for the game starting at the new configuration $v'_j$. Therefore, no new node will be subject to further transformations in subsequent iterations along the given topological ordering of $N'$. It follows that the number of iterations of the described procedure is bounded by $\text{nodes}(G_c(\sigma)) \times \text{MaxIn}$, where MaxIn is the largest in-degree over the nodes of $G_c(\sigma)$. Thus, the number of iterations is bounded by a polynomial in the size of the strategy graph of the greedy winning strategy, which is in its turn polynomial in the size of $(H_1, H_2)$.

Finally, from the monotone winning strategy $\sigma_{G'}$ encoded by the output $G'$ of the above procedure, a tree projection $H_a$ of $(H_1, H_2)$ is immediately available. Just define $\text{nodes}(H_a) = \text{nodes}(H_1)$ and $\text{edges}(H_a) = \{M \mid \sigma_{G'}(v) = (h, M) \text{ for some configuration } v \text{ in the domain of } \sigma_{G'}\}$. See [30], for more detail about such a relationship between monotone strategies and tree projections.

With the above result in place, let $C_{gtp}$ denote the class of all pairs $(Q, V)$ such that there exists a greedy winning strategy $\sigma$ for the Captain in the game $\text{R \& C}(H_Q, H_V)$. As shown in the proof of Theorem 5.14, based on $\sigma$ a tree projection of $H_Q$ w.r.t. $H_V$, which we call greedy tree projection, can be computed in polynomial time. Therefore, the following is immediately established.

**Corollary 5.16** $C_{gtp}$ is an island of tractability.

### 5.4 Captain vs Marshal

A related class of tractable pairs has been defined in [4] in terms of the Robber and Marshal game played by one Marshal and the Robber on the hypergraphs $(H_1, H_2)$. This game has been originally defined on a single hypergraph to characterize hypertree decompositions [25], and its natural extension to pairs of hypergraphs has been defined and studied in [4]. The game is as follows. The Marshal may control one hyperedge of $H_2$, at each step. The Robber stands on a node and can run at great speed along hyperedges of $H_1$; however, (s)he is not permitted to run through a node that is controlled by the Marshal. Thus, a configuration is a pair $(h, C)$, where $h$ is the hyperedge controlled by the Marshal, and $C$ is an $[h]$-component where the Robber stands. Let $(h_p, C_p)$ be a configuration. This is a capture configuration, where the Marshal wins, if $C_p \subseteq h_p$. Otherwise, the Marshal moves to another hyperedge $h_r \in \text{edges}(H_2)$; while (s)he moves, the Robber may run through those nodes that are left by the Marshal or not yet occupied. Thus, the Robber selects an $[h_r]$-component $C_r$ such that $C_r \cup C_p$ is $[h_p \cap h_r]$-connected. We say that the Marshal has a
**winning strategy** if, starting from the initial configuration \((\emptyset, \mathcal{N})\), (s)he may end up the game in a capture position, no matter of the Robber’s moves. A winning strategy is **monotone** if the Marshal may monotonically shrink the set of nodes where the Robber stands.

Because only nodes in the frontier are actually used at each step in the monotone Robber and Marshal game, the monotone variants of the above two games clearly define the same hypergraph properties.

**Fact 5.17** The following are equivalent:

1. There is a monotone winning strategy for the Marshal in the Robber and Marshal game on \((\mathcal{H}_1, \mathcal{H}_2)\).
2. There is a monotone winning greedy-strategy for the Captain in the Robber and Captain game on \((\mathcal{H}_1, \mathcal{H}_2)\).

Let \(C_{rm}\) denote the class of all pairs \((Q, \mathcal{V})\) such that there exists a monotone winning strategy for the Marshal on \((\mathcal{H}_Q, \mathcal{H}_V)\). From the results in [4, 3], \(C_{rm}\) is an island of tractability as well. However, the set of tractable instances identified by greedy winning strategies in the Robber and Captain game properly includes this class. The reason is that greedy winning strategies are allowed to be non-monotone.

**Theorem 5.18** \(C_{rm} \subseteq C_{gtp}\).

**Proof.** Because greedy strategies are not required to be monotone, \(C_{rm} \subseteq C_{gtp}\) follows from Fact 5.17. For the proper inclusion, just consider again Example 5.8. The pair of hypergraphs shown in Figure 7 is such that the Marshal has no monotone winning strategy, while the Captain has a (non-monotone) winning greedy strategy.

For completeness, recall that the non-monotone variant of the Marshal and Robber game is instead too powerful to be useful. Indeed, there are pairs of hypergraphs where the Marshal has a non-monotone winning strategy but no tree projection exists. We refer the interested reader to [4] for more detail about the monotonicity gap in the Robber and Marshal game, and to [32] for a measure of distance between non-monotone strategies in the Robber and Marshal game and tree projections.

### 5.5 Greedy Decomposition Methods

The tractability result about the general case of greedy tree projections can be immediately applied to every structural decomposition method, in order to get new tractable variants of these methods.

Recall from Definition 4.1 that a structural decomposition method \(\text{DM}\) is a pair of polynomial-time computable functions \(v_{-\text{DM}}\) and \(d_{-\text{DM}}\) that, given a

\[\text{This example is in fact inspired by a similar simpler pair of hypergraphs where no monotone strategy for the Marshal exists, described in [4].}\]
conjunctive query $Q$ and a database $DB'$, compute a view system $V = v$-$DM(Q)$ and a database $DB'' = d$-$DM(Q, DB')$ over the vocabulary of $V$ that may be used to answer $Q$ on $DB'$. In particular, the decompositions of $Q$ according to $DM$ are tree projections of $H_Q$ w.r.t. $H_V$. Then, it is natural to consider the greedy variant of any structural decomposition method $DM$, denoted by $greedy$-$DM$, whose associated decompositions are the $greedy$ tree projections of $H_Q$ w.r.t. $H_V$.

From Corollary 5.16, every decomposition method, possibly an intractable one such as the generalized hypertree decomposition method, defines an island of tractability by means of its greedy variant.

**Fact 5.19** Let $DM$ be a structural decomposition method and let $greedy$-$DM$ be its greedy variant. Then, the class of all queries having a greedy-$DM$ decomposition is recognizable in polynomial time, and every query in the class may be evaluated in polynomial time over any given database.

For a notable example, consider the method based on generalized hypertree decompositions. Let $k \geq 1$. Recall that the width-$k$ generalized hypertree decompositions of a query $Q$ are the tree projections of $(H_Q, H^k_Q)$, as the view set $v$-$hw_k(Q)$ contains one distinct view over each set of variables that can be covered by at most $k$ query-atoms. Then, the width-$k$ greedy hypertree-decompositions (we omit “generalized”, for short) of $Q$ are the greedy tree projections of $(H_Q, H^k_Q)$. Accordingly, the greedy (generalized) hypertree-width of $Q$, denoted by $gr$-$hw$, is the smallest $k$ such that $Q$ has a greedy hypertree decomposition. In fact, this greedy variant provides a new tractable approximation of the (intractable) notion of generalized hypertree decomposition, which is better than (standard) hypertree decompositions.

**Fact 5.20** For any query $Q$, $ghw(Q) \leq gr$-$hw(Q) \leq hw(Q)$ holds. Moreover, there are queries $Q$ for which $gr$-$hw(Q) < hw(Q)$, even for $gr$-$hw(Q) = 2$.

**Proof.** The first relationship is immediate: in the first inequality we use the fact that greedy hypertree decompositions are a special case of generalized hypertree decompositions, while the second inequality holds because the notion of hypertree decomposition is characterized by the monotone Robber and Marshals...
game, played on $\mathcal{H}_Q$ by a Robber and $k$ Marshals [25]. This game is equivalent to play the monotone game with one Marshal on the pair of hypergraphs $(\mathcal{H}_Q, \mathcal{H}_Q^k)$, which is the same as playing the monotone Robber and Captain game.

For the strict upper bound $gr-hw(Q) < hw(Q)$, consider the query $Q_0$, taken from [14, 26], whose hypergraph $\mathcal{H}_{Q_0}$ is depicted in the left part of Figure 12. For this query, it is shown in [26] that $hw(Q_0) = 3$ and $ghw(Q_0) = 2$. However, $gr-hw(Q_0) = 2$ holds. Indeed, there is a winning greedy strategy for the Captain in the game played on $(\mathcal{H}_{Q_0}, \mathcal{H}_{Q_0}^2)$, as shown in the central part of Figure 12 and thus there exists a greedy tree projection of $\mathcal{H}_{Q_0}$ w.r.t. $\mathcal{H}_{Q_0}^2$. In the figure, the set of selected cops at each step is underlined in such a way that the reader may identify the original pair of hyperedges from $\mathcal{H}_{Q_0}$ that forms the chosen squad in $\mathcal{H}_{Q_0}^2$. Note that the strategy is non-monotone, as it is witnessed by the right branch where the Robber can return on the node $B$. However, by using the construction in Theorem 5.14 it can be turned into a monotone (while not greedy) one, by removing the escape door $B$ in the first move of the Captain (see the right part of the figure). From the monotone strategy, we immediately get the desired tree projection. □

More general examples are given by the subedge-based decomposition methods, defined in [26]. Recall that a subedge-method $\text{DM}$ is based on a function $f$ associating with each integer $k \geq 1$ and each hypergraph $\mathcal{H}_Q = (V, E)$ of some query $Q$ a set $f(\mathcal{H}_Q, k)$ of subedges of $\mathcal{H}_Q$, that is, a set of subsets of hyperedges in $E$. Moreover, the set of width-$k$ $\text{DM}$-decompositions of $Q$ can be obtained as follows: (1) obtain a hypertree decomposition $\mathcal{H}_d = (V, E \cup f(\mathcal{H}, k))$, and (2) convert $\mathcal{H}_d$ into a generalized hypertree decomposition of $\mathcal{H}_Q$ by replacing each subedge $h \in f(\mathcal{H}_Q, k) \setminus E$ occurring in $\mathcal{H}_d$ by some hyperedge $h' \in E$ such that $h \subseteq h'$ (which exists because $h$ is a subedge).

Because such a method is based on width-$k$ hypertree decompositions, in the tree projection framework it can be recast as follows. A width-$k$ $\text{DM}$-decomposition is any tree decomposition of $\mathcal{H}_Q$ w.r.t. $\mathcal{H}_Q^k$ associated with some monotone winning strategy of the Robber and Marshal game on this pair of hypergraphs. On the other hand, according to its greedy variant greedy-$\text{DM}$, the width-$k$ decompositions are the greedy tree projections of $\mathcal{H}_Q$ w.r.t. $\mathcal{H}_Q^k$. It follows that the greedy variant of this method is more powerful, in general.

**Fact 5.21** Let $\text{DM}$ be any subedge-based decomposition method. Let $k \geq 1$ and let $Q$ be a query. Then, a width-$k$ $\text{DM}$-decomposition of $Q$ exists only if a width-$k$ greedy-$\text{DM}$-decomposition of $Q$ exists. The converse does not hold, in general.

**Proof.** The first entailment follows from Theorem 5.18. The fact that the converse does not hold in general, follows from Fact 5.20 because the hypertree decomposition method is a subedge-based method (based on the function $f(\mathcal{H}_Q, k) = \emptyset$). □

This is a remarkable result, as in [26] some examples of subedge-based decomposition methods, such as the component hypertree decompositions, are shown to
generalize most previous proposals of tractable structural decomposition methods, such as hypertree and spread-cut decompositions (in fact, all of them, but the approximation of fractional hypertree decomposition, later introduced in [40]). From Fact 5.21, their greedy variants are even more powerful.

6 Tractability of Tree Projections over Small Arity Structures

In this (light) section, we consider the case of relational structures having small arity, which is a relevant special case in real-world applications.

In fact, observe that any variable that is not involved in any join operation in a conjunctive query (that is, any variable that occurs in one atom only) is irrelevant and may be projected out in a preprocessing phase. It follows that the effective arity to be considered in our structural techniques is actually determined by the largest number of variables that any atom has in common with other atoms (i.e., those variables involved in join operations), independently of the arity of the relations in the original database schema. This number is often small, in practice.

Therefore, it is interesting to investigate whether the general problem of computing a tree projection of a pair of hypergraphs is any easier in the case of small arity structures (for the sake of presentation, we just consider here the standard structure arity, leaving to the interested reader the straightforward extension to the above mentioned “effective arity”). We next show that the problem is indeed in polynomial-time for bounded-arity structures, and it is moreover fixed-parameter tractable (FPT), if the arity is used as a parameter of the problem. This is not difficult to prove, but it was never stated before (as far as we know), and we believe it is important to pinpoint this tractability result.

Recall that a problem is FPT if there is an algorithm that solves the problem in fixed-parameter polynomial-time, that is, with a cost $f(k)O(n^{O(1)})$, for some computable function $f$ that is applied to the parameter $k$ only. In other words, this algorithm not only runs in polynomial time if $k$ is bounded by a fixed number, but it also exhibits a “nice” dependency on the parameter, because $k$ is not in the exponent of the input size $n$. Let $p$-TP be the problem of computing a tree projection of $H_Q$ w.r.t. $H_V$, for a given pair $(Q, V)$, parameterized by the maximum arity of the relations occurring in $(Q, V)$.

Theorem 6.1 The problem $p$-TP is fixed-parameter tractable.

Proof. Let $(Q, V)$ be an input pair for $p$-TP, let $(H_Q, H_V)$ be the pair of associated hypergraphs, and let $k$ be the parameter.

12In fact, it is easy to further generalize this line of reasoning, by considering as “effective arity” the maximum cardinality over the hyperedges in the GYO-reduct of $H_Q$. (Recall that the GYO reduct of a hypergraph is obtained by iteratively removing nodes that occur in one hyperedge only and hyperedges included in other hyperedges, until no further removal is possible—see, e.g., [51].)
Compute the simplicial version $H_s$ of the hypergraph $H_V$, that is, the hypergraph having the same set of nodes as $H_V$, and where $edges(H_s) = \{ h' \neq \emptyset \mid h' \subseteq h, h \in edges(H_V) \}$. Therefore, $edges(H_s)$ contains all subsets of every hyperedge of $H_V$. Clearly, $H_s$ can be computed in time $O(2^k \times |edges(H_V)|)$, and the tree projections of $(H_Q, H_s)$ are the same as the tree projections of $(H_Q, H_V)$. To conclude, observe that any tree projection of the latter pair can be computed in polynomial-time by Theorem 5.14 and the fact that, having a squad for every possible set of cops in any squad/hyperedge of $H_V$, the greedy strategies in the game $R&C(H_Q, H_s)$ are precisely the (unrestricted) strategies in the game $R&C(H_Q, H_V)$, which characterize the tree projections of $(H_Q, H_V)$. □

The above tractability result is smoothly inherited by all structural decomposition methods $\text{DM}$ such that the arity of the views in $\nu$-$\text{DM}$ is $O(f(k))$ for some computable function $f$ that does not depend on the size of the input. For instance, this is the case for the methods based on bounded (generalized hyper)tree decompositions, but not for fractional hypertree decompositions. In particular, if $w$ is the fixed maximum width for a class of queries having bounded generalized hypertree width, the maximum arity of the computed views is $w \times k$. Thus, if $p$-$\text{ghw}_w$ denotes the problem of computing a width-$w$ generalized hypertree decomposition of a query, parameterized by the maximum arity of the query atoms, we immediately get the following result.

**Corollary 6.2** The problem $p$-$\text{ghw}_w$ is fixed-parameter tractable.

We believe that this is a useful result. Indeed, even if for queries $Q$ having maximum arity $k$ we have $\text{ghw}(Q) \leq tw(Q) \leq k \times \text{ghw}(Q)$, we know that the problem of evaluating queries is not fixed-parameter tractable, with respect to the (generalized hyper)treewidth parameter. It follows that, under usual fixed-parameter complexity assumptions, an exponential dependency on such width parameters is unavoidable, hence evaluating such queries has a cost of the form $O(n^{f(w)})$, where $w$ is the treewidth (or the hypertree width) and $n$ is the combined size of the database and the query (which is typically largely dominated by the size of the database). We thus argue that employing generalized hypertree width instead of treewidth provides an exponential saving in the query-evaluation time, in general, and it is convenient even for small arity instances. Moreover, recall that the computation of the decomposition depends on the hypergraph only (and not on the database) and, unlike other fixed-parameter algorithms, the algorithm described in Theorem 6.1 is “practical,” as there are no huge constants and the dependence on the arity parameter is single-exponential.

\[13\] Note that the same relationship holds for the monotone strategies and, hence, for the Marshal’s strategies in the Robber and Marshal game over the pair $(H_Q, H_s)$, as observed by Adler [3].
7 Conclusion

In this paper, we have fully characterized the power of algorithms for evaluating conjunctive queries (and constraint satisfaction problems) based on enforcing local consistency. We studied both the general framework where consistency is enforced over arbitrary views and the more specific cases where views are computed according to structural decomposition methods. These results have already found application to the problems of enumerating query answers \[32\] and computing optimal solutions \[33\].

In addition to the questions mentioned in the Introduction, it is worthwhile recalling another open question that eventually finds an answer with these results. The question was raised in \[29\], where the tree projection theorem was proved. Roughly, a query program \(P\) is a finite sequence of steps involving project, select and join operations. The relation computed in the final step is the result of \(P\). The tree projection theorem states that a query program \(P\) solves a query \(Q\) (i.e., the result of \(P\) always coincides with the answers of \(Q\) over its set of output variables) if, and only if, there is a tree projection of \(Q\) w.r.t. the hypergraph associated with the various relations/views determined by \(P\). A crucial point here is that \(P\) is a fixed program, so that the number of its operations does not depend on the database size. The natural question in \[29\] was therefore to ask what happens if \(P\) is allowed to contain a “semi-join loop,” that is, a loop that is to be executed until nothing changes in the involved relations/views. Is it the case that the tree projection theorem still holds for such programs, where the number of steps is data-dependent? The results in the paper provide a positive answer to this question for the setting of simple queries (implicitly) considered in \[29\] and, in fact, also a complete answer covering the general case where queries may contain more atoms over the same relation symbol.

Finally, by exploiting a recent hypergraph-game characterization of tree projections, we also identified new islands of (structural) tractability, and we pinpointed the fixed-parameter tractability of tree projections and of (most) structural decomposition methods when small arity structures are considered. We believe that such results may be very useful in practical applications, and we are currently working on direct implementations of the proposed techniques in real-world database management systems.

There are still a number of interesting questions to be answered about structural decomposition methods. For instance, even for the bounded arity case, the frontier of tractability for the problem of enumerating with polynomial delay the answers of a conjunctive query \(Q\) over a given arbitrary set of output variables is not known (see \[34,10\]). Moreover, in the general unbounded-arity case, the frontier of tractability is not known even for Boolean conjunctive queries. In fact, in the unbounded arity case, the notion of submodular width \[41\] allows us to identify the class of conjunctive queries that are fixed-parameter tractable (where the parameter is the size of the query), assuming the exponential-time hypothesis. As a consequence, we now have an interesting gap to be explored between the polynomial-time tractability of instances having bounded fractional
hypertree width [35, 40] and the fixed-parameter tractability of instances having bounded submodular width.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison-Wesley, 1995.

[2] I. Adler. Marshals, monotone marshals, and hypertree-width. Journal of Graph Theory, 47(4), pp. 275–296, 2004.

[3] I. Adler. Width Functions for Hypertree Decompositions. PhD Thesis, University of Freiburg, 2006.

[4] I. Adler. Tree-Related Widths of Graphs and Hypergraphs. SIAM Journal Discrete Mathematics, 22(1), pp. 102–123, 2008.

[5] I. Adler, G. Gottlob, and M. Grohe. Hypertree-Width and Related Hypergraph Invariants. European Journal of Combinatorics, 28, pp. 2167–2181, 2007.

[6] A. Atserias, A. Bulatov, and V. Dalmau. On the Power of k-Consistency. In Proc. of ICALP’07, pp. 279–290, 2007.

[7] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the Desirability of Acyclic Database Schemes. Journal of the ACM, 30(3), pp. 479–513, 1983.

[8] P.A. Bernstein and N. Goodman. The power of natural semijoins. SIAM Journal on Computing, 10(4), pp. 751–771, 1981.

[9] H.L. Bodlaender and F.V. Fomin. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. SIAM Journal on Computing, 25(6), pp. 1305-1317, 1996.

[10] A. Bulatov, V. Dalmau, M. Grohe, and D. Marx. Enumerating Homomorphisms. Journal of Computer and System Sciences, 78(2), pp. 638-650, 2012.

[11] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. Journal of the ACM, 26:114–133, 1981.

[12] H. Chen and V. Dalmau. Beyond Hypertree Width: Decomposition Methods Without Decompositions. In Proc. of CP’05, pp. 167–181, 2005.

[13] D. A. Cohen and P. Jeavons. The Complexity of Constraint Languages. In Handbook of Constraint Programming, F. Rossi, P. van Beek, and T. Walsh, Eds., Elsevier, 2006.
[14] D. A. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, 74(5), pp. 721-743, 2008.

[15] V. Dalmau, Ph.G. Kolaitis, and M.Y. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In *Proc. of CP ’02*, pp. 310–326, 2002.

[16] R. Dechter. Constraint Processing. Morgan Kaufmann, 2003.

[17] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pp. 353–366, 1989.

[18] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[19] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *Journal of the ACM*, 49(6):716–752, 2002.

[20] P. Fraigniaud and N. Nisse. Connected Treewidth and Connected Graph Searching. In *Proc. of LATIN’06*, pp. 479–490, 2006.

[21] E.C. Freuder. Complexity of K-tree structured constraint satisfaction problems. In *Proc. of the 8th National Conference on Artificial Intelligence*, pp. 4–9, 1990.

[22] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124(2), 243–282, 2000.

[23] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3), pp. 431–498, 2001.

[24] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3), pp. 579–627, 2002.

[25] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4), pp. 775–808, 2003.

[26] G. Gottlob, Z. Miklós, and T. Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *Journal of the ACM*, 56(6), 2009.

[27] G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *Journal of the ACM*, 55(2), Article 9, 2008.

[28] N. Goodman and O. Shmueli. Syntactic characterization of tree database schemas. *Journal of the ACM*, 30(4):767–786, 1983.
[29] N. Goodman and O. Shmueli. The tree projection theorem and relational query processing. *Journal of Computer and System Sciences*, 29(3), pp. 767–786, 1984.

[30] G. Greco and F. Scarcello. Tree Projections: Hypergraph Games and Minimality. In *Proc. of ICALP*’08, pp. 736–747, 2008. Full version available as CoRR technical report at http://arxiv.org/abs/1212.2314.

[31] G. Greco and F. Scarcello. The Power of Tree Projections: Local Consistency, Greedy Algorithms, and Larger Islands of Tractability. In *Proc. of PODS*’10, pp. 327–338, 2010.

[32] G. Greco and F. Scarcello. Structural Tractability of Enumerating CSP Solutions. In *Proc. of CP*’10, pp. 236–251, 2010.

[33] G. Greco and F. Scarcello. Structural Tractability of Constraint Optimization. In *Proc. of CP*’11, pp. 340–355, 2011.

[34] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

[35] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proc. of SODA ’06*, pp. 289–298, 2006.

[36] D. Johnson and A. Klug. Testing containment of Conjunctive Queries Under Functional and Inclusion Dependencies. *Journal of Computer and System Sciences*, 28(1), pp. 167–189, 1984.

[37] D.S. Johnson. A Catalog of Complexity Classes, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pp. 67-161, 1990.

[38] Ph.G. Kolaitis. Constraint Satisfaction, Databases, and Logic. In *Proc. of IJCAI’03*, pp. 1587–1595, 2003.

[39] A. Lustig and O. Shmueli. Acyclic hypergraph projections. *Journal of Algorithms*, 30(2):400–422, 1999.

[40] D. Marx. Approximating fractional hypertree width. *ACM Transactions on Algorithms*, 6(2), 2010.

[41] D. Marx. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. In *Proc. of STOC’10*, pp. 735–744, 2010.

[42] N. Robertson and P.D. Seymour. Graph minors III: Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36, pp. 49–64, 1984.

[43] R. Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *Journal of Computer and System Sciences*, 77(3):572–594, 2011.
[44] Y. Sagiv and O. Shmueli. Solving Queries by Tree Projections. *ACM Transaction on Database Systems*, 18(3), pp. 487–511, 1993.

[45] O. Reingold. Undirected ST-connectivity in log-space, *Journal of the ACM*, 55(4), 2008.

[46] W.L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21, pp. 218-235, 1980.

[47] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58, pp. 22–33, 1993.

[48] F. Scarcello, G. Gottlob, and G. Greco. Uniform Constraint Satisfaction Problems and Database Theory. In *Complexity of Constraints*, LNCS 5250, pp. 156–195, Springer-Verlag, 2008.

[49] S. Subbarayan and H. Reif Andersen. Backtracking Procedures for Hyper-tree, HyperSpread and Connected Hypertree Decomposition of CSPs. In *Proc. of IJCAI’07*, pp. 180–185, 2007.

[50] R.E. Tarjan, and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566-579, 1984.

[51] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.

[52] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB’81*, pp. 82–94, 1981.