The Dynamic Descriptive Complexity of k-Clique

Thomas Zeume*

TU Dortmund University
thomas.zeume@cs.tu-dortmund.de

Abstract. In this work the dynamic descriptive complexity of the k-clique query is studied in a framework introduced by Patnaik and Immerman. It is shown that when edges may only be inserted then k-clique can be maintained by a quantifier-free update program of arity $k-1$, but it cannot be maintained by a quantifier-free update program of arity $k-2$ (even in the presence of unary auxiliary functions). This establishes an arity hierarchy for graph queries for quantifier-free update programs under insertions. The proof of the lower bound uses upper and lower bounds for Ramsey numbers.

1 Introduction

The k-clique query — does a given graph contain a k-clique? — can be expressed by an existential first-order formula with k quantifiers. In this work we study the descriptive complexity of the k-clique query in a setting where edges may be inserted dynamically into a graph. In particular we are interested in lower bounds for the resources necessary to answer this query dynamically.

The dynamic descriptive complexity framework (short: dynamic complexity) introduced by Patnaik and Immerman [14] models the setting of dynamically changing graphs. For a graph subject to changes, auxiliary relations are maintained with the intention to help answering a query $\mathcal{Q}$. When an insertion (or, in the general setting, a deletion) of an edge occurs, every auxiliary relation is updated through a first-order query that can refer to both the graph itself and the auxiliary relations. The query $\mathcal{Q}$ is maintained by such a program, if one designated auxiliary relation always stores the current query result. The class of all queries maintainable by first-order update programs is called DynFO.

Since k-clique can be expressed in existential first-order logic, it can be trivially maintained by a first-order update program. Therefore for characterizing the precise dynamic complexity of this query we need to look at fragments of DynFO. It turns out that k-clique can still be maintained when the update formulas are not allowed to use quantifiers at all and auxiliary relations may only have restricted arity. We obtain the following characterization.

* The author acknowledges the financial support by DFG grant SCHW 678/6-1.

Main result: When only edge insertions are allowed then k-clique ($k \geq 3$) can be maintained by a quantifier-free update program of arity $k-1$, but it cannot be maintained by a quantifier-free update program of arity $k-2$.

Actually we prove that every property expressible by a positive existential first-order formula with k quantifiers and, possibly, negated equality atoms can be maintained by a $(k-1)$-ary quantifier-free program under insertions.

In order to understand why the lower bound contained in the above result is interesting, we shortly discuss the status quo of lower bound methods for the dynamic complexity framework. Up to now very few lower bounds are known; all of them for fragments of DynFO obtained by either bounding the arity of the auxiliary relations or by restricting the usage of quantifiers (or by restricting both). Usually those bounds have been stated only for the setting where both insertions and deletions are allowed. We emphasize that our lower bound for the insertion-only setting immediately transfers to this more general setting.

The study of bounded arity auxiliary relations was started by Dong and Su [4]. They exhibited concrete graph queries that cannot be maintained in unary DynFO, and they showed that DynFO has an arity hierarchy for general (that is non-graph) queries. Both results rely on previously obtained static lower bounds.

Hesse started the study of the quantifier-free fragment of DynFO (short: DynProp) in [13]. Although this fragment appears to be rather weak at first glance, deterministic reachability [13] and regular languages [9] can be maintained in DynProp. In [9], Gelade et al. also provided first lower bounds. They proved that non-regular languages as well as the alternating reachability problem cannot be maintained in this fragment. The use of very restricted graphs in the proof of the latter result implies that there is a $\exists^*\forall\exists^*$FO-definable query that cannot be maintained in DynProp. In [16] it was shown that reachability and 3-clique cannot be maintained in the binary quantifier-free fragment of DynFO.

In general, it is a difficult task to prove lower bounds in the dynamic complexity setting; even when update formulas are restricted to the quantifier-free fragment of first-order logic. We are not at the point where we can, when given a query, apply a set of tools in order to prove that the query cannot be maintained in DynProp. Finding more queries that cannot be maintained in DynProp seems to be a reasonable approach towards finding more generic proof methods.

The lower bound provided by the main result follows this approach and is interesting in two ways. First, it exhibits, for every k, a query in $\exists^k$FO that cannot be maintained in $(k-2)$-ary DynProp, even when only insertions are allowed. We believe that finding simple queries that cannot be maintained will advance the understanding of dynamic complexity. Second, the main result establishes the first arity hierarchy for graph queries, although for a weak fragment of DynFO and for insertions only.

The proof of the lower bound uses upper and lower bounds for Ramsey numbers. This has been quite curious for us.

A natural question is how far this method to prove lower bounds can be pushed. As an intermediate step between the quantifier-free fragment and

DynFO itself, Hesse suggested the study of quantifier-free update programs with auxiliary functions [13]. The main result can be extended as follows.

Extension of the main result: k-clique ($k \geq 3$) cannot be maintained by a quantifier-free update program of arity $k-2$ with unary auxiliary functions.

So far there have been only two lower bounds for dynamic classes with auxiliary functions. Alternating reachability was actually shown to be not maintainable in the quantifier-free fragment of DynFO even in the presence of a successor and a predecessor function [9]. Further, in [16], it was shown that reachability cannot be maintained in unary DynProp with unary auxiliary functions. Thus our extension is a first lower bound for arbitrary unary auxiliary functions and k-ary auxiliary relations, for every fixed k. We also explain why the lower bound technique does not extend to binary auxiliary functions. To this end we show that binary DynQF can maintain every boolean graph property when the domain is large with respect to the actually used domain.

Related work. Up to now we mentioned only work immediately relevant for this work. For the interested reader we give a short list of further related work.

Further lower bounds have been shown in [1, 2, 10]. Further upper bounds have been shown in [8, 12, 15, 10]. Many other aspects such as whether the auxiliary relations are determined by the current structure (see e.g. [14, 3, 10]) and the presence of an order (see e.g. [10]) have been studied.

Outline. In Section 2 we fix some of our notations and in Section 3 we recapitulate the formal dynamic complexity framework. In Sections 4 and 5 we prove the upper and lower bound of the main result, respectively. In Section 6 we study the extension of DynProp by auxiliary functions.

Acknowledgement. I am grateful to Thomas Schwentick for encouraging discussions and many suggestions for improving a draft of this work. Further I thank Nils Vortmeier for proofreading.

2 Preliminaries

We fix some of our notations. Most notations are reused from [16]. The reader can feel free to skip this section and return when encountering unknown notations.

A *domain* D is a finite set. A (relational) *schema* τ consists of a set τ of relation symbols together with an arity function $\mathrm{Ar} : \tau \to \mathbb{N}$. A *database* $\mathcal{D}$ of schema τ with domain D is a mapping that assigns to every relation symbol $R \in \tau$ a relation of arity $\mathrm{Ar}(R)$ over D. A *τ-structure* $\mathcal{S}$ is a pair $(D, \mathcal{D})$ where $\mathcal{D}$ is a database with schema τ and domain D. If $\mathcal{S}$ is a structure over domain D and D' is a subset of D, then the substructure of $\mathcal{S}$ induced by D' is denoted by $\mathcal{S} \restriction D'$.

A tuple $\vec{a} = (a_1, \ldots, a_k)$ is *$\prec$-ordered* with respect to a linear order[1] $\prec$ of the domain, if $a_1 \prec \ldots \prec a_k$. The *$k$-ary atomic type* $\langle \mathcal{S}, \vec{a} \rangle$ of $\vec{a}$ over D with respect

[1] All linear orders in this work are strict.

to $\mathcal{S}$ is the set of all atomic formulas $\varphi(\vec{x})$ with $\vec{x} = (x_1, \ldots, x_k)$ for which $\varphi(\vec{a})$ holds in $\mathcal{S}$, where $\varphi(\vec{a})$ is short for the substitution of $\vec{x}$ by $\vec{a}$ in φ. As we only consider atomic types here, we will often simply say type instead of atomic type.

For a set A, denote by A^k the set of all k-tuples over A and, following [11], by $[A]^k$ the set of all k-element subsets of A. A *k-hypergraph* G is a pair (V, E) where V is a set and E is a subset of $[V]^k$. If $E = [V]^k$ then G is called *complete*. An *r-coloring col* of G is a mapping that assigns to every edge in E a color from $\{1, \ldots, r\}$. A *r-colored k-hypergraph* is a pair (G, col) where G is a k-hypergraph and col is a r-coloring of G. If the name of the r-coloring is not important we also say *G is r-colored*.

3 Dynamic Setting

The following introduction to dynamic descriptive complexity is borrowed from previous work [16, 17]. Although the focus of this work is on maintaining the k-clique query under insertions, we introduce the general dynamic complexity framework in order to be able to give a broader discussion of concrete results.

A *dynamic instance* of a query $\mathcal{Q}$ is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a database over a domain D and α is a sequence of modifications to $\mathcal{D}$, i.e. a sequence of insertions and deletions of tuples over D. The dynamic query $\textsc{Dyn}(\mathcal{Q})$ yields as result the relation that is obtained by first applying the modifications from α to $\mathcal{D}$ and then evaluating the query $\mathcal{Q}$ on the resulting database. The database resulting from applying a modification δ to a database $\mathcal{D}$ is denoted by $\delta(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of modifications $\alpha = \delta_1 \ldots \delta_m$ to a database $\mathcal{D}$ is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \delta_m(\ldots(\delta_1(\mathcal{D}))\ldots)$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every (input) database $\mathcal{D}$, an initial state with initial auxiliary data. The latter defines the new state of the dynamic program for each possible modification δ.

A *dynamic schema* is a tuple where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary database, respectively. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}}$.

Definition 1. *(Update program) An* update program $\mathcal{P}$ *over a dynamic schema (τ_{in}, τ_{aux}) is a set of first-order formulas (called update formulas in the following) that contains, for every $R \in \tau_{aux}$ and every $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ with $S \in \tau_{in}$, an update formula $\phi_\delta^R(\vec{x}; \vec{y})$ over the schema τ where $\vec{x}$ and $\vec{y}$ have the same arity as S and R, respectively.*

A *program state* $\mathcal{S}$ over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a structure $(D, \mathcal{I}, \mathcal{A})$ where D is a finite domain, $\mathcal{I}$ is a database over the input schema (the *current database*) and $\mathcal{A}$ is a database over the auxiliary schema (the *auxiliary database*). The *semantics of update programs* is as follows. For a modification $\delta(\vec{a})$, where $\vec{a}$ is a tuple over D, and program state $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ we denote by $P_\delta(\mathcal{S})$ the state $(D, \delta(\mathcal{I}), \mathcal{A}')$, where $\mathcal{A}'$ consists of relations $R' \stackrel{\text{def}}{=} \{\vec{b} \mid \mathcal{S} \models \phi_\delta^R(\vec{a}; \vec{b})\}$. The effect $P_\alpha(\mathcal{S})$ of a modification sequence $\alpha = \delta_1 \ldots \delta_m$ to a state $\mathcal{S}$ is the state $P_{\delta_m}(\ldots(\mathcal{P}_{\delta_1}(\mathcal{S}))\ldots)$.

Definition 2. *(Dynamic program) A* dynamic program *is a triple* $(P, \textsc{Init}, Q)$*, where*

- P *is an update program over some dynamic schema* (τ_{in}, τ_{aux})*,*
- Init *is a mapping that maps* τ_{in}*-databases to* τ_{aux}*-databases, and*
- $Q \in \tau_{aux}$ *is a designated* query symbol*.*

A dynamic program $\mathcal{P} = (P, \textsc{Init}, Q)$ *maintains* a dynamic query $\textsc{Dyn}(\mathcal{Q})$ if, for every dynamic instance $(\mathcal{D}, \alpha)$, the relation $\mathcal{Q}(\alpha(\mathcal{D}))$ coincides with the query relation $Q^{\mathcal{S}}$ in the state $\mathcal{S} = P_\alpha(\mathcal{S}_{\textsc{Init}}(\mathcal{D}))$, where $\mathcal{S}_{\textsc{Init}}(\mathcal{D})$ is the initial state for $\mathcal{D}$, i.e. $\mathcal{S}_{\textsc{Init}}(\mathcal{D}) \stackrel{\text{def}}{=} (D, \mathcal{D}, \textsc{Init}_{\text{aux}}(\mathcal{D}))$.

Definition 3. *(*DynFO *and* DynProp*)* DynFO *is the class of all dynamic queries that can be maintained by dynamic programs with first-order update formulas and arbitrary initialization mappings.* DynProp *is the subclass of* DynFO*, where update formulas are not allowed to use quantifiers. A dynamic program is* k-ary *if the arity of its auxiliary relation symbols is at most* k*. By* k*-ary* DynProp *(resp.* DynFO*) we refer to dynamic queries that can be maintained with* k*-ary dynamic programs.*

In the literature, classes with restricted initialization mappings have been studied as well, see [17] for a discussion. The choice made here is not a real restriction as lower bounds proved for arbitrary initialization hold for restricted initialization as well. On the other hand, our upper bounds also hold for other settings of initialization; with the single exception of Theorem 6, which requires arbitrary initialization. Furthermore our results also hold in the related setting where domains can be infinite.

4 k-Clique can be maintained under insertions with Arity $k-1$

In this section we prove that the k-clique query can be maintained in $(k-1)$-ary DynProp when only edge insertions are allowed. Instead of proving this result directly, we show that the class of all semi-positive existential first-order queries can be maintained in DynProp under insertions.

A *positive existential first-order query* over schema τ is a query that can be expressed by a first-order formula of the form $\varphi(\vec{y}) = \exists \vec{x} \psi(\vec{x}, \vec{y})$ where ψ is a quantifier-free formula that contains only literals of the form $z_i = z_j$ and $R(\vec{z})$ with $R \in \tau$. For *semi-positive existential first-order queries* literals of the form $z_i \neq z_j$ are allowed as well.

We will prove that every semi-positive existential first-order query can be maintained in DynProp when only insertions are allowed. More precisely, it will be shown that $(k-1)$-ary DynProp is sufficient for boolean queries with k existential quantifiers. In particular k-Clique can be maintained in $(k-1)$-ary DynProp. Before turning to the proof we give some intuition.

Example 1. We show how to maintain 3-Clique in binary DynProp under insertions. The very simple idea is to use an additional binary auxiliary relation R

that stores all edges whose insertion would complete a triangle. Hence a tuple (a, b) is inserted into R as soon as deciding whether there is a 3-clique containing the nodes a and b only depends on those two nodes. More precisely (a, b) is added to R, if an edge (c, a) is inserted to the input graph and the edge (c, b) is already present (or vice versa).

Thus the update formula for R is

$$\phi^R_{\text{INS}E}(u, v; x, y) \stackrel{\text{def}}{=} u \neq v \wedge x \neq y \wedge \Big(\big(E\{u, y\} \wedge v = x\big) \vee \big(E\{u, x\} \wedge v = y\big) \\ \vee \big(E\{v, y\} \wedge u = x\big) \vee \big(E\{v, x\} \wedge u = y\big) \Big)$$

where $E\{x, y\}$ is an abbreviation for $E(x, y) \vee E(y, x)$.

The update formula for the query symbol Q is

$$\phi^Q_{\text{INS}E}(u, v; x, y) = Q \vee R(u, v)$$

□

The general proof for arbitrary semi-positive existential first-order properties extends the approach taken in the example.

Theorem 1. *An ℓ-ary query expressible by a semi-positive existential first-order formula with k quantifiers can be maintained under insertions in $(\ell + k - 1)$-ary* DynProp.

Proof sketch. For simplicity we restrict the sketch to boolean graph queries. The proof easily carries over to arbitrary semi-positive existential queries.

Basically a semi-positive existential sentence with k quantifiers can state which subgraphs with k nodes shall occur in a graph. Therefore it is sufficient to construct a dynamic quantifier-free program that maintains whether the input graph contains a subgraph H. Such a program can work as follows. For every induced, proper subgraph $H' = \{u_1, \ldots, u_m\}$ of H, the program maintains an auxiliary relation that stores all tuples $\vec{a} = (a_1, \ldots, a_m)$ such that inserting H' into $\{a_1, \ldots, a_m\}$ (with a_i corresponding to u_i) yields a graph that contains H.

In particular, auxiliary relations have arity of at most $k - 1$ (as only proper subgraphs of H have a corresponding auxiliary relation). Furthermore the graph H is contained in the input graph whenever the value of the 0-ary relation corresponding to the empty subgraph of H is true. In the example above, the relation R is the relation for the subgraph of the 3-clique graph that consists of a single edge, and the designated query relation is the 0-ary relation for the empty subgraph.

Those auxiliary relations can be updated as follows. Assume that a tuple $\vec{a} = (a_1, \ldots, a_m)$ is contained in the relation corresponding to H'. If, after the insertion of an edge with end point a_m, every edge from u_m in H' has a corresponding edge from a_m in the graph induced by $\{a_1, \ldots, a_m\}$, then the tuple $\vec{a}' = (a_1, \ldots, a_{m-1})$ has to be inserted into the auxiliary relation for the subgraph $H' \restriction \{u_1, \ldots, u_{m-1}\}$. This is because inserting the graph $H' \restriction \{u_1, \ldots, u_{m-1}\}$ into $\{a_1, \ldots, a_{m-1}\}$ will now yield a graph that contains H. Observe that for those updates no quantifiers are needed. □

5 k-Clique cannot be maintained with Arity $k-2$

In this section we prove that the k-clique query cannot be maintained by a $(k-2)$-ary quantifier-free update program when $k \geq 3$. The proof uses two main ingredients; the substructure lemma from [9, 16] and a new Ramsey-like lemma. We state those lemmas next.

For stating the substructure lemma we need the following notion of corresponding modifications in isomorphic structures. Let π be an isomorphism from a structure $\mathcal{A}$ to a structure $\mathcal{B}$. Two modifications $\delta(\vec{a})$ on $\mathcal{A}$ and $\delta'(\vec{b})$ on $\mathcal{B}$ are said to be π-*respecting* if $\delta = \delta'$ and $\vec{b} = \pi(\vec{a})$. Two sequences $\alpha = \delta_1 \cdots \delta_m$ and $\beta = \delta'_1 \cdots \delta'_m$ of modifications respect π if δ_i and δ'_i are π-respecting for every $i \leq m$. For a discussion of the lemma we refer the reader to the long version of [16]. Recall that $P_\alpha(\mathcal{S})$ denotes the state obtained by executing the dynamic program $\mathcal{P}$ for the modification sequence α from state $\mathcal{S}$.

Lemma 1 (Substructure Lemma [9, 16]). *Let $\mathcal{P}$ be a* DynProp*-program with designated Boolean query symbol Q, and let $\mathcal{S}$ and $\mathcal{T}$ be states of $\mathcal{P}$ with domains S and T. Further let $A \subseteq S$ and $B \subseteq T$ such that $\mathcal{S} \restriction A$ and $\mathcal{T} \restriction B$ are isomorphic via π. Then Q has the same value in $P_\alpha(\mathcal{S})$ and $P_\beta(\mathcal{T})$ for all π-respecting sequences α, β of modifications on A and B.*

The second ingredient exhibits a disparity between upper bounds for Ramsey numbers in k-ary structures and lower bounds for Ramsey numbers in $(k+1)$-dimensional hypergraphs. While the first condition in the following lemma guarantees the existence of a Ramsey clique of size $f(|A|)$ in k-ary structures over A, the second condition states that there is a 2-coloring of the complete $(k+1)$-hypergraph over A that does not contain a Ramsey clique of size $f(|A|)$. This disparity is the key to the lower bound proof.

Lemma 2. *Let $k \in \mathbb{N}$ be arbitrary and τ a k-ary schema. Then there is a function $f : \mathbb{N} \to \mathbb{N}$ and an $n \in \mathbb{N}$ such that for every domain A larger than n the following conditions are satisfied:*

(S1) For every τ-structure $\mathcal{S}$ over A and every linear order $\prec$ on A there is a subset A' of A of size $|A'| \geq f(|A|)$ such that all $\prec$-ordered k-tuples over A' have the same type in $\mathcal{S}$.

(S2) The set $[A]^{k+1}$ of all $(k+1)$-hyperedges over A can be partitioned into two sets B and B' such that for every set $A' \subseteq A$ of size $|A'| \geq f(|A|)$ there are $(k+1)$-hyperedges $b, b' \subseteq A'$ with $b \in B$ and $b' \in B'$.

The two lemmas above can be used to obtain the lower bound for the k-clique query as follows. The proof of Lemma 2 will be sketched afterwards.

Theorem 2. $(k+2)$-Clique *($k \geq 1$) cannot be maintained under insertions by a k-ary* DynProp*-program.*

Proof. Towards a contradiction assume that there is a k-ary DynProp-program $\mathcal{P}$ over k-ary schema τ that maintains $(k+2)$-Clique. Let n and f

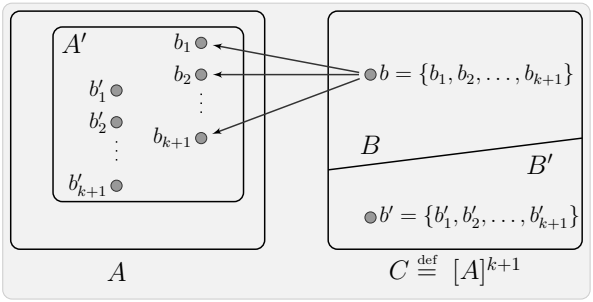

Fig. 1. The construction from the proof of Theorem 2.

be as in Lemma 2. For a set A larger than n let $\prec$ be an arbitrary order on A and let $D \stackrel{\text{def}}{=} A \uplus C$ be a domain with $C \stackrel{\text{def}}{=} [A]^{k+1}$. Further let B, B' be the partition of $[A]^{k+1}$ guaranteed to exist by (S2) in Lemma 2.

We consider a state $\mathcal{S}$ over domain D where the input graph G contains the edges

$$\{(b_1, b), (b_2, b), \ldots, (b_{k+1}, b) \mid b = \{b_1, b_2, \ldots, b_{k+1}\} \in B\}$$

See Figure 1 for an illustration.

By Condition (S1) there is a subset $A' \subseteq A$ of size $|A'| \geq f(|D|)$ such that all ordered k-tuples over A' have the same τ-type in $\mathcal{S}$. Then by (S2) there are $(k+1)$-hyperedges $b, b' \subseteq A'$ with $b \in B$ and $b' \in B'$. Without loss of generality $b = \{b_1, b_2, \ldots, b_{k+1}\}$ with $b_1 \prec \ldots \prec b_{k+1}$ and $b' = \{b'_1, b'_2, \ldots, b'_{k+1}\}$ with $b'_1 \prec \ldots \prec b'_{k+1}$. By construction of the graph G, all elements in b are connected to the node $b \in C$ while there is no node in C connected to all elements of b'. Thus applying the modification sequences

(α) Insert the edges (b_i, b_j) in lexicographic order with respect to $\prec$.
(β) Insert the edges (b'_i, b'_j) in lexicographic order with respect to $\prec$.

yields one graph with a $(k+2)$-clique and one graph without a $(k+2)$-clique, respectively. However, by the substructure lemma, the program $\mathcal{P}$ yields the same result since the substructures induced by $\vec{b} = (b_1, \ldots, b_{k+1})$ and $\vec{b}' = (b'_1, \ldots, b'_{k+1})$ are isomorphic. This is the desired contradiction. □

In the following we give a rough sketch of the proof of Lemma 2. The *k-dimensional Ramsey* number for r colors and clique-size l, denoted by $R_k(l; r)$, is the smallest number n such that every r-coloring of a complete k-hypergraph with n nodes has a monochromatic clique of size l. The *tower function* $\text{tow}_k(n)$ is defined by $\text{tow}_k(n) \stackrel{\text{def}}{=} 2^{2^{\cdot^{\cdot^{2^n}}}}$ with $(k-1)$ many 2's. The following classical result for asymptotic bounds on Ramsey numbers due to Erdős, Hajnal and Rado is the key to prove Lemma 2. The concrete formulation is from [5].

Theorem 3. *[6, 7] Let k, l and r be positive integers. Then there exist positive constants c_k, $c_{k,r}$ and l_k such that*

(a) $R_k(l;r) \leq tow_k(c_{k,r}l)$
(b) $R_k(l;2) \geq tow_{k-1}(c_k l^2)$ *for all* $l \geq l_k$

The theorem immediately implies that (T1) Ramsey cliques in r-colored k-dimensional complete hypergraphs are of size at least $\Omega(\log^{(k-1)}(n))$; and that (T2) there are 2-colorings of the $(k+1)$-dimensional complete hypergraphs such that monochromatic cliques are of size $O((\log^{(k-1)}(n))^{\frac{1}{2}})$. Here $\log^{(k)}(n)$ denotes $\log(\log(\ldots(\log n)\ldots))$ with k many log's. Those conditions are already quite similar to the conditions (S1) and (S2). The major difference is that (T1) is about hypergraphs and not about structures with a k-ary schema.

Fortunately the upper bound from Theorem 3 can be generalized to Ramsey numbers for structures. To this end some notions need to be transferred from hypergraphs to structures. Let τ be a k-ary schema, let $\mathcal{S}$ be a τ-structure over domain D and $\prec$ an order on D. A subset $D' \subseteq D$ of $\mathcal{S}$ is called an *$\prec$-ordered τ-clique* if all $\prec$-ordered k-tuples $\vec{a} \in D'^k$ have the same τ-type. Denote by $R(l;\tau)$ the smallest number n such that every τ-structure with n elements contains an $\prec$-ordered τ-clique of size l, for every order $\prec$ of the domain.

Theorem 4. *Let τ be a schema with maximal arity k and let l be a positive integer. Then there exists a constant c such that $R(l;\tau) \leq tow_k(cl)$.*

Proof. The proof of Observation 1' in [9, p. 11] yields this bound. For the sake of completeness we repeat the construction.

Let $\mathcal{S}$ be a τ-structure over domain D of size $\mathrm{tow}_k(cl)$. Further let $\prec$ be an arbitrary order on D. Define a coloring *col* of the complete k-dimensional hypergraph with nodes D as follows. An edge $\{e_1, \ldots, e_k\}$ with $e_1 \prec \ldots \prec e_k$ is colored by the type $\langle S, e_1, \ldots, e_k\rangle$. By Theorem 3 there is an induced monochromatic sub-k-hypergraph with domain $D' \subseteq D$ with $|D'| \geq l$. By the definition of the coloring *col*, two $\prec$-ordered k-tuples over D' have the same type and therefore D' is a $\prec$-ordered τ-clique in $\mathcal{S}$ as well. □

The previous theorem implies that (T1') Ramsey cliques in k-ary structures are of size at least $\Omega(\log^{(k-1)}(n))$. Then Lemma 2 follows from the facts (T1') and (T2) by choosing f as the function in $\Omega(\log^{(k-1)}(n))$ guaranteed to exist by (T1'). The function f satisfies (S1) and (S2) due to (T1') and (T2). This completes the lower bound proof.

6 Adding Auxiliary Functions

In quantifier-free update programs, as considered up to here, only the modified and the updated tuple can be accessed while updating an auxiliary tuple. Since lower bounds for first-order update programs where arbitrary elements can be accessed in updates seem to be out of reach for the moment, it seems natural to look for extensions of quantifier-free update programs that allow for accessing more elements in some restricted way.

Here we study DynProp programs extended by auxiliary functions, an extension proposed by Hesse [13]. Auxiliary functions are updated by update terms that may use function symbols and a special if-then-else construct. For a discussion of previous work on this extension we refer to the introduction. A formal treatment of this extension can be found in [16, 9].

The lower bound from the previous section can be generalized to quantifier-free programs that may use unary functions.

Theorem 5. $(k+2)$-Clique *($k \geq 1$) cannot be maintained under insertions by a k-ary* DynProp*-program with unary auxiliary functions.*

The proof is along the same lines as the proof of Theorem 2. Instead of the substructure lemma for DynProp a corresponding lemma for DynQF from [9, 16] is used. However, this substructure lemma for DynQF requires to exhibit isomorphic substructures that, additionally, have similar neighbourhoods.

A natural question is whether the lower bounds transfer to k-ary auxiliary functions. We conjecture that they do, but we will argue that the techniques used so far are not sufficient for proving lower bounds for binary auxiliary functions.

The fundamental difference between unary and binary auxiliary functions is that, on the one hand, unary functions can access elements that depend either on the tuple that has been modified in the input structure or on the auxiliary tuple under consideration but not on both. On the other hand binary functions can access elements that depend on both tuples.

A consequence is that binary DynQF can maintain every boolean graph property when the domain is large with respect to the actually used domain. We make this more precise. In the following we assume that all domains D are a disjoint union of a modifiable domain D^+ and a non-modifiable domain D^-, and that modifications may only involve tuples over D^+. Auxiliary data, however, may use the full domain. A dynamic complexity class $\mathcal{C}$ *profits from padding* if every boolean graph property can be maintained whenever the non-modifiable domain is sufficiently large in comparison to the modifiable domain[2].

Above we have seen that DynProp with unary auxiliary functions does not profit from padding.

Theorem 6. *Binary* DynQF *profits from padding.*

Proof sketch. We only show how ternary DynQF profits from padding, the adaption to binary DynQF is not difficult. Let $\mathcal{Q}$ be an arbitrary boolean graph property. In the following we construct a ternary DynQF program $\mathcal{P}$ which maintains $\mathcal{Q}$ if $2^{|D^+|^2} = |D^-|$. The idea is to identify D^- with the set of all graphs over D^+, that is D^- contains an element c_G for every graph G over D^+. A unary relation $R_{\mathcal{Q}}$ stores those elements of D^- that correspond to graphs with the property $\mathcal{Q}$. Finally the program maintains a pointer p to the element in D^- corresponding to the graph stored in D^+. The pointer is updated upon

[2] Note that this type of padding is different from the padding technique used by Patnaik and Immerman for maintaing a PTIME-complete problem in DynFO [14].

edge modification by using ternary functions f_{INS} and f_{DEL} initialized by the initialization mapping in a suitable way.

The program $\mathcal{P}$ is over schema $\tau = \{Q, p, f_{\text{INS}}, f_{\text{DEL}}, R_{\mathcal{Q}}\}$ where p is a constant, f_{INS} and f_{DEL} are ternary function symbols, $R_{\mathcal{Q}}$ is a unary relation symbol and Q is the designated query symbol.

We present the initialization mapping of $\mathcal{P}$ first. The initial state $\mathcal{S}$ for a graph H is defined as follows. The functions f_{INS} and f_{DEL} are independent of H and defined via

$$f^{\mathcal{S}}_{\text{INS}}(a, b, c_G) = c_{G+(a,b)}$$
$$f^{\mathcal{S}}_{\text{DEL}}(a, b, c_G) = c_{G-(a,b)}$$

for $a, b \in D^+$ and $c_G \in D^-$. For all other arguments the value of the functions is arbitrary. Here $G+(a,b)$ and $G-(a,b)$ denote the graphs obtained by adding the edge (a,b) to G and removing the edge (a,b) from G, respectively. The relation $R^{\mathcal{S}}_{\mathcal{Q}}$ contains all c_G with $G \in \mathcal{Q}$. Finally the constant $p^{\mathcal{S}}$ points to c_H.

It remains to exhibit the update formulas. After a modification, the pointer p is moved to the node corresponding to the modified graph, and the query bit is updated accordingly:

$$t^p_{\text{INS}}(u, v) = f_{\text{INS}}(u, v, p) \qquad t^Q_{\text{INS}}(u, v) = R_{\mathcal{Q}}(f_{\text{INS}}(u, v, p))$$
$$t^p_{\text{DEL}}(u, v) = f_{\text{DEL}}(u, v, p) \qquad t^Q_{\text{DEL}}(u, v) = R_{\mathcal{Q}}(f_{\text{DEL}}(u, v, p))$$

By further extending the non-modifiable domain, this construction can be extended to binary DynQF. □

Hence the ability to profit from padding distinguishes binary DynQF and DynProp extended by unary functions. Although the proof of the preceding theorem requires the non-modifiable domain to be of exponential size with respect to the modifiable domain, the construction also explains why the lower bound technique from the previous sections cannot be immediately applied to binary DynQF. In the lower bound construction only tuples over the set A are modified, while tuples containing elements from $C = [a]^k$ are not modified. Thus, by treating C as a non-modifiable domain, it can be used to store information as in the proof above. As the modification sequences used in the lower bounds are of length k^2, finding similar substructures in structures with binary auxiliary functions becomes much harder.

7 Conclusion and Future Work

In this work we exhibited a precise dynamic descriptive complexity characterization of the k-clique query when only insertions are allowed. The characterization implies an arity hierarchy for graph queries for DynProp under insertions. We also discussed the limit of our proof methods.

While proving lower bounds for full DynFO — a major long-term goals in dynamic descriptive complexity — might be really hard to achieve, we believe

that the following goals are suitable for both developing new lower bound methods and for further improving the current methods.

Goal 1. *Prove general quantifier-free lower bounds for insertions and deletions for the reachability query and the k-clique query.*

It is known that both queries cannot be maintained in binary DynProp. We conjecture that 3-clique cannot be maintained in DynProp under deletions.

Goal 2. *Find a general framework for proving quantifier-free lower bounds.*

Goal 3. *Find a natural query that cannot be maintained in binary* DynQF.

References

[1] Guozhu Dong, Leonid Libkin, and Limsoon Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In *DBPL '95*, page 7, 1995.

[2] Guozhu Dong, Leonid Libkin, and Limsoon Wong. Incremental recomputation in local languages. *Inf. Comput.*, 181(2):88–98, 2003.

[3] Guozhu Dong and Jianwen Su. Deterministic FOIES are strictly weaker. *Ann. Math. Artif. Intell.*, 19(1-2):127–146, 1997.

[4] Guozhu Dong and Jianwen Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998.

[5] Dwight Duffus, Hanno Lefmann, and Vojtech Rödl. Shift graphs and lower bounds on Ramsey numbers $r_k(l;r)$. *Discrete Mathematics*, 137(1-3):177–187, 1995.

[6] P. Erdös and R. Rado. Combinatorial theorems on classifications of subsets of a given set. *Proc. London Math. Soc. (3)*, 2:417–439, 1952.

[7] Paul Erdős, András Hajnal, and Richard Rado. Partition relations for cardinal numbers. *Acta Mathematica Hungarica*, 16(1):93–196, 1965.

[8] Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *PODS '98*, pages 235–243, 1998.

[9] Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012.

[10] Erich Grädel and Sebastian Siebertz. Dynamic definability. In *ICDT '12*, pages 236–248, 2012.

[11] R.L. Graham, B.L. Rothschild, and J.H. Spencer. *Ramsey Theory*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1990.

[12] William Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 296(3):473–485, 2003.

[13] William Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.

[14] Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.

[15] Volker Weber and Thomas Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007.

[16] Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. In *MFCS '13*, pages 837–848, 2013. Full version available at http://arxiv.org/abs/1306.3056.

[17] Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In *ICDT '14*, pages 38–49, 2014.