Dynamic Complexity: A brief introduction

Nikolaos Nikolopoulos

ALMA
INTER-INSTITUTIONAL GRADUATE PROGRAM
≪ALGORITHMS, LOGIC AND DISCRETE MATHEMATICS≫

Overview

Overview

Motivation

There are problems which cannot be appropriately described by traditional complexity classes. Examples :

- Web server goes offline; data packages' rerouting
- Data change in a database; efficient computation of new queries
- Image recognition after a small addition or subtraction of an element

But what is the similarity between those problems?

Motivation

There are problems which cannot be appropriately described by traditional complexity classes. Examples :

- Web server goes offline; data packages' rerouting
- Data change in a database; efficient computation of new queries
- Image recognition after a small addition or subtraction of an element

But what is the similarity between those problems? Change!

Change $\rightarrow$ affects the problem state

Locality of Change $\rightarrow$ Auxiliary Data $\rightarrow$ Efficient Updates

Approaches

In general approaches for such dynamic problems utilize *auxiliary data* i.e. some additional information besides input data to boost the update process. Mainly there are two different approaches in that direction :

- *Algorithmic* an effort to design non-trivial algorithms that need less resources (time, space, disk access etc) to recompute desired results.
- *Declarative* utilization of logical formalism to specify updates of the auxiliary data. Hence, every data change is modeled using a series of logical queries.

"Naturalness" in Complexity

Natural descriptive characterizations :

- *SPACE* $\rightarrow \#$ variables
- *PARALLEL TIME* $\rightarrow$ quantifier depth
- *SEQ. TIME*[1] $\rightarrow$?

[1] probably unnatural stemming from the "*Von-Neumann bottleneck*"

"Naturalness" in Complexity

Natural descriptive characterizations :

- *SPACE* $\rightarrow$ # variables
- *PARALLEL TIME* $\rightarrow$ quantifier depth
- *SEQ. TIME*[1] $\rightarrow$?

In Dynamic Complexity naturalness is still unclear; however the framework from Descriptive Complexity is being kept.

[1] probably unnatural stemming from the "*Von-Neumann bottleneck*"

Descriptive Complexity Framework

As usual, a *vocabulary* $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ where:

Descriptive Complexity Framework

As usual, a *vocabulary* $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ where:

- $R_i^{a_i}$ is a relation R_i with arity a_i
- c_j is a constant

Descriptive Complexity Framework

As usual, a *vocabulary* $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ where:

- $R_i^{a_i}$ is a relation R_i with arity a_i
- c_j is a constant

A *structure* with vocabulary τ looks like:

$$\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$$

Also, $\forall i\ R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ and $\forall c_j \in \tau \Rightarrow \exists c_j^{\mathcal{A}} : c_j^{\mathcal{A}} \in |\mathcal{A}|$

Descriptive Complexity Framework

As usual, a *vocabulary* $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ where:

- $R_i^{a_i}$ is a relation R_i with arity a_i
- c_j is a constant

A *structure* with vocabulary τ looks like:

$$\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$$

Also, $\forall i \; R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ and $\forall c_j \in \tau \Rightarrow \exists c_j^{\mathcal{A}} : c_j^{\mathcal{A}} \in |\mathcal{A}|$

Finally, since $\mathrm{STRUCT}[\tau] = \{\mathcal{B} \mid \mathcal{B} \text{ is a finite structure over } \tau\}$, a problem P corresponds to a set $S : S \subseteq \mathrm{STRUCT}[\tau]$ for some τ.

Overview

1 Introduction

2 Dynamic Complexity Classes
- *Dyn-C*
- *Dyn-FO*
- Problems in *Dyn-FO*
- *Dyn-PROP*
- Dynamic Reductions
- Historical overview

3 Epilogue

Definition of *Dyn-C*

Definition (*Dyn-$\mathcal{C}$*)

Let C be a complexity class, let $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$, let $S \subseteq \mathrm{STRUCT}[\sigma]$ and let[2] :

$$\mathcal{R}_{n,\sigma} = \{ins(i, \bar{a}), del(i, \bar{a}), set(j, \bar{a}) \ s.t. \\ 1 \leq i \leq r, \bar{a} \in \{0, \ldots, n-1\}^{a_i}, 1 \leq j \leq s\}$$

Also let $eval_{n,\sigma} : \mathcal{R}^*_{n,\sigma} \to \mathrm{STRUCT}[\sigma] \ s.t. \ eval_{n,\sigma}(\emptyset) = \mathcal{A}_0^n$.

Then $S \in Dyn\text{-}\mathcal{C} \Leftrightarrow \exists T \subseteq \mathrm{STRUCT}[\tau] \ : \ T \in C \wedge \exists f_n, g_n \ :$
$f_n : \mathcal{R}^*_{n,\sigma} \to \mathrm{STRUCT}[\tau]; g_n : \mathrm{STRUCT}[\tau] \times \mathcal{R}_{n,\sigma} \to \mathrm{STRUCT}[\tau]$

[2] we may also have an enhanced set of operations $\mathcal{O}_{n,\sigma}$

Dyn-C cont'd

Functions f_n, g_n should satisfy the following properties :

- $g_n, f_n(\emptyset)$ computable in complexity $\mathcal{C}$ (with respect to n)
- $\forall r \in \mathcal{R}^*_{n,\sigma}\ [eval_{n,\sigma}(r) \in S \Leftrightarrow f_n(r) \in T]$
- $\forall r \in \mathcal{R}^*_{n,\sigma}, s \in \mathcal{R}_{n,\sigma}\ [f_n(rs) = g_n(f_n(r), s)]$
- $\|f_n(r)\| = \|eval_{n,\sigma}(r)^{\mathcal{O}(1)}\|$

Dyn-C cont'd

Functions f_n, g_n should satisfy the following properties :

- $g_n, f_n(\emptyset)$ computable in complexity $\mathcal{C}$ (with respect to n)
- $\forall r \in \mathcal{R}^*_{n,\sigma}\ [eval_{n,\sigma}(r) \in S \Leftrightarrow f_n(r) \in T]$
- $\forall r \in \mathcal{R}^*_{n,\sigma}, s \in \mathcal{R}_{n,\sigma}\ [f_n(rs) = g_n(f_n(r), s)]$
- $\|f_n(r)\| = \|eval_{n,\sigma}(r)^{\mathcal{O}(1)}\|$

There are also some variants of *Dyn-*$\mathcal{C}$:

- *Dyn*$_s$-$\mathcal{C}$ if we forbid delete queries in $\mathcal{R}_{n,\sigma}$
- *Dyn-*$\mathcal{C}^+$ if we allow polynomial precomputation for $f_n(\emptyset)$

Note: Always only one $r \in \mathcal{R}^*_{n,\sigma}$ that affects a tuple is allowed (or a constant number at most)

Dyn-FO definition & a small example

Based on the previous definition of *Dyn-*$\mathcal{C}$:

Definition (*Dyn-FO*)

Dyn-FO is the set of all boolean queries that can be maintained using *FO* formulas after changes that affect a constant number of tuples in the input.

Example (PARITY $\in$ *Dyn-FO*)

PARITY query is true iff input string has an odd number of 1s. Let $\sigma = \langle M^1 \rangle$ the vocabulary of PARITY, $\mathcal{A}_w$ the encoding of a binary string w so that $\mathcal{A} \models M(i)$ iff $w(i) = 1.$ Also we consider $\tau = \langle M^1, b \rangle$ as vocabulary and T our *FO* problem as $T = \{\mathcal{A} \in \text{STRUCT}[\tau] \mid \mathcal{A} \models b\}$; b is a boolean constant symbol that acts as flag: it keeps track of the current parity.

PARITY example cont'd

Example (PARITY $\in$ *Dyn-FO* cont'd)

We initialize $f_n(\emptyset) = \langle \{0, 1, \ldots, n-1\}, \emptyset, \mathit{false} \rangle$, so that our data structure is all 0s and constant b as false. Our objective is clear : *FO* computation of $g_n(\mathcal{B}, s)\ \forall s \in R_{n,\sigma}$. So we have the following *FO* formulas :

$$\begin{array}{rl} \textbf{ins}(a, M) & \begin{array}{l} M' \equiv M(x) \vee x = a \\ b' \equiv (b \wedge M(a)) \vee (\neg b \wedge \neg M(a)) \end{array} \\ \textbf{del}(a, M) & \begin{array}{l} M' \equiv M(x) \wedge x \neq a \\ b' \equiv (b \wedge \neg M(a)) \vee (\neg b \wedge M(a)) \end{array} \end{array}$$

Since those formulas are *FO* and work in constant time, we get that PARITY $\in$ *Dyn-FO*.

Note: it is known that PARITY $\notin$ *FO*!

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	
00001		

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	
00001		00001 1

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	
00001		00001 1
	ins(2,S)	

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	
00001		00001 1
	ins(2,S)	
01001		

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	
00001		00001 1
	ins(2,S)	
01001		01001 0

PARITY: How it works?

Structure	Request	Data Structure
00000		00000 0
	ins(1,S)	
10000		10000 1
	del(1,S)	
00000		00000 0
	ins(5,S)	
00001		00001 1
	ins(2,S)	
01001		01001 0

On state i we have a query $g_n(\mathcal{B}_{i-1}, r_i)$ where $\mathcal{B}_{i-1}$ is the current structure and $r_i \in \mathcal{R}_{n,\sigma}$ a request.
So with input $\mathcal{B}_{i-1} = \langle \{0, 1, \ldots, n-1\}, M, b \rangle$ the query produces the updated structure $\mathcal{B}_i = \langle \{0, 1, \ldots, n-1\}, M', b' \rangle$ with the formulas *ins*, *del*.

REACH(*acyclic*)

The reachability problem in acyclic graphs aka REACH(*acyclic*) refers to the existence of an $s - t$ path in a directed acyclic graph (presuming that the graph remains acyclic after each request)

Theorem (PI97)

REACH(*acyclic*) $\in$ *Dyn-FO*

Basic idea: We need to evaluate boolean query $\mathcal{T} = \{\mathcal{B} \in \text{STRUCT}[\langle E^2, P^2, s, t\rangle] \mid \mathcal{B} \models P(s, t)\}$ by updating the path relation $\mathcal{P}$ and the edge relation E against every edge insertion or deletion in the graph using *FO* updates.

REACH(*acyclic*) cont'd

Proof.

We construct the following *FO* queries :

ins(E, a, b) :

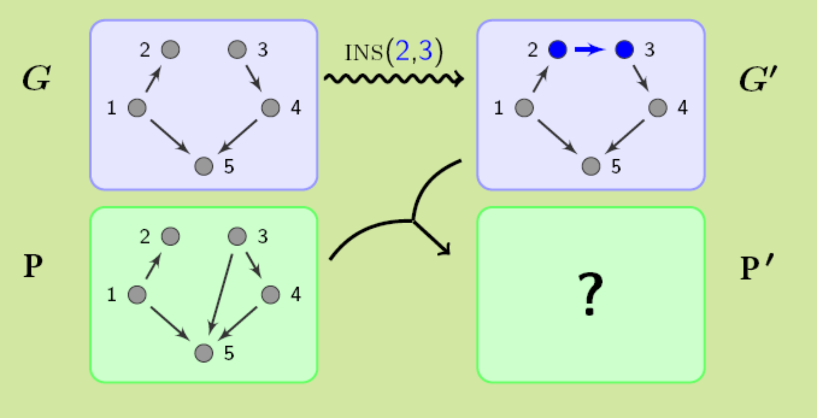
$$P'(x, y) \equiv P(x, y) \vee (P(x, a) \wedge P(b, y))$$

del(E, a, b) :

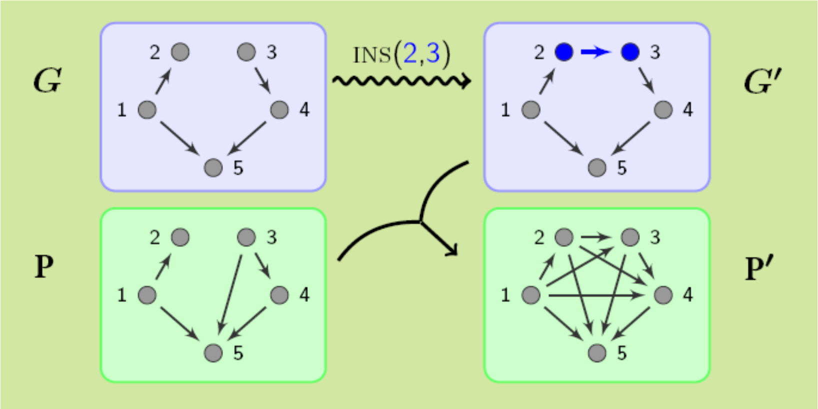
$$\begin{aligned} P'(x, y) \equiv P(x, y) \wedge [\neg P(x, a) \vee \neg P(b, y) \vee \\ (\exists u, v)(P(x, u) \wedge P(u, a) \wedge E(u, v) \wedge \neg P(v, a) \wedge P(v, y) \wedge \\ (v \neq b \vee u \neq a))] \end{aligned}$$

Since **ins,del** $\in$ *FO* $\Rightarrow$ REACH(*acyclic*) $\in$ *Dyn-FO*. □

REACH(*acyclic*) : How it works?



REACH(*acyclic*) : How it works?



REACH_u

The undirected reachability problem aka REACH_u is not *FO* expressible. But what about *Dyn-FO*?

Theorem (PI97)

$\text{REACH}_u \in \textit{Dyn-FO}$

Basic idea: We maintain a forest i.e. a collection of connected components for the undirected graph using three relations :

- $E(x, y)$ edge relation
- $F(x, y)$ edge in forest relation
- $PV(x, y, u)$ $x - y$ path via node u

REACH_u

The undirected reachability problem aka REACH_u is not *FO* expressible. But what about *Dyn-FO*?

Theorem (PI97)

$\text{REACH}_u \in \textit{Dyn-FO}$

Basic idea: We maintain a forest i.e. a collection of connected components for the undirected graph using three relations :

- $E(x, y)$ edge relation
- $F(x, y)$ edge in forest relation
- $PV(x, y, u)$ $x - y$ path via node u

We need to evaluate boolean query :

$$\mathcal{T} = \{\mathcal{A} \in \text{STRUCT}[\langle E^2, F^2, PV^3, s, t\rangle] \mid \mathcal{A} \models PV(s, t, t)\}$$

$\mathrm{REACH}_u \in$ *Dyn-FO*

Proof.

For our convenience we also define the following relations :

$$Eq(x, y, a, b) \equiv (x = a \wedge y = b) \vee (x = b \wedge y = a)$$
$$P(x, y) \equiv (x = y) \vee PV(x, y, y)$$

We need to handle *insertions* and *deletions* in a "*FO* way".

$$\begin{aligned} \textbf{ins}(E, a, b) : E'(x, y) &\equiv E(x, y) \vee Eq(x, y, a, b) \\ F'(x, y) &\equiv F(x, y) \vee (Eq(x, y, a, b) \wedge \neg P(a, b)) \\ PV'(x, y, z) &\equiv PV(x, y, z) \vee \\ &\quad (\exists u, v)[Eq(u, v, a, b) \wedge P(x, u) \wedge P(v, y) \\ &\quad \wedge (PV(x, u, z) \vee PV(v, y, z))] \end{aligned}$$

$\text{REACH}_u \in$ *Dyn-FO*

Proof.

For **del**(E, a, b) the trivial case is when $\neg F(a, b)$, where we only set $E'(a, b) = \mathit{false}$.
Otherwise, we define :

$$\begin{aligned}
T(x, y, z) &\equiv PV(x, y, z) \wedge \neg(PV(x, y, a) \wedge P(x, y, b)) \\
New(x, y) &\equiv E(x, y) \wedge T(a, x, a) \wedge T(b, y, b) \wedge \\
&\qquad (\forall u, v)[(E(u, v) \wedge T(a, u, a) \wedge \\
&\qquad T(b, v, b)) \rightarrow (x < u \vee (x = u \wedge y \leq v))]
\end{aligned}$$

$\text{REACH}_u \in \textit{Dyn-FO}$

Proof.

Finally we define E', F', PV' :

$$
\begin{aligned}
E'(x,y) &\equiv E(x,y) \wedge \neg Eq(x,y,a,b) \\
F'(x,y) &\equiv (F(x,y) \wedge \neg Eq(x,y,a,b)) \vee New(x,y) \vee New(y,x) \\
PV'(x,y,z) &\equiv T(x,y,z) \vee [(\exists u,v)(New(u,v) \vee New(v,u)) \wedge \\
&\qquad T(x,u,x) \wedge T(y,v,y) \wedge (T(x,u,z) \vee T(y,v,z))]
\end{aligned}
$$

□

Other known problems

From the introduction of *Dyn-FO* many problems have been proven to be *FO* computable using an auxiliary *FO* structure :

- $\text{REACH}_d \in \textit{Dyn-FO}$ using *FO* reduction to REACH_u
- $LCA \in \textit{Dyn-FO}$

$$LCA(a, x, y) \Leftrightarrow$$

$$P(a, x) \wedge P(a, y) \wedge (\forall z)((P(z, x) \wedge P(z, y)) \rightarrow P(z, a))$$

- All regular languages are in *Dyn-FO*

Dyn-PROP

Definition (*Dyn-PROP*)

Dyn-PROP is the set of all boolean queries that can be maintained using *quantifier-free FO* formulas after changes that affect a constant number of tuples in the input.

Dyn-PROP

Definition (*Dyn-PROP*)

Dyn-PROP is the set of all boolean queries that can be maintained using *quantifier-free FO* formulas after changes that affect a constant number of tuples in the input.

Example

Let G be a graph into which only edges' insertions are allowed. It is easy to see that using :

- an auxiliary relation T which shall contain all node pairs that are connected by a path in G
- a *FO* update formula $K_E^T(u, v, x, y) \equiv T(x, y) \vee (T(x, u) \wedge T(v, y))$

we verify the existence of an $s - t$ path without quantifiers in *FO*.

Other problems in *Dyn-PROP*

The absence of quantifiers reduces the expressibility of the class. However, *Dyn-PROP* contains problems that are not *FO* computable :

- PARITY $\in$ *Dyn-PROP*

  (if we recall the dynamic version of PARITY, we shall see that utilizes no quantifiers)
- $\text{REACH}_d \in$ *Dyn-PROP* [Hes03b]
- Regular languages are exactly those languages maintainable in *Dyn-PROP*! [GMS12]

What about reductions?

Reductions allow us to compare complexity classes and/or problems. There are many types of reductions e.g. *Turing*, *Karp*, *Cook*. We have also seen *FO* reductions i.e. a way of reducing problems in the descriptive context.

What about reductions?

Reductions allow us to compare complexity classes and/or problems. There are many types of reductions e.g. *Turing, Karp, Cook*. We have also seen *FO* reductions i.e. a way of reducing problems in the descriptive context.

It turns out that *FO* reductions are too powerful for Dynamic Complexity, so they need to be restricted somehow.

What about reductions?

Reductions allow us to compare complexity classes and/or problems. There are many types of reductions e.g. *Turing, Karp, Cook*. We have also seen *FO* reductions i.e. a way of reducing problems in the descriptive context.

It turns out that *FO* reductions are too powerful for Dynamic Complexity, so they need to be restricted somehow.

Definition (bfo)

Bounded expansion,FO reductions aka bfo are *FO* reductions that :

- each tuple/constant of the input structure affects constant tuples/constants of the output
- maps $\mathcal{A}_0^n$ to a structure of bounded tuples

As usual if S is reducible to T via bfo, we write $S \leq_{bfo} T$.

bfo Reductions cont'd

In the previous definition we imposed a limitation on the initial structure i.e. $\mathcal{A}_0^n$. If we allow *unbounded* initial tuple expansion, then we get *bfo*$^+$, essentially a variant of bfo that allows *precomputation*.

bfo Reductions cont'd

In the previous definition we imposed a limitation on the initial structure i.e. $\mathcal{A}_0^n$. If we allow *unbounded* initial tuple expansion, then we get bfo^+, essentially a variant of bfo that allows *precomputation*.

Example (REACH$_d \leq_{bfo}$ REACH$_u$)

Given a directed graph G we apply the following :

- Remove edges leaving t
- Remove edges from all other vertices so that they all have *outdegree* 1
- Mark remaining edges as undirected

and we call the produced graph G' which is undirected.

bfo example cont'd

Example ($\text{REACH}_d \leq_{bfo} \text{REACH}_u$)

Express the previous steps using *FO*:

$\boldsymbol{I_{d-u}} = \lambda_{xy}(\phi_{d-u}, s, t)$:

$$a(x,y) \equiv E(x,y) \wedge x \neq t \wedge (\forall z)(E(x,z) \rightarrow z = y)$$
$$\phi_{d-u}(x,y) \equiv a(x,y) \vee a(y,x)$$

Now we have an $s-t$ deterministic path in G iff there is an $s-t$ path in G'.

The reduction above is obviously *FO*; but it is also *bfo*! Why?

bfo example cont'd

Example ($\text{REACH}_d \leq_{bfo} \text{REACH}_u$)

Express the previous steps using *FO*:

$\boldsymbol{I_{d-u}} = \lambda_{xy}(\phi_{d-u}, s, t)$:

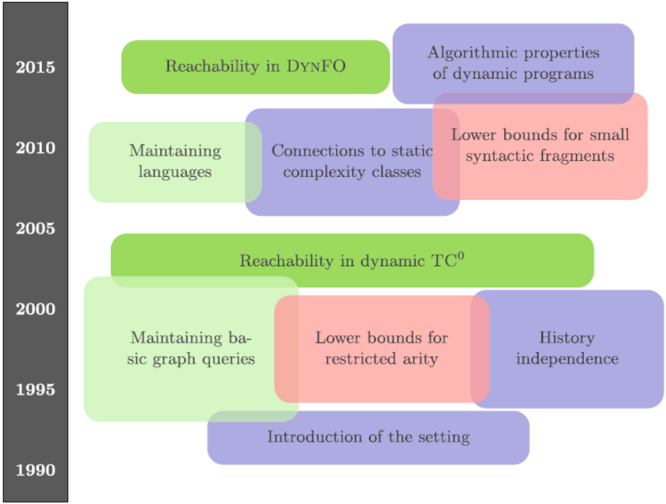
$$a(x,y) \equiv E(x,y) \wedge x \neq t \wedge (\forall z)(E(x,z) \rightarrow z = y)$$

$$\phi_{d-u}(x,y) \equiv a(x,y) \vee a(y,x)$$

Now we have an $s-t$ deterministic path in G iff there is an $s-t$ path in G'.

The reduction above is obviously *FO*; but it is also *bfo*! Why? Each request **ins** or **del** in G, causes at most two edges to be inserted or deleted.

Historical overview



About REACH

Among graph queries, REACH is probably the most studied query.

- (1995) By the introduction of the Dynamic Complexity Framework, it was known that $\text{REACH}_u, \text{REACH}_d \in \textit{Dyn-FO}$
- (2003) Hesse showed $\text{REACH} \in \textit{Dyn-TC}^0$ (AC^0 but with maj. gates)
- (2015) Datta et al. showed that $\text{REACH} \in \textit{Dyn-FO}$
- (2018) extended for changes of size $\frac{\log n}{\log \log n}$
- (04/2020) extended for polylogarithmically sized changes for REACH_u,REACH_d

Overview

Synopsis

Today we've seen :

- Why static complexity fails to capture certain problems' aspects?
- The expansion of Descriptive Complexity framework to capture dynamic problems
- A general definition of *Dyn*-$\mathcal{C}$
- Representative examples of Dynamic Complexity classes
- Known problems expressed with Dynamic Complexity
- Historical overview of the area

References

- Immerman, Neil. "Descriptive complexity" (1999) : 221-234
- Patnaik, Sushant & Immerman, Neil. "Dyn-FO: A Parallel Dynamic Complexity Class" (1997)
- Zeume, Thomas. "An Update on Dynamic Complexity Theory" (2018)

The end

Fin

How many light bulbs does it take to change a light bulb? One, if it knows its own Gödel number!